Fine-grained I/O Complexity via Reductions: *New lower bounds, faster algorithms, and a time hierarchy*

Erik D. Demaine[1], Andrea Lincoln[1], Quanquan C. Liu[1], Jayson Lynch[1], and Virginia Vassilevska Williams[1]

1 **Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA, USA**
{edemaine,andreali,quanquan,jaysonl,virgi}@mit.edu

Abstract

This paper initiates the study of I/O algorithms (minimizing cache misses) from the perspective of fine-grained complexity (conditional polynomial lower bounds). Specifically, we aim to answer why sparse graph problems are so hard, and why the Longest Common Subsequence problem gets a savings of a factor of the size of cache times the length of a cache line, but no more. We take the reductions and techniques from complexity and fine-grained complexity and apply them to the I/O model to generate new (conditional) lower bounds as well as faster algorithms. We also prove the existence of a time hierarchy for the I/O model, which motivates the fine-grained reductions.

- Using fine-grained reductions, we give an algorithm for distinguishing 2 vs. 3 diameter and radius that runs in $O(|E|^2/(MB))$ cache misses, which for sparse graphs improves over the previous $O(|V|^2/B)$ running time.
- We give new reductions from radius and diameter to Wiener index and median. These reductions are new in both the RAM and I/O models.
- We show meaningful reductions between problems that have linear-time solutions in the RAM model. The reductions use low I/O complexity (typically $O(n/B)$), and thus help to finely capture the relationship between "I/O linear time" $\Theta(n/B)$ and RAM linear time $\Theta(n)$.
- We generate new I/O assumptions based on the difficulty of improving sparse graph problem running times in the I/O model. We create conjectures that the current best known algorithms for Single Source Shortest Paths (SSSP), diameter, and radius are optimal.
- From these I/O-model assumptions, we show that many of the known reductions in the word-RAM model can naturally extend to hold in the I/O model as well (e.g., a lower bound on the I/O complexity of Longest Common Subsequence that matches the best known running time).
- We prove an analog of the Time Hierarchy Theorem in the I/O model, further motivating the study of fine-grained algorithmic differences.

9th Innovations in Theoretical Computer Science Conference (ITCS 2018).
Editor: Anna R. Karlin; Article No. 34; pp. 34:1–34:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The I/O model (or external-memory model) was introduced by Aggarwal and Vitter [5] to model the non-uniform access times of memory in modern processors. The model nicely captures the fact that, in many practical scenarios, cache misses between levels of the memory hierarchy (including disk) are the bottleneck for the program. As a result, the I/O model has become a popular model for developing cache-efficient algorithms.

In the I/O model, the expensive operation is bringing a *cache line* of B contiguous words from the "main memory" (which may alternately represent disk) to the "cache" (local work space). The cache can store up to M words in total, or M/B cache lines. Computation on data in the cache is usually treated as free, and thus the main goal of I/O algorithms is to access memory with locality. That is, when bringing data into cache from main memory in contiguous chunks, we would like to take full advantage of the fetched cache line. This is preferable to, say, randomly accessing noncontiguous words in memory.

When taking good graph algorithms for the RAM model and analyzing them in the I/O model, the running times are often very bad. Take for example Dijkstra's algorithm or the standard BFS algorithm. These algorithms fundamentally look at an adjacency list, and follow pointers to every adjacent node. Once the new node is reached, the process repeats, accessing all the adjacent nodes in priority order that have not previously been visited. This behavior looks almost like random access! Unless one can efficiently predict the order these nodes will be reached, the nodes will likely be stored far apart in memory. Even worse, this optimal order could be very different depending on what node one starts the algorithm at.

Because of this bad behavior, I/O-efficient algorithms for graph problems take a different approach. For dense graphs, one approach is to reduce the problems to matrix equivalent versions. For example, APSP is solved by $(\min, +)$ matrix multiplication [34, 37]. The locality of matrices leads to efficient algorithms for these problems.

Unfortunately, sparse graph problems are not solved efficiently by $(\min, +)$ matrix multiplication. For example, the best algorithms for directed single-source shortest paths in sparse graphs take $O(n)$ time, giving no improvement from the cache line at all [16, 18, 8]. Even in the undirected case, the best algorithm takes $O(n/\sqrt{B})$ time in sparse graphs [32].

The Diameter problem in particular has resisted improvement beyond $O(|V|/\sqrt{B})$ even in undirected unweighted graphs [30], and in directed graphs, the best known algorithms still run in time $\Omega(|V|)$ [17, 8]. For this reason, we use this as a conjecture and build a network of reductions around sparse diameter and other sparse graph problems.

In this paper we seek to explain why these problems, and other problems in the I/O model, are so hard to improve and to get faster algorithms for some of them.

In this paper we use reductions to generate new algorithms, new lower bounds, and a time hierarchy in the I/O model. Specifically, we get new algorithms for computing the diameter and radius in sparse graphs, when the computed radii are small. We generate novel reductions (which work in both the RAM and I/O models) for the Wiener Index problem (a graph centrality measure). We generate further novel reductions which are meaningful in the I/O model related to sparse graph problems. Finally, we show that an I/O time hierarchy exists, similar to the classic Time Hierarchy Theorem.

1.1 Caching Model and Related Work.

Cache behavior has been studied extensively. In 1988, Aggarwal and Vitter [5] developed the *I/O model*, also known as the external-memory model [25], which now serves as a theoretical formalization for modern caching models. A significant amount of work on algorithms and data structures in this model has occurred including items like buffer-trees [7], B-trees [13], permutations and sorting [5], ordered file maintenance [14], $(\min, +)$ matrix multiplication [34], and triangle listing [33]. Frigo, Leiserson, Prokop and Ramachandran [27] proposed the *cache-oblivious model*. In this model, the algorithm is not given access to the cache size M, nor is it given access to the cache-line size B. Thus the algorithm must be oblivious to the cache, despite being judged on its cache performance. Some surveys of the work include [42, 6, 25].

When requesting cache lines from main memory in this paper we will only request the B words starting at location xB for integers x. Another common model (which we do not follow in this paper)

allows arbitrary offsets for the cache line pulls. This can be simulated with at most twice as many cache misses and twice as much cache.

1.2 Fine-grained Complexity.

A popular area of recent study is fine-grained complexity. The field uses efficient reductions to uncover relationships between problems whose classical algorithms have not been improved substantially in decades. Significant progress has been made in explaining the lack of progress on many important problems [10, 44, 2, 3, 15, 1, 36] such as APSP, othogonal vectors (OV), 3-SUM, longest common subsequence (LCS), edit distance and more. Such results focused on finding reductions from these problems to other (perhaps less well-studied) problems such that an improvement in the upper bound on any of these problems will lead to an improvement in the running time of algorithms for these problems. For example, research around the All-Pairs Shortest Paths problem (APSP) has uncovered that many natural, seemingly simpler graph problems on n node graphs are fine-grained equivalent to APSP, so that an $O(n^{3-\varepsilon})$ time algorithm for $\varepsilon > 0$ for one of the problems implies an $O(n^{3-\varepsilon'})$ time algorithm for some $\varepsilon' > 0$ for all of them.

1.3 History of Upper Bounds

In the I/O model, the design of algorithms for graph problems is difficult. This is demonstrated by the number of algorithms designed for problems like Sparse All Pairs Shortest Paths, Breath First Search, Graph Radius and Graph Diameter where very minor improvements are made (see Table 1 for definitions of problems). Note that dense and sparse qualifiers in the front of problems indicate the problem defined over a dense/sparse graph, respectively.

The Wiener index problem measures the total distance from all points to each other. Intuitively, this measures how close or far points in the graph are from each other. In this respect, Wiener index is similar to the radius, diameter and median measures of graph distance.

The history of improvements to the upper bound of negative triangle in the I/O model is an important example of the difficulty in the design of I/O efficient algorithms for graph problems (see Table 2 for a summary). For a long time, no improvements in terms of M were made to the upper bound for negative triangle. A key was re-interpreting the problem as a repeated scan of lists.

We feel that the history of negative triangle has taught us that upper bounds in the I/O model on graph problems are best achieved by creating efficient reductions from a graph problem to a non-graph problem. Hence the study of fine-grained reductions in the I/O model is crucial to using this approach in solving such graph problems with better I/O efficiency. Graph problems tempt the algorithmic designer into memory access patterns that look like random access, whereas matrix and array problems immediately suggest memory local approaches to these problems. We consider the history of the negative triangle problem to be an instructive parable of why the matrix and array variants are the right way to view I/O problems.

1.4 Our Results

We will now discuss our results in this paper. In all tables in this paper, our results will be in bold. We demonstrate the value of reductions as a tool to further progress in the I/O model through our results.

Our results include improved upper bounds, a new technique for lower bounds in the I/O model and the proof of a computational hierarchy. Notably, in this paper we tie the I/O model into both fine-grained complexity and classical complexity.

1.4.1 Upper Bounds

We get improved upper bounds on two sparse graph problems and have a clarifying note to the community about matrix multiplication algorithms.

For both the sparse 2 vs 3 diameter and sparse 2 vs 3 radius problems, we improve the running time from $O(n^2/B)$ to $O(n^2/(MB))$. We get these results by using an insight from a pre-existing reduction to two very local problems which have trivial $O(n^2/(MB))$ algorithms that solve them. Note that this follows the pattern we note in Section 1.3 in that we produce a reduction from a graph problem to a non-graph problem to obtain better upper bounds in terms of M.

Furthermore, previous work in the I/O model related to matrix multiplication seems to use the naive matrix multiplication n^3 bound, or the Strassen subdivision. However, fast matrix multiplication

Problem Name	Problem Definition
Orthogonal Vector (OV)	Given two sets U and V of n vectors each with elements $\{0,1\}^d$ where $d = \omega(\log n)$, determine whether there exist vectors $u \in U$ and $v \in V$ such that $\sum_{i=1}^{d} u_i \cdot v_i = 0$.
Longest Common Subsequence (LCS)	Given two strings of n symbols over some alphabet Σ, compute the length of the longest sequence that appears as a subsequence in both input strings.
Edit Distance (ED)	Given two strings s_1 and s_2, determine the minimum number of operations that converts s_1 to s_2.
Sparse Diameter	Given a sparse graph $G = (V, E)$, determine if $\max_{u,v \in V} d(u, v)$ where $d(u, v)$ is the distance between nodes u and v in V.
2 vs. 3 Sparse Diameter	Given a sparse graph $G = (V, E)$, determine if $\max_{u,v \in V} d(u, v) \leq 2$ where $d(u, v)$ is the distance between nodes u and v in V.
Hitting Set (HS)	Given two lists of size n, V and W, where the elements are taken from a universe U, does there exist a set in V that hits (contains an element of) every set in W.
Sparse Radius	Given a sparse graph $G = (V, E)$, determine $\min_{u \in V} (\max_{v \in V} d(u, v))$ where $d(u, v)$ is the distance between nodes u and v in V.
2 vs. 3 Sparse Radius	Given a sparse graph $G = (V, E)$, determine if $\min_{u \in V} (\max_{v \in V} d(u, v)) \leq 2$ where $d(u, v)$ is the distance between nodes u and v in V.
3 vs. 4 Sparse Radius	Given a sparse graph $G = (V, E)$, determine if $\min_{u \in V} (\max_{v \in V} d(u, v)) \leq 3$ where $d(u, v)$ is the distance between nodes u and v in V.
Median	Let $d(u, v)$ be the shortest path distance between nodes u and v in a graph G. The median is the node v that minimizes the sum $\sum_{u \in V} d(v, u)$.
3-SUM	Given a set of n integers, determine whether the set contains three integers a, b, c such that $a + b = c$.
Convolutional 3-SUM	Given three lists A, B and C each consisting of n numbers, return true if $\exists i, j, k \in [0, n-1]$ such that $i + j + k \equiv 0$ **mod** n and $A[i] + B[j] + C[k] = 0$.
0 Triangle	Given a graph G, return true if $\exists a, b, c \in V$ such that $w(a, b) + w(b, c) + w(c, a) = 0$ where $w(u, v)$ is the weight of the edge (u, v).
All-Pairs Shortest Paths (APSP)	Given a directed or undirected graph with integer weights, determine the shortest distance between all pairs of vertices in the graph.
Wiener Index	Let $d(u, v)$ be the shortest paths distance between nodes u and v in a graph G. The Wiener Index of G is $\sum_{u \in V} \sum_{v \in V} d(u, v)$.
Negative Traingle	Given a graph G, return true if $\exists a, b, c \in V$ such that $w(a, b) + w(b, c) + w(c, a) < 0$ where $w(u, v)$ is the weight of the edge (u, v).
$(\min, +)$-Matrix Multiplication	Given two n by n matrices A and B, return $C[i, k] = \min_{j \in [1,n]} (A[i, j] + B[j, k])$.
Sparse Weighted Diameter	Given a sparse, weighted graph $G = (V, E)$, determine $\max_{u,v \in V} d(u, v)$ where $d(u, v)$ is the distance between nodes u and v in V.

Table 1 Fine-grained problems definitions.

Dense APSP		Sparse Weighted Diameter		Sparse $-\triangle$	
$\tilde{O}(n^3)$	[naive]	$\tilde{O}(n^2)$	[naive]	$\tilde{O}(n^{1.5})$	[naive]
$\tilde{O}(n^3/(\sqrt{M}))$	[29]	$\tilde{O}(n^2/\sqrt{B})$	[9]	$\tilde{O}(n+n^{1.5}/B)$	[31]
$\tilde{O}(n^3/(\sqrt{M}B))$	Extension*	$\tilde{O}(n^2/\sqrt{B})$	[19]	$\tilde{O}(n^{1.5}/B)$	[26]
				$\tilde{O}(n^{1.5}/(\sqrt{M}B))$	[33]

Table 2 History of APSP upper bounds. * This is an extension of [29] see e.g. [33].

algorithms which runs in n^ω time imply a nice self-reduction. Thus, we can get better I/O algorithms which run in the most recent fast matrix multiplication time. We want to explicitly add a note in the literature that fast matrix multiplication in the I/O model should run in time $T_{MM}(n,M,B) = O(n^{\omega'}/(M^{\omega'/2-1}B))$ where ω' is the matrix multiplication exponent, if it is derived using techniques bounding the rank of the matrix multiplication tensor. The current best ω' is $\omega' < 2.373$ [41, 28] giving us the I/O running time of $T_{MM}(n,M,B) = O(n^{2.373}/(M^{0.187}B))$. We give these results in Section 2.3.

1.4.2 I/O model Conjectures

In the I/O model a common way to get upper bounds is to get a self-reduction where a large problem is solvable by a few copies of a smaller problem. We make the small subproblems so small they fit in cache. If the problem is laid out in a memory local fashion in main memory then it will take M/B I/Os to solve a subproblem that fits in memory M.

In Section 2.2, we give an I/O-based Master Theorem which gives the running time for algorithms with recurrences of the form $T(n,M,B) = \alpha T(n/\beta,M,B) + f(n,M,B)$ (like the classic Master Theorem from [24]) and $T(n,M,B) = g^2 T(n/\beta,M,B) + f(n,M,B)$ (self-reduction). The running times generated by these recurrences match the best known running times of All-Pairs Shortest Paths (APSP), 3-SUM, Longest Common Subsequence (LCS), Edit Distance, Orthogonal Vectors (OV), and more. Thus, if we conjecture that a recursive algorithm has a running time that is optimal for a problem, we are able to transfer this bound over to the I/O model using our Master Theorem and self-reduction framework in a natural way.

Lower Bounds From Fine-Grained Complexity Assumptions.

We demonstrate that many of the reductions in the RAM model between problems of interest and common fine-grained assumptions give lower bounds in the I/O model. We generate reasonable I/O conjectures for these problems and demonstrate that the reductions are I/O-efficient. First, we begin with the conjectures.

▶ Conjecture 1 (I/O All-Pairs Shortest Paths (APSP) Conjecture). APSP requires $\frac{n^{3-o(1)}}{M^{1/2+o(1)}B^{1+o(1)}}$ I/Os.

▶ Conjecture 2 (I/O 3-SUM Conjecture). 3-SUM requires $\frac{n^{2-o(1)}}{M^{1+o(1)}B^{1+o(1)}}$ I/Os.

▶ Conjecture 3 (I/O Orthogonal Vectors (OV) Conjecture). OV requires $\frac{n^{2-o(1)}}{M^{1+o(1)}B^{1+o(1)}}$ I/Os.

▶ Conjecture 4 (I/O Hitting Set (HS) Conjecture). HS requires $\frac{n^{2-o(1)}}{M^{1+o(1)}B^{1+o(1)}}$ I/Os.

From these conjectures we can generate many lower bounds. Many of our lower bounds are tight to the fastest known algorithms. These reductions have value even if the conjectures are refuted since many of these reductions also give upper bounds for other problems–leading to better algorithms for many problems even if the conjectures are refuted.

Problem	Upper Bound	UB source	Lower Bound	LB from	LB source
OV	$\tilde{O}(n^2/(MB))$	**Lem 49**	$\tilde{\Omega}(n^2/(MB))$	IO OV Conj	By Def
LCS	$\tilde{O}(n^2/(MB))$	[20]	$\tilde{\Omega}(n^2/(MB))$	IO OV Conj	**Lem 52**
Edit Distance	$\tilde{O}(n^2/(MB))$	[21]	$\tilde{\Omega}(n^2/(MB))$	IO OV Conj	**Lem 51**
Sparse Diameter	$\tilde{O}(n^2/B)$	[9]	$\tilde{\Omega}(n^2/(MB))$	IO OV Conj	**Lem 50**
2 vs. 3 Sprs. Diameter	$\tilde{O}(n^2/(MB))$	**Full Paper**	$\tilde{\Omega}(n^2/(MB))$	IO OV Conj	**Lem 50**
Hitting Set	$\tilde{O}(n^2/(MB))$	**Lem 54**	$\tilde{\Omega}(n^2/(MB))$	IO HS Conj	By Def.
Sparse Radius	$\tilde{O}(n^2/B)$	[9]	$\tilde{\Omega}(n^2/(MB))$	IO HS Conj	**Lem 55**
2 vs. 3 Sparse Radius	$\tilde{O}(n^2/(MB))$	**Full Paper**	$\tilde{\Omega}(n^2/(MB))$	IO HS Conj	**Lem 55**
3 vs. 4 Sparse Radius	$\tilde{O}(n^2/B)$	[9]	$\tilde{\Omega}(n^2/(MB))$	IO HS Conj	**Lem 55**
Sparse Median	$\tilde{O}(n^2/B)^a$	[9]	$\tilde{\Omega}(n^2/(MB))$	IO HS Conj	**Thm 17**
Sparse Median	$\tilde{O}(n^2/B)^b$	[9]	$\tilde{\Omega}(n^2/B)$	3 vs. 4 Sprs. Radius	**Thm 17**
3-SUM	$\tilde{O}(n^2/(MB))$	[12]	$\tilde{\Omega}(n^2/(MB))$	IO 3-SUM Conj	By Def
Conv. 3-SUM	$\tilde{O}(n^2/(MB))^c$	[12]	$\tilde{\Omega}(n^2/(MB))$	IO 3-SUM Conj	**Lem 30**
0 Triangle	$\tilde{O}(n^3/(\sqrt{M}B))$	**Lem 48**	$\tilde{\Omega}(n^2/(MB))$	IO 3-SUM Conj	**Thm 33**
APSP	$\tilde{O}(n^3/(\sqrt{M}B))$	[34]	$\tilde{\Omega}(n^2/(\sqrt{M}B))$	IO APSP Conj	By Def
Wiener Index	$\tilde{O}(n^3/(\sqrt{M}B))^d$	[34]	$\tilde{\Omega}(n^2/(\sqrt{M}B))$	IO APSP Conj	**Thm 18**
0 Triangle	$\tilde{O}(n^3/(\sqrt{M}B))$	**Lem 48**	$\tilde{\Omega}(n^2/(\sqrt{M}B))$	IO APSP Conj	**Thm 47**
$-$ Triangle	$\tilde{O}(n^3/(\sqrt{M}B))$	**Thm 45**	$\tilde{\Omega}(n^2/(\sqrt{M}B))$	IO APSP Conj	**Thm 40**
(min,+) MM	$\tilde{O}(n^3/(\sqrt{M}B))$	[34]	$\tilde{\Omega}(n^2/(\sqrt{M}B))$	IO APSP Conj	**Thm 40**

[a] Upper bound comes from APSP in general graphs.
[b] Upper bound comes from APSP in general graphs.
[c] The upper bound is a trivial extension of the 3-SUM upper bound for explanation see Lemma 34.
[d] This upper bound comes directly from applying the algorithm for APSP, solving APSP, and summing the results.

Table 3 Previous results and our results on upper and lower bounds of problems. Sparse (Sprs.) means that $|E| = O(|V|)$.

1.4.3 Lower Bounds from Sparse Graph Problems

In addition to the upper, lower bounds, and reductions presented in the I/O model for the standard RAM problems listed in Table 3, we introduce novel upper, lower bounds, and reductions between graph problems. The reason for this focus is the fact that, more than in the RAM model, the I/O model has a history of particularly slow algorithms in graphs. In particular, sparse graph problems have very slow algorithms. We make novel reductions between sparse graph problems, many of which apply to the RAM model as well, such that solving one of these problems will solve many other variations of hard sparse graph problems in the I/O model.

We provide reductions between problems that currently require $\Omega(n/\sqrt{B})$ time to solve. Thus, these problems specifically require linear time reductions. We show equivalence between the following set of problems for undirected/directed and unweighted graphs: (s,t)-shortest path, finding the girth through an edge, and finding the girth through a vertex.

We additionally generate a new reduction from sparse weighted Diameter to the sparse Wiener Index problem in Section 3.1. This reduction holds in the RAM model as well as the I/O model.

1.4.4 Hierarchy

The time and space hierarchy theorems are fundamental results in computational complexity that tell us there are problems which can be solved on a deterministic Turing Machine with some bounded time or space, which cannot be solved on a deterministic Turing Machine which has access to less time or space. See notably the famous time and space hierarchies [38]. For some classes, for example BPP, no time hierarchy is known to exist (e.g., [43, 11]).

In Section 5.3, we show similar separation hierarchies exist in the I/O model once again using the simulations between the RAM and I/O models and our complexity class $\mathit{CACHE}_{M,B}(t(n))$ defined in Section 5.2 as the set of problems solvable in $O(t(n))$ cache misses.

▶ **Theorem 5.** *If the memory used by the algorithm is referenceable by $O(B)$ words (i.e. the entire input can be brought into cache by bringing in at most $O(B)$ words), then*

$$\mathit{CACHE}_{M,B}(t(n)) \subsetneq CACHE_{M,B}((t(n)\,B)^{1+\varepsilon}).$$

Notably, this theorem applies any time we use a polynomially size memory and our word size is $w = \Omega(\lg n)$, which is the standard case in the RAM model.

This separation is motivation for looking at complexity of specific problems and trying to understand what computational resources are necessary to solve them.

1.4.5 Improved TM Simulations of RAM Imply Better Algorithms

In the full version, we show that improved simulations of RAM machines by Turing Machines would imply better algorithms in the I/O model. Specifically, if we can simulate RAM more efficiently with either multi-tape Turing machines or multi-dimensional Turing machines, then we can show that we can gain some cache locality and thus save by some factor of B, the cache line size.

1.5 Organization

In this paper, we argue that the lens of reductions offer a powerful way to view the I/O model. We show that reductions give novel upper and lower bounds. We also define complexity classes for the I/O model and prove a hierarchy theorem further motivating the analysis of the I/O model using fine-grained complexity.

We begin with faster algorithms obtained through reductions which are collected in Section 2. Section 2.1 develops such algorithms for small diameter and radius. Section 2.2 develops the I/O Master Theorem, which is more broadly a useful tool for analyzing almost all cache-oblivious algorithms. Section 2.3 uses this theorem to show how all recent improvements to matrix multiplication's RAM running time also give efficient cache-oblivious algorithms.

One can get new lower bounds, by using the techniques from fine-grained complexity. Some fine-grained reductions from the RAM model also work in the I/O model, we show examples in Sections 4.1, 4.2, 4.3. We also get new reductions that work in both the RAM and I/O model related

to the Wiener Index problem in Section 3.1. Some reductions in the RAM model do not work in the I/O model; thus, in Section 3.1, we give novel reductions between several algorithms which take $O(n^2/B)$ and $O(n^2/(MB))$ time. We also get reductions that are meaningful in the I/O model which are not in the RAM model, notably, between problems whose fastest algorithms are $O(n/\sqrt{B})$ and $O(n)$, respectively, in Section 3.2.

One can also use reductions and simulation arguments to prove a hierarchy theorem for the I/O model, explained in Section 5.

2 Algorithms in the I/O Model

In this section, we discuss our improved algorithms, algorithm analysis tools, and how reductions generate algorithms. As is typical in the I/O model, we assume that all inputs are stored in disk and any computation done on the inputs are done in cache (after some or all of the inputs are brought into cache). Section 2.1 gives better algorithms for the 2 vs 3 Diameter problem and the 2 vs 3 Radius problem in the I/O model. Section 2.3 gives improved algorithms for Matrix Multiplication in the I/O model.

Self-reductions are commonly used for cache-oblivious algorithms, because dividing until the subproblems are arbitrarily small allows for the problems to always fit in cache. In the RAM model, self-reductions allow for easy analysis via the Master Theorem. Despite the amount of attention to analyzing self-reductions in the I/O model, no one has written down the I/O-based Master Theorem. In Section 2.2, we describe and prove a version of the Master Theorem for the I/O model. We present a proof of this theorem to simplify our analysis and to help future papers avoid redoing this analysis.

Finally, in the full version of the paper we explain how some reductions in the RAM model imply faster algorithms in the I/O model.

2.1 Algorithms for Sparse 2 vs 3 Radius and Diameter

For both the radius and diameter problems on unweighted and undirected graphs, we can show distinguishing between a graph with a diameter or radius of 2 and one with a larger diameter or radius can be solved efficiently. Our algorithm relies on the reinterpretation of the 2 vs 3 problem as a set-disjointness problem. Every node, v, has an associated set, S_v, its adjacency list union itself. If two nodes have disjoint sets S_v and S_u, then they are distance greater than 2 from each other. Our algorithm for 2 vs 3 diameter and radius save an entire factor of M from the previously best known running times.

This is a similar idea to the reduction from 2 vs 3 diameter to OV and from 2 vs 3 radius to Hitting Set in the RAM model. These reductions were introduced by Abboud, Vassilevska-Williams and Wang [4]. While these reductions exist in the RAM model, they don't result in faster algorithms for 2 vs 3 diameter and radius in the I/O model because they use a hashing step that results in BFS being run from $\frac{|E|}{\Delta}$ nodes for some parameter Δ that can be set that gives the orthogonal vectors instance a dimension of Δ^2. In the I/O model, BFS is quite inefficient: we would need to set $\Delta \approxeq M$ to get an efficient algorithm using the approach in [4]. But, with a dimension of M^2 the OV algorithm will run very slowly. Therefore, below we present a solution to the set disjointness problem with no hashing into a smaller dimension.

Below we present the cache-aware algorithm for distinguishing 2 vs 3 diameter in an undirected, unweighted graph which runs in $O\left(\frac{|E|^2}{MB} + sort(|E|)\right)$ time where $sort(|E|)$ is the time to sort the elements in $|E|$ in the I/O model and $sort(|E|) = O\left(\frac{|E|\log|E|}{B}\right)$. We leave the proofs of cache-oblivious 2 vs 3 diameter and radius in the full version . For both 2 vs 3 diameter and radius we get running times of $O\left(\frac{n^2}{MB} + \frac{|E|\lg(|E|)}{B}\right)$. The algorithms and proofs for cache-obliviousness are finicky, but fundamentally are self-reductions of the form $T(n) = 4T(n/2) + n/B$. We leave the proofs to the full version of the paper.

We will start by giving a non-oblivious algorithm which relies on a recursive self-reduction. We will then show how to make this oblivious. It is easier to explain the analysis and algorithm when we can rely on the size of cache, but we can avoid that and get an oblivious algorithm anyway. The previous best algorithm is from Arge, Meyer and Toma which achieves $O(|V|sort(|E|)) =$

$O\left(\frac{|V||E|\log|E|}{B}\right)$ [9]. We get an improvement over the previous algorithm in terms of running time whenever $\frac{|E|}{|V|} = o(M)$.

▶ **Theorem 6.** *Determining if the diameter of an undirected, unweighted graph is 1, 2 or greater than 2 can be done in $O\left(\frac{|E|^2}{MB} + sort(|E|)\right)$ time in the I/O-model.*

See full version of the paper for proof.

The proofs of cache-oblivious 2 vs 3 Diameter and 2 vs 3 Radius are included in the full version .

2.2 Master Theorem in the I/O Model

In this section, we formally define our Master Theorem framework for the I/O model and provide bounds on the I/O complexity of problems whose I/O complexity fits the specifications of our framework. In addition, we also describe some example uses of our Master Theorem for the I/O model.

The Master Theorem recurrence in the RAM model looks like $T(n) = aT(n/b) + f(n)$. We will use a similar recurrence but all functions will now be defined over n, M and B. The I/O-Master Theorem function $f(n, M, B)$ includes all costs that are incurred in each layer of the recursive call. This includes the I/O complexity of reading in an input, processing the input, processing the output and writing out the output. In this section, we assume that $f(n, M, B)$ is a monotonically increasing function in terms of n in order to apply our Master Theorem framework. What this means is that for any fixed M and B we want the number of I/Os to increase or stay the same as n increases. Given that $f(n, M, B)$ specifies the I/O complexity of reading in the inputs and writing out the outputs, we prove the following version of the Master Theorem in the I/O model.

▶ **Theorem 7** (I/O Master Theorem)**.** *If $f(n, M, B)$ contains the cost of reading in the input (for each subproblem) and writing output (after computation of each subproblem), then the following holds. Given a recurrence of $T(n, M, B) = \alpha T(n/\beta, M, B) + f(n, M, B)$, where $\alpha \geq 1$ and $\beta > 1$ are constants, and a base case of $T(n/x, M, B) = t(x, M, B)$ (where $t(x, M, B) = \Omega(1)$) for some $x \leq n$ and some function $t(x, M, B)$. Let $A(n, M, B) = \left(\frac{n}{x}\right)^{\log_\beta(\alpha)} t(x, M, B)$, $C(n, M, B) = \left(\frac{n}{x}\right)^{\log_\beta(\alpha)+\varepsilon_1} t(x, M, B)$ and $D(n, M, B) = \left(\frac{n}{x}\right)^{\log_\beta(\alpha)-\varepsilon_2} t(x, M, B)$ for some $\varepsilon_1, \varepsilon_2 > 0$, then we get the following cases:*

Case 1: *If $f(n, M, B) = O\left(D(n, M, B)\right)$ then $T(n) = \Theta\left(A(n, M, B) + \frac{n}{B}\right)$.*

Case 2: *If $C(n, M, B) = O\left(f(n, M, B)\right)$ and $\alpha T(n/\beta, M, B) \leq cf(n, M, B)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta\left(f(n, M, B) + \frac{n}{B}\right)$.*

Case 3: *If $f(n, M, B) = \Theta(A(n, M, B))$, then $T(n, M, B) = \Theta(f(n, M, B)\log\left(\frac{n}{x}\right) + \frac{n}{B})$.*

Case 4: *If $f(n, M, B)$ has a constant number of terms, $f(n, M, B) = \Omega\left(\frac{n}{B}\right)$, and none of the previous cases are satisfied, then $T(n, M, B) = O\left(A(n, M, B) + f(n, M, B)\left(\frac{n}{x}\right)^{\log_\beta \alpha} + \frac{n}{B}\right)$ (note that this includes if A and f are incomparable), with tighter upper bounds provided in our proof (in the full version of the paper) depending on characteristics of the actual function, $f(n, M, B)$.*

See full version of the paper for proof.

One-Layer Self-Reductions

We state a relationship between one-layer self-reductions and our Master Theorem framework above. We refer to the process of solving a problem by reducing to several problems of smaller size each of which can be solved in cache and one recursive call is necessary as a *one-layer self-reduction*. Suppose the runtime of an algorithm in the RAM model is $n^{\log_\beta \alpha}$, then by dividing the problems into $\frac{n^{\log_\beta \alpha}}{M}$ subproblems each of which takes M/B I/Os to process, the I/O complexity of the algorithm is $\Theta\left(\frac{n^{\log_\beta \alpha}}{M^{\log_\beta \alpha - 1}B}\right)$ which is the same result we obtain via our Master Theorem framwork above when $t(x, M, B) = M/B$.

We now prove formally the theorem related to one-layer self-reductions.

▶ **Theorem 8.** *Let P be a problem of size n which can be reduced to $g(n/M)$ sub-problems, each of which takes $T(M, M, B)$ I/Os to process. The runtime of such a one-layer self reduction for the problem P is $T(n, M, B) = g(n/M)T(M, M, B) + f(n, M, B)$ where $f(n, M, B) = \Omega\left(\frac{n}{B}\right)$.*

See full version of the paper for proof.

2.3 Faster I/O Matrix Multiplication via I/O Master Theorem

As we mentioned above, any I/O algorithm that has a self-reduction to one of the forms stated in Section 2.2. Using our I/O Master Theorem, we can show a comparable I/O matrix multiplication bound to the matrix multiplication bound based on finding the rank of the Matrix Multiplication Tensor in the RAM model.

Recent improvements to matrix multiplication's running time also imply faster cache oblivious algorithms. Recent work has improved the bounds on ω where ω is the constant such that for any $0 < \varepsilon < 1$ there is an algorithm for n by n matrix multiplication that runs in $n^{\omega+\varepsilon}$. The most recent improvements on these bounds have been achieved by bounding the rank of the Matrix Multiplication Tensor [41, 28].

The I/O literature does not seem to have kept pace with these improvements. While previous work discusses the efficiency of naive matrix multiplication and Strassen matrix multiplication, it does not discuss the further improvements that have been generated.

We note in this section that the modern techniques to improve matrix multiplication running time, those of bounding the rank of the Matrix Multiplication Tensor, all imply cache-efficient algorithms.

▶ **Theorem 9** (Matrix Multiplication I/O Complexity[23]). *Let $T_{MA}(n) = O\left(\frac{n^2}{B}\right)$ be the time it takes to do matrix addition on matrices of size n by n. If the matrix multiplication tensor's rank is bounded such that the RAM model running time is $n^{\omega'+\varepsilon}$ for any $0 < \varepsilon < 1$ then the following self-reduction exists for some constant α,*

$$T_{MM}(n) = O\left(\alpha^{\omega'+\varepsilon} T_{MM}\left(\frac{n}{\alpha}\right) + \alpha^{\omega'} T_{MA}\left(\frac{n}{\alpha}\right)\right).$$

Self-reductions feed conveniently into cache oblivious algorithms. Notably, when we plug this equation into the I/O Master Theorem from Section 2.2, we obtain the following bound as given in Lemma 10. Recursive structures like this tend to result in cache-oblivious algorithms. After all, regardless of the size of cache, the problems will be broken down until they fit in cache. Then, when a problem and the algorithm's execution fit in memory, the time to answer the query is $O(M/B)$, regardless of the size of M and B.

▶ **Lemma 10.** *If the matrix multiplication tensor's rank is bounded such that the RAM model running time is $n^{\omega'+\varepsilon}$ for any $0 < \varepsilon < 1$ and Theorem 9 holds with base case $T_{MM}(\sqrt{M}) = \frac{M}{B}$, then the running time for cache-oblivious matrix multiplication in the I/O model is at most $O\left(\frac{n^{\omega'+\varepsilon}}{M^{\frac{\omega'+\varepsilon}{2}-1}B} + \frac{n^2}{B}\right)$ for any $0 < \varepsilon < 1$.*

See full version of the paper for proof.

3 Novel Reductions

In this section we cover reductions related to Wiener Index, Median, Single Source Shortest Paths, and s-t Shortest Paths. We first cover our super linear lower bounds, then cover the linear lower bounds.

3.1 Super Linear Lower Bounds

We present reductions in the I/O model which yield new lower bounds. We have as corollaries of these same reductions related lower bounds in the RAM model. Many of these reductions relate to the problem of finding the Wiener Index of the graph.

We show diameter reduces to Wiener Index, APSP reduces to Wiener Index, and we show 3 vs 4 radius reduces to median.

▶ **Definition 11** (Wiener Index). Given a graph $G = (V, E)$ let $D[i, j]$ be the shortest path distance between node i and node j in the graph G, where $D[i, i] = 0$), $n = |V|$, and $m = |E|$. The Wiener index of the graph is $\sum_{\forall i \in V} \sum_{\forall j \in V} D[i, j]$.

▶ **Lemma 12.** *If (directed/undirected) Wiener index in a (weighted/unweighted) graph, $G = (V, E)$, is solvable in $T(n, m, M, B)$ time then for any choice of sets $X \subset V$ and $T \subset V$ the sum $\sum_{x \in X} \sum_{t \in T} \delta(x, t)$ is computable in $O\left(T(n, m, M, B) + \frac{m}{B}\right)$ time.*

See full version of the paper for proof.

▶ **Lemma 13.** *If undirected Wiener index in an unweighted graph, $G = (V, E)$, is solvable in $T(n, m, M, B)$ time then for any choice of sets $X \subset V$ and $T \subset V$ the sum $\sum_{x \in X} \sum_{t \in T} \max\{\min\{\delta(u, v), k + 1\}, k\}$ is computable in $O\left(T(kn, km, M, B) + \frac{k|E|}{B}\right)$ time.*

See full version of the paper for proof.

▶ **Theorem 14.** *If undirected Wiener index in an unweighted graph is solvable in $T(n, m, M, B)$ time then counting the number of pairs $x \in X$ and $t \in T$ in a directed/undirected, unweighted graph where $\delta(x, t) = k$ and computing the number of pairs where $\delta(x, t) \geq k$ is computable in $O\left(T(kn, km, M, B) + \frac{km}{B}\right)$ time.*

See full version of the paper for proof.

We now show that Wiener index, in sparse graphs, can efficiently return small diameters. Notably, this means that improvements to the sparse Wiener index algorithm will imply faster algorithms for the sparse diameter problem than exist right now.

▶ **Corollary 15.** *If Wiener index is solvable in $T(n, m, M, B)$ time then returning $\min\{$diameter$, k\}$ is solvable in $O\left(\log(k) T(kn, km, kM, kB) + \frac{km}{B}\right)$ time.*

See full version of the paper for proof.

▶ **Corollary 16.** *If Wiener index is solvable in $T(n, m, M, B)$ time then returning the number of nodes at distances in $[1, k]$ from each other can be done in $O\left(kT(kn, kM, kB) + \frac{k^2 m}{B}\right)$ time.*

See full version of the paper for proof.

Next, we prove that improvements to median finding in sparse graphs improve the radius algorithm, using a novel reduction. Notably, in the I/O model 3 vs 4 radius is slower than 2 vs 3 radius; whereas, in the RAM-model, these two problems both run in n^2 time. The gap in the I/O model of a factor of M is what allows us to make these statements meaningful.

▶ **Theorem 17.** *If median is solvable in $T(n, m, M, B)$ time then 3 vs 4 radius is solvable in $O\left(T(n, m, M, B) + \frac{n^2}{MB} + \frac{E \log(E)}{B}\right)$ time.*

See full version of the paper for proof.

It has previously been shown that Wiener Index is equivalent to APSP in the RAM model. Here we show this also holds in the I/O-model.

▶ **Theorem 18.** *If Wiener Index is solvable in $n^{3-\varepsilon}$ time in a dense graph then APSP is solvable in $\tilde{O}(n^{3-\varepsilon} + n^2/B)$ time.*

See full version of the paper for proof.

3.2 Linear-Time Reductions

In fine-grained complexity, it often does not make sense to reduce linear-time problems to one another because problems often have a trivial lower bound of $\Omega(n)$ needed to read in the entire problem. However, in the I/O model, truly linear time—the time needed to read in the input—is $\Theta(n/B)$. Despite significant effort, many problems do not achieve this full factor of B in savings, and thus linear lower bounds of $\Omega(n)$ are actually interesting. We can use techniques from fine-grained complexity to try to understand some of this difficulty.

In the remainder of this section, we cover reductions between linear-time graph problems whose best known algorithms take longer than $O(|E|/B)$ time. This covers many of even the most basic problems, like the s-t shortest path problem, which asks for the distance between two specified

nodes s and t in a graph G. The *sparse* s-t shortest paths problem has resisted improvement beyond $O(|V|/\sqrt{B})$ even in undirected unweighted graphs [30], and in directed graphs, the best known algorithms still run in time $\Omega(|V|)$ [17, 8].

Notably, the undirected unweighted s-t shortest path problem is solved by Single Source Shortest Paths (SSSP) and Breadth First Search (BFS). Further note that for directed graphs the best known algorithms for SSSP, BFS, and Depth First Search (DFS) in sparse, when $|E| = O(|V|)$, directed graphs take $O(|V|)$ time. Which is a cache miss for every constant number of operations, giving no speed up at all. SSSP, BFS, and DFS solve many other basic problems like graph connectivity.

By noting these reductions we want to show that improvements in one problem propagate to others. We also seek to explain why improvements are so difficult on these problems. Because, improving one of these problems would improve many others, any problem which requires new techniques to improve implies the others must also need these new techniques. Furthermore, any lower bound proved for one problem will imply lower bounds for the other problems reduced to it. We hope that improvements will be made to algorithms or lower bounds and propagated accordingly.

We show reductions between the following three problems in weighted and unweighted as well as directed and undirected graphs.

▶ **Definition 19** (s-t-shortest-path(G, s, t))**.** Given a graph G and pointers to two verticies s and t, return the length of the shortest path between s and t.

▶ **Definition 20** (Girth-Containing-Edge(G, e))**.** Given a graph G and a pointer to an edge e, return the length of the shortest cycle in G which contains e.

▶ **Definition 21** (Girth-Containing-Vertex(G, v))**.** Given a graph G and a pointer to a vertex v, return the length of the shortest cycle in G which contains v.

We now begin showing that efficient reductions exist between these hard to solve linear problems.

▶ **Theorem 22.** *Given an algorithm that solves (undirected/directed) s-t-shortest-path in $f(n, |E|, M, B)$ time (undirected/directed) Girth-Containing-Edge can be solved in $O(f(n, |E|, M, B) + O(1))$ time.*

See full version of the paper for proof.

▶ **Theorem 23.** *Given an algorithm that solves (undirected/directed) Girth-Containing-Edge in $f(n, |E|, M, B)$ time (undirected/directed) s-t-shortest-path can be solved in $O(f(n, |E|, M, B) + O(1))$ time.*

See full version of the paper for proof.

▶ **Theorem 24.** *Given an algorithm that solves (undirected/directed) Girth-Containing-Vertex in $f(n, |E|, M, B)$ time (undirected/directed) s-t-shortest-path can be solved in $O(f(n, |E|, M, B) + O(1))$ time.*

See full version of the paper for proof.

▶ **Theorem 25.** *Given an algorithm that solves (undirected/directed) Girth-Containing-Vertex in $f(n, |E|, M, B)$ time (undirected/directed) Girth-Containing-Edge can be solved in $O(f(n, |E|, M, B) + O(1))$ time.*

See full version of the paper for proof.

▶ **Theorem 26.** *Given an algorithm that solves directed Girth-Containing-Edge in $f(n, |E|, M, B)$ time then directed Girth-Containing-Vertex is solvable in $O(f(n, |E|, M, B) + n/B)$ time.*

See full version of the paper for proof.

▶ **Theorem 27.** *Given an algorithm that solves directed s-t-shortest-path in $f(n, |E|, M, B)$ time then directed Girth-Containing-Vertex is solvable in $O(f(n, |E|, M, B) + n/B)$ time.*

See full version of the paper for proof.

When solving Girth-Containing-Vertex in the directed case, we know which direction the path must follow the edges and can perform this decomposition. Unfortunately this no longer works in the undirected case and a more complex algorithm is needed, giving slightly weaker results.

▶ **Theorem 28.** *Given an algorithm that solves undirected s-t-shortest-path in $f(n, |E|, M, B)$ time then undirected Girth-Containing-Vertex is solvable in $O((f(n, |E|, M, B) + n/B)\lg(n))$ time.*

See full version of the paper for proof.

▶ **Theorem 29.** *Given an algorithm that solves undirected Girth-Containing-Edge in $f(n, |E|, M, B)$ time then undirected Girth-Containing-Vertex is solvable in $O((f(n, |E|, M, B) + n/B)\lg(n))$ time.*

See full version of the paper for proof.

4 Lower Bounds from Fine-Grained Reductions

The fundamental problems in the fine-grained complexity world are good starting points for assumptions in the I/O model because these problems are so well understood in the RAM model. Additionally, both APSP and 3-SUM have been studied in the I/O model [9, 34, 35]. These reductions allow us to propagate believed lower bounds from one problem to others, as well as propagate any potential future algorithmic improvements.

4.1 Reductions to 3-SUM

We will show that 3-SUM is reducible to both convolution 3-SUM and 0 triangle in the I/O-model.

▶ **Lemma 30.** *If convolution 3-SUM is solved in $f(n, M, B)$ time then 3-SUM is solved in $O(g^3 f(n/g, M, B)) + n^2/(gMB))$ time for all $g \in [1, n]$.*

See full version of the paper for proof.

▶ **Corollary 31.** *If convolution 3-SUM is solved in time $O(n^{2-\varepsilon}/(MB))$ or $O(n^2/(M^{1+\varepsilon}B))$ or $O(n^2/(MB^{1+\varepsilon}))$ then 3-SUM is solved in $O(n^{2-\varepsilon'}/(MB))$ or $O(n^2/(M^{1+\varepsilon'}B))$ or $O(n^2/(MB^{1+\varepsilon'}))$ time, violating the I/O 3-SUM conjecture.*

See full version of the paper for proof.

▶ **Lemma 32.** *If 0 triangle is solved in $f(n, M, B)$ time then convolution 3-SUM is solved in $O(\sqrt{n} f(\sqrt{n}, M, B) + n^{1.5}\lg_{M/B}(n)/B)$ time.*

See full version of the paper for proof.

▶ **Theorem 33.** *If 0 triangle is solved in time $O(n^{3-\varepsilon}/(MB))$ or $O(n^3/(M^{1+\varepsilon}B))$ or $O(n^3/(MB^{1+\varepsilon}))$ then 3-SUM is solved in $O(n^{2-\varepsilon'}/(MB))$ or $O(n^2/(M^{1+\varepsilon'}B))$ or $O(n^2/(MB^{1+\varepsilon'}))$ time, violating the I/O 3-SUM conjecture.*

See full version of the paper for proof.

▶ **Lemma 34.** *If 3-SUM is solved in $f(n, M, B)$ I/Os then Convolution 3-SUM is solvable in $O(f(n, M, B) + n/B)$ I/Os.*

See full version of the paper for proof.

▶ **Corollary 35.** *Convolution 3-SUM is solvable in $O(n^2/(MB) + n/B)$*

Proof. Given Lemma 34 ◀

4.2 APSP Reductions in the IO-Model

▶ **Definition 36** (All-Pairs-Shortest-Path(G))**.** Given a fully connected graph G with large edge weights (weights between $-n^c$ and n^c for some constant c) return the path lengeths between all pairs of nodes in a matrix D where $D[i][j] =$ the length of the shortest path from node i to node j).

Another related version of APSP requires us to return all the shortest paths in addition to the distances. To represent this information efficiently, one is required to return an n by n matrix P where the $P[i][j]$ is the next node after i on the shortest path from i to j. The matrix P allows one to extract the shortest path between two points by following the path through the matrix P. This problem is also called APSP.

▶ **Definition 37** (Three-Layer-APSP(G)). Solve APSP on G where G is promised to be a bipartite graph G which has partitions A, B, and C, such that there are no edges within A,B or C and no edges between A and C.

▶ **Definition 38** (Negative-triangle-detection(G)). Given a graph G, retuning true if there is a negative triangle and false if there is no negative triangle. This problem is also called $-\triangle$ detection.

▶ **Definition 39** ($(\min,+)$-Matrix-Multiplication(A,B)). This problem is a variant on matrix multiplication.Given an n by n matrix A and an n by n matrix B return an n by n matrix C such that $C[i][j] = \min(\{A[i][k] + B[k][j] | \forall k \in [1,n]\})$.

The motivation for showing I/O equivalences between these problems is two fold. First, just as in the RAM model, these reductions can provide a shared explanation for why some problems have seen no improvement in their I/O complexity for years.

The set of reductions.

▶ **Theorem 40.** *If $(\min,+)$ runs in time $f(n,M,B)$, then APSP runs in $O(\lg(n) f(n,M,B))$.*

See full version of the paper for proof.

▶ **Theorem 41.** *If All Pairs Shortest Paths runs in time $f(n,M,B)$, then negative weight triangle detection in a tripartite graph runs in $O(f(n,M,B))$ cache misses.*

See full version of the paper for proof.

▶ **Theorem 42.** *If negative weight triangle detection in a tripartite graph runs in time $f(n,M,B)$, then three layer APSP with weights in the range $[-W,W]$ runs in $O\left(\lg(W) n^2 f\left(n^{1/3},M,B\right)\right)$ cache misses.*

version of the paper for proof.

▶ **Corollary 43.** *If negative weight triangle detection in a tripartite graph runs in time $f(n,M,B)$, then three layer APSP over weights in the range $[-poly(n), poly(n)]$ runs in $O\left(\lg(n) n^2 f\left(n^{1/3},M,B\right)\right)$ cache misses.*

See full version of the paper for proof.

▶ **Lemma 44.** *If All Pairs Shortest Paths runs in time $f(n,M,B)$, then three layer APSP runs in $O(f(n,M,B))$ cache misses.*

See full version of the paper for proof.

The equivalences.

▶ **Theorem 45.** *The following problems run in time $\widetilde{O}(\frac{N^3}{\sqrt{M}B} + \frac{n^2}{B})$ cache misses:*

1. *All Pairs Shortest Paths*
2. *$(\min,+)$ matrix multiplication*
3. *Negative triangle detection in a tripartite graph*
4. *Three layer APSP*

See full version of the paper for proof.

▶ **Corollary 46.** *The following solve APSP faster.*

1. *If $(\min,+)$ matrix multiplication is solvable in $f(n)$ time then APSP is solvable in $O(lg(n)f(n,M,B))$ time.*
2. *If negative triangle detection in a tripartite graph is solvable in $f(n)$ time then APSP is solvable in $O(lg^(n)n^2 f(n^{1/3},M,B))$ time.*

See full version of the paper for proof.

▶ **Lemma 47.** *If 0 triangle is solved in $f(n,M,B)$ time then $-\triangle$ over numbers in the range of $[-W,W]$ is solved in $O(lg(W)f(n,M,B) + lg(W)n^2/B)$.*

See full version of the paper for proof.

▶ **Lemma 48.** *Zero triangle is solvable in $O(n^3/(\sqrt{M}B) + n^2/B)$ I/Os.*

See full version of the paper for proof.

4.3 Orthogonal Vectors (OV)

▶ **Lemma 49.** *OV is solvable in $O(n^2/(MB) + n/B)$ I/Os cache obliviously.*
See full version of the paper for proof.

▶ **Lemma 50.** *If sparse diameter is solvable in $f(n, M, B)$ I/Os then OV is solvable in $\tilde{O}(f(n, M, B) + n/B)$ I/Os.*
See full version of the paper for proof.

▶ **Lemma 51.** *If Edit Distance is solvable in $f(n, M, B)$ time then OV is solvable in $\tilde{O}(f(n, M, B) + n/B)$ I/Os.*
See full version of the paper for proof.

▶ **Lemma 52.** *If Longest Common Subsequence is solvable in $f(n, M, B)$ time then OV is solvable in $\tilde{O}(f(n, M, B) + n/B)$ I/Os.*
See full version of the paper for proof.

4.4 Hitting Set

Abboud, Vassilevska-Williams and Wang define a new problem, called Hitting Set [4].

▶ **Definition 53** (Hitting Set)**.** Given an input of a list A and B of d dimensional 1 and 0 vectors return $True$ if there $\exists a \in A$ such that $\forall b \in B\ a \cdot b > 0$.

▶ **Lemma 54.** *Hitting Set is solvable in $\tilde{O}(n^2/(MB) + n/B)$ I/Os.*

See full version of the paper for proof.

▶ **Lemma 55.** *If sparse radius is solvable in $f(n, M, B)$ I/Os then Hitting Set is solvable in $\tilde{O}(f(n, M, B) + n/B)$ I/Os.*

See full version of the paper for proof.

5 I/O Model Complexity Classes

In this section we examine the I/O model from a complexity theoretic perspective. Section 5.1 provides some necessary background information. In Section 5.2 we define the classes $\mathit{PCACHE}_{M,B}$ and $\mathit{CACHE}_{M,B}\left(t\left(n\right)\right)$ describing the problems solvable in a polynomial number of cache misses and $O\left(t\left(n\right)\right)$ cache misses respectively. We then demonstrate that $\mathit{PCACHE}_{M,B}$ lies between P and $PSPACE$ for reasonable choices of cache size. In the full versionwe provide simulations between the I/O model and both the RAM and Turing machine models. In Section 5.3 we prove the existence of a time hierarchy in $\mathit{CACHE}_{M,B}\left(t\left(n\right)\right)$. The existence of a time hierarchy in the I/O model grounds the study of fine-grained complexity by showing that such increases in running time do provably allow more problems to be solved. The techniques to achieve many of the results in this section also follow in the same theme of reductions, although the focus of the problems examined is quite different.

5.1 Hierarchy Preliminaries: Oracle Model

Oracles are used to prove several results, most notably in the time-hierarchy proof. The oracle model was introduced by Turing in 1938 [40]. The definition we use here comes from Soare [39] and similar definitions can be found in computational complexity textbooks.

In the Turing machine oracle model of computation we add a second tape and corresponding tape head. This oracle tape and its oracle tape head can do everything the original tape can: reading, writing and moving left and right. This oracle tape head has two additional states ASK and $RESPONSE$. After writing to the oracle tape, the tape head can go into the ASK state. In the ASK state the oracle computation is done on the input written to the oracle tape and then the tape head is changed to the $RESPONSE$ state. All of this is done in one computational step. If the oracle is the function language $L: \{0,1\}^n \to \{0,1\}^*$, then the output is written on the tape for the input i is $L(i)$. This allows the Turing machine to make $O(1)$ cost black box calls to the oracle language and get strings as output.

The notation A^B describes a computational class of the languages decidable by an oracle Turing machine of A with a B oracle. The oracle language will be a language decidable in the function version of B. The oracle machine will then be resource limited as A is resource limited.

In this paper we will also talk about RAM machine oracles. This is a simple extension of the typical Turing machine oracle setup. The RAM machine will have two randomly accessible memories. One will be the standard RAM memory. The other memory will be the oracle memory, the RAM can read and write words to this memory and can additionally enter the ASK state. One time step after entering the ASK state the RAM will be returned to the $RESPONSE$ state and the contents of the oracle memory will contain the oracle language output, $L(i)$.

5.2 *PCACHE*$_{M,B}$ and its relationship with P and PSPACE

First we define the class of problems solvable given some function, $t(n)$, the number of cache misses, up to constant factors.

▶ **Definition 56.** Let *CACHE*$_{M,B}(t(n))$ be the set of problems solvable in $O(t(n))$ cache misses on a IO-model machine with a cache of size $O(M)$ and a cache line size of size $O(B)$.

Next, we consider the class of problems solvable by any polynomial number of cache misses.

▶ **Definition 57.** Let *PCACHE*$_{M,B}$ be the set of problems solvable in polynomial numbers of cache misses on a IO-model machine with a cache of size $O(M)$ and a cache lines of size $O(B)$.

$$\textit{PCACHE}_{M,B} = \bigcup_{i=1}^{\infty} CACHE_{M,B}(n^i)$$

First, let us note that the CACHE class can simulate the RAM class. The IO-model is basically a RAM model with extra power.

▶ **Lemma 58.** $RAM(t(n)) \subseteq CACHE_{M,B}(t(n))$

See full version of the paper for proof.

Now we introduce a complexity class *MEM*. Note that this class is very similar to $SPACE$.

▶ **Definition 59.** We define the class *MEM*$(s(n))$ to be the set of problems solvable in $SPACE(s(n))$ when the input is of size $O(s(n))$.

Why *MEM* and not $SPACE$? We want to use the *MEM* class as an oracle which will model computation doable on a cache machine in one cache miss. When $t(n) = \Omega(n)$ then *MEM*$(t(n)) = SPACE(t(n))$; however, these classes differ when we have a small work space. A $SPACE(o(n))$ machine is given a read-only tape of size n and compute space $o(n)$. This extra read-only tape gives the $SPACE$ machine too much power when compared with the cache. Notably, we can scan through the entire input with one step with a $SPACE(\lg(n))$ oracle. A cache would require n/B time to scan this input.

▶ **Lemma 60.** *MEM*$(Mw) \subseteq CACHE_{M,B}(M/B)$.

See full version of the paper for proof.

Now we prove that a RAM machine with oracle access to our *MEM* oracle can be simulated by a cache machine. We simulate the MEM machine and RAM machine together efficiently in cache.

▶ **Corollary 61.** $RAM(t(n))^{MEM(Mw)} \subseteq CACHE_{M+3,B}(t(n))$.

See full version of the paper for proof.

The class *PCACHE*$_{M,B}$ is equivalent to a polynomial time algorithm with oracle access to a MEM oracle. Intuitively, in both cases we get to use a similarly powerful object (the cache or the MEM oracle) a polynomial number of times.

▶ **Theorem 62.** *If* $Bw = O(poly(n))$, *then* *PCACHE*$_{M,B} = P^{MEM(Mw)}$

See full version of the paper for proof.
We then note that in many cases MEM and SPACE are equivalent.

▶ **Corollary 63.** *If* $Bw = O(poly(n))$ *and* $Mw = \Omega(n)$*, then* $PCACHE_{M,B} = P^{SPACE(Mw)}$.

See full version of the paper for proof.
Finally, we note that P is a subset of $PCACHE_{M,B}$.

▶ **Lemma 64.** *If* $Mw = \Theta(poly(n))$*, then* $PCACHE_{M,B} \subseteq PSPACE$

See full version of the paper for proof.

▶ **Lemma 65.** $\bigcup_{c=1}^{\infty} PCACHE_{n^c,B} = PSPACE$

See full version of the paper for proof.

5.3 $CACHE_{M,B}$ Hierarchy

In this section we prove that a hierarchy exists in the IO-model. The separation in the CACHE hierarchy is B times the separation for the RAM hierarchy. We know that RAM machines given polynomially more time can solve more problems than those given polynomially less.

▶ **Theorem 66.** *For* $\varepsilon \geq 0$*,*

$$RAM^O(t(n)) \subsetneq RAM^O((t(n))^{1+\varepsilon}).\ [22]$$

Let $s(n)$ be the space usage of the algorithm running on the RAM machine. Let $\alpha = \frac{B+\lceil \lg(s(n))/w \rceil}{B}$, which is the number of cache lines needed to represent both a cache line and its memory address. Note, in the case where one word is large enough to address all of the memory used by the algorithm (a standard assumption) $\alpha = 1 + 1/B \leq 2$. We now give a simulation of a CACHE machine by a RAM machine with MEM oracle.

▶ **Lemma 67.** $CACHE_{M,B}(t(n)) \subseteq RAM(t(n)(B+\alpha))^{MEM(Mw\alpha)}$

See full version of the paper for proof.
Plugging our simulation into the RAM hierarchy gives a separation result for the CACHE complexity classes.

▶ **Theorem 68.** *For all* $\varepsilon > 0$

$$CACHE_{M,B}(t(n)) \subsetneq CACHE_{M\alpha,B}((\alpha t(n) B)^{1+\varepsilon}).$$

See full version of the paper for proof.
Under reasonable assumptions about the values of input and word sizes, we can construct a cleaner version of the above theorem.

▶ **Corollary 69.** *When* $s(n) = 2^{O(wB)}$*, in other words the memory used by the algorithm is referenceable by* $O(B)$ *words,*

$$CACHE_{M,B}(t(n)) \subsetneq CACHE_{M,B}((t(n) B)^{1+\varepsilon}).$$

Proof. Note $\alpha = 1 + 1/B = O(1)$, and thus is a constant with respect to the time and size of memory which are defined asymptotically. Thus this factor disappears. ◀

Acknowledgements

We thank the anonymous reviewers for their helpful suggestions.

References

1. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is Valiant's parser. In *Proceedings of the IEEE 56th Annual Symposium on Foundations of Computer Science*, FOCS 2015, pages 98–117, Berkeley, CA, October 2015.
2. Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2015, pages 1681–1697, San Diego, CA, January 2015.
3. Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 434–443, 2014.
4. Amir Abboud, Virginia Vassilevska Williams, and Joshua Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 377–391, 2016.
5. Alok Aggarwal and S. Vitter, Jeffrey. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, September 1988.
6. Lars Arge. External-memory algorithms with applications in gis. In *Algorithmic Foundations of Geographic Information Systems, This Book Originated from the CISM Advanced School on the Algorithmic Foundations of Geographic Information Systems*, pages 213–254, 1997. URL: http://dl.acm.org/citation.cfm?id=648260.753061.
7. Lars Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003. URL: http://dx.doi.org/10.1007/s00453-003-1021-x, doi:10.1007/s00453-003-1021-x.
8. Lars Arge, Michael A. Bender, Erik D. Demaine, Bryan Holland-Minkley, and J. Ian Munro. An optimal cache-oblivious priority queue and its application to graph algorithms. *SIAM Journal on Computing*, 36(6):1672–1695, 2007. URL: https://doi.org/10.1137/S0097539703428324, arXiv:https://doi.org/10.1137/S0097539703428324, doi:10.1137/S0097539703428324.
9. Lars Arge, Ulrich Meyer, and Laura Toma. External memory algorithms for diameter and all-pairs shortest-paths on sparse graphs. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming*, ICALP 2004, pages 146–157, Turku, Finland, July 2004. URL: https://doi.org/10.1007/978-3-540-27836-8_15, doi:10.1007/978-3-540-27836-8_15.
10. Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In *Proceedings of the 47th Annual ACM on Symposium on Theory of Computing*, pages 51–58, 2015.
11. Boaz Barak. A probabilistic-time hierarchy theorem for "slightly non-uniform" algorithms. In *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques*, RANDOM 2002, pages 194–208, Cambridge, MA, September 2002.
12. Ilya Baran, Erik D Demaine, and Mihai Pătraşcu. Subquadratic algorithms for 3sum. In *Workshop on Algorithms and Data Structures*, pages 409–421, 2005.
13. R. Bayer and E. McCreight. Organization and maintenance of large ordered indices. In *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, SIGFIDET '70, pages 107–141, 1970. URL: http://doi.acm.org/10.1145/1734663.1734671, doi:10.1145/1734663.1734671.
14. Michael A. Bender, Richard Cole, Erik D. Demaine, Martin Farach-Colton, and Jack Zito. Two simplified algorithms for maintaining order in a list. In *Proceedings of the 10th Annual European Symposium on Algorithms*, ESA 2002, pages 152–164, 2002. URL: http://dl.acm.org/citation.cfm?id=647912.740822.
15. Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *Proceedings of the 55th IEEE Annual Symposium on Foundations of Computer Science*, FOCS 2014, pages 661–670, Philadelphia, PA, October 2014.
16. Gerth Stølting Brodal. Cache-oblivious algorithms and data structures. In Torben Hagerup and Jyrki Katajainen, editors, *Proceedings of the 9th Scandinavian Workshop on Algorithm Theory*, SWAT 2004, pages 3–13. Humlebæk, Denmark, July 2004. URL: https://doi.org/10.1007/978-3-540-27810-8_2, doi:10.1007/978-3-540-27810-8_2.

17 Yi-Jen Chiang, Michael T Goodrich, Edward F Grove, Roberto Tamassia, Darren Erik Vengroff, and Jeffrey Scott Vitter. External-memory graph algorithms. In *SODA*, volume 95, pages 139–149, 1995.

18 Rezaul Alam Chowdhury and Vijaya Ramachandran. Cache-oblivious shortest paths in graphs using buffer heap. In *Proceedings of the 16th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA 2004, pages 245–254, 2004. URL: http://doi.acm.org.libproxy.mit.edu/10.1145/1007912.1007949, doi:10.1145/1007912.1007949.

19 Rezaul Alam Chowdhury and Vijaya Ramachandran. External-memory exact and approximate all-pairs shortest-paths in undirected graphs. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 735–744, 2005.

20 Rezaul Alam Chowdhury and Vijaya Ramachandran. Cache-oblivious dynamic programming. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 591–600, 2006.

21 Rezaul Alam Chowdhury and Vijaya Ramachandran. Cache-oblivious dynamic programming. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 591–600, 2006.

22 Stephen A. Cook and Robert A. Reckhow. Time-bounded random access machines. In *Proceedings of the 4th Annual ACM Symposium on Theory of Computing*, pages 73–80, 1972.

23 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251 – 280, 1990. Computational algebraic complexity editorial. URL: http://www.sciencedirect.com/science/article/pii/S0747717108800132, doi:http://dx.doi.org/10.1016/S0747-7171(08)80013-2.

24 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

25 Erik D. Demaine. Cache-oblivious algorithms and data structures. Lecture Notes from the EEF Summer School on Massive Data Sets, 2002.

26 Roman Dementiev. *Algorithm engineering for large data sets.* PhD thesis, Saarland University, 2006.

27 Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *Proceedings of the 40th Annual Symposium on the Foundations of Computer Science (FOCS)*, pages 285–298, 1999.

28 François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC 2014, Kobe, Japan, July 23-25, 2014*, pages 296–303, 2014.

29 Jia-Wei Hong and H. T. Kung. I/O complexity: The red-blue pebble game. In *Proceedings of the 13th Annual ACM Symposium on the Theory of Computation (STOC)*, pages 326–333, 1981.

30 Kurt Mehlhorn and Ulrich Meyer. External-memory breadth-first search with sublinear i/o. In Rolf Möhring and Rajeev Raman, editors, *Proceedings of the 10th Annual European Symposium on Algorithms*, ESA 2002, pages 723–735. Rome, Italy, September 2002. URL: https://doi.org/10.1007/3-540-45749-6_63, doi:10.1007/3-540-45749-6_63.

31 Bruno Menegola. An external memory algorithm for listing triangles. Bachelor's thesis, 2010. URL: http://hdl.handle.net/10183/26335.

32 Ulrich Meyer and Norbert Zeh. I/o-efficient undirected shortest paths. In Giuseppe Di Battista and Uri Zwick, editors, *Proceedings of the 11th Annual European Symposium on Algorithms*, ESA 2003, pages 434–445. Budapest, Hungary, September 2003. URL: https://doi.org/10.1007/978-3-540-39658-1_40, doi:10.1007/978-3-540-39658-1_40.

33 Rasmus Pagh and Francesco Silvestri. The input/output complexity of triangle enumeration. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 224–233, 2014.

34 Rasmus Pagh and Morten Stöckel. The input/output complexity of sparse matrix multiplication. In Andreas S. Schulz and Dorothea Wagner, editors, *Proceedings of the 22th Annual European Symposium on Algorithms*, ESA 2014, pages 750–761. Wroclaw, Poland, September 2014. URL: https://doi.org/10.1007/978-3-662-44777-2_62, doi:10.1007/978-3-662-44777-2_62.

35 Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, pages 603–610, 2010.

36 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the 45th Symposium on Theory of Computing Conference*, STOC 2013, pages 515–524, Palo Alto, CA, June 2013.

37 Raimund Seidel. On the all-pairs-shortest-path problem. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, STOC 1992, pages 745–749, New York, NY, USA, 1992. URL: http://doi.acm.org/10.1145/129712.129784, doi:10.1145/129712.129784.

38 Michael Sipser. *Introduction to the Theory of Computation*, volume 2. 2006.

39 Robert I. Soare. *Recursively enumerable sets and degrees: A study of computable functions and computably generated sets*. 1999.

40 A. M. Turing. Systems of logic based on ordinals. *Proceedings of the London Mathematical Society*, s2-45(1):161–228, 1939.

41 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *STOC*, pages 887–898, 2012.

42 Jeffrey Scott Vitter. External memory algorithms and data structures: dealing with massive data. *ACM Computing Surveys*, 33(2), 2001.

43 Ryan Williams. Hierarchy for BPP vs derandomization. Theoretical Computer Science Stack Exchange. http://cstheory.stackexchange.com/q/6769. URL: http://cstheory.stackexchange.com/q/6769, arXiv:http://cstheory.stackexchange.com/q/6769.

44 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 645–654, 2010.