Spectral Lower Bounds on the I/O Complexity of Computation Graphs

Saachi Jain
saachi@cs.stanford.edu
Stanford University
Stanford, CA, USA

Matei Zaharia
matei@cs.stanford.edu
Stanford University
Stanford, CA, USA

ABSTRACT

We consider the problem of finding lower bounds on the I/O complexity of arbitrary computations in a two level memory hierarchy. Executions of complex computations can be formalized as an evaluation order over the underlying computation graph. However, prior methods for finding I/O lower bounds leverage the graph structures for specific problems (e.g matrix multiplication) which cannot be applied to arbitrary graphs. In this paper, we first present a novel method to bound the I/O of any computation graph using the first few eigenvalues of the graph's Laplacian. We further extend this bound to the parallel setting. This spectral bound is not only efficiently computable by power iteration, but can also be computed in closed form for graphs with known spectra. We apply our spectral method to compute closed-form analytical bounds on two computation graphs (the Bellman-Held-Karp algorithm for the traveling salesman problem and the Fast Fourier Transform), as well as provide a probabilistic bound for random Erdős Rényi graphs. We empirically validate our bound on four computation graphs, and find that our method provides tighter bounds than current empirical methods and behaves similarly to previously published I/O bounds.

CCS CONCEPTS

• **Mathematics of computing** → **Spectra of graphs**; • **Theory of computation** → *Lower bounds and information complexity*; • **Software and its engineering** → *Input / output*.

KEYWORDS

computational graphs, spectral graph theory, I/O lower bounds

ACM Reference Format:
Saachi Jain and Matei Zaharia. 2020. Spectral Lower Bounds on the I/O Complexity of Computation Graphs. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '20), July 15–17, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3350755.3400210

1 INTRODUCTION

Many important applications are bottlenecked not by processing speeds, but by I/O cost: the speed to transfer data items between fast memory (e.g., registers or the CPU cache) and slow memory (e.g., RAM or disk). There has thus been considerable interest in designing I/O efficient algorithms and in understanding I/O lower bounds [3, 8, 12, 14].

Past work on I/O lower bounds has largely focused on finding bounds for specific algorithms, such as matrix multiplication or the Fast Fourier Transform [3, 8, 12, 14]. However, these approaches leverage properties specific to the tasks at hand, and do not translate across tasks. In this paper, we explore methods that can be applied to *arbitrary* computations and can be computed efficiently in an automatic fashion. Such generic bounds can be used to characterize the I/O cost of computations that are too complex to analyze by hand. Our method also provides a new approach for finding closed form theoretical bounds on computation graphs as long as the Laplacian eigenvalues (or bounds on these values) are known.

We approach the problem of minimizing I/O for an arbitrary computation as finding an optimal evaluation order on the underlying directed computation graph. In a computation graph, each vertex represents a single operation: the parents of the vertex indicate the operands of the operation. We assume a two-level memory architecture with a fixed amount of fast memory and infinite slow memory: I/O is incurred when transferring data between fast and slow memory (Section 3).

We present a novel method to provide lower bounds on the I/O for any computation graph using the eigenvalues of the graph Laplacian (Section 4). We further extend this bound to the parallel setting. This spectral bound is efficiently computable and can be applied to arbitrarily large and complex graphs. For graphs with known spectra, this bound can also be computed in closed form. We compute closed form bounds for two computation graphs: the Bellman-Held-Karp algorithm for the traveling salesman problem (TSP) as well as the Fast Fourier Transform (FFT). In the process, we also present a novel result on the multiplicity of the eigenvalues of the butterfly graph, which we use to complete the bound for the FFT. We find that spectral bound for the FFT graph is at most a factor of $(1/\log M)$ weaker than the previously published asymptotically tight bound (where M is the size of fast memory), which was computed via direct inspection of the butterfly graph using S partitions [14]. We additionally present a probabilistic bound for random Erdős Rényi graphs.

We evaluate our method empirically by computing lower bounds for four types of computation graphs: the Fast Fourier Transform, matrix multiplication (naive and Strassen), and the Bellman-Held-Karp algorithm (Section 6). We find that our bounds are tighter than current automatic methods [9] and behave similarly to published analytical bounds.

2 RELATED WORK

Hong and Kung first framed the problem of I/O complexity as the "red-blue pebble game" and used it to prove several bounds [14]. The game represents slow memory as an infinite pool of blue pebbles and fast memory as a finite set of red pebbles. An evaluation then corresponds to pebbling each vertex of the graph according to the game; I/O is incurred when placing a red pebble on top of a blue pebble (reading from slow memory) or vice-versa (writing to slow memory).

Lower bounds on naïve matrix multiplication often use the Loomis-Whitney theorem, which embeds operations in the voxels of a computation cube [2, 12]. However, volume based arguments such as Loomis-Whitney do not apply for more general computations. I/O bounding techniques for algorithms beyond matrix multiplication generally focus on the computation graph itself.

Most current work on lower bounds via computation graphs requires manual inspection of the graph. In [14], the authors find a $2S$ partition of the computation graph to bound I/O—a proof technique that is non-trivial for complex graphs. In [16] and [5], the authors use path routing and dichotomy width respectively to find lower bounds. In [3], the authors reduce the I/O problem to a graph partitioning problem in order to find a lower bound for Strassen's matrix multiplication algorithm using the edge expansion of the graph, which was computed by hand by recursively decomposing the Strassen computation graph. None of these methods can easily be computed automatically for arbitrary graphs. Instead, lower bounds on each graph must be separately proved by inspecting the specific graph, and are thus difficult to generalize. We instead focus on methods that can *automatically* compute lower bounds for any input graph, regardless of its structure.

To our knowledge, there are only two works that discuss automated methods for lower bounds for arbitrary graphs. In the first work, the authors find automatic bounds by computing convex min s-t cuts on the sub-graphs [9]. With a runtime $O(n^5)$ for a graph with n nodes, this method is significantly slower than our spectral method, which can be computed in $O(n^3)$. We compare against this method in Section 6 and find that it yields looser bounds than our proposed spectral method. The second work uses an Integer Linear Program (ILP) to solve for the $2S$ partition of the computation graph [8]. This method is computationally expensive because it necessitates an exact ILP solver and is thus combinatorial in difficulty. Since this ILP based method is intractable, we do not compare its performance against the spectral bound as the method cannot be performed for large graphs, instead limiting ourselves to methods that can be computed in polynomial time.

3 COMPUTATION GRAPHS AND MEMORY MODEL

A computation can be represented by an underlying directed computation graph G. Each operation, including the inputs and outputs, is represented by a vertex. An edge from u to v indicates that the operation v was computed with u as an operand. The graph is acyclic, with the inputs as sources and the outputs as sinks. For example, the inner product of two vectors with two elements each can be represented as a 7 vertex graph: 4 vertices for inputs, 2 vertices for the intermediate products, and a single vertex for the sum. (Figure 1).

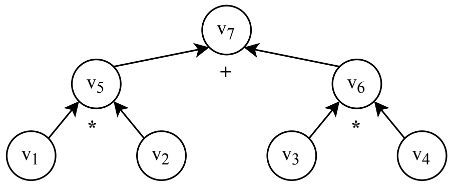


Figure 1: Computation graph of an inner product.

We assume a two level memory hierarchy on a single processor with infinite slow memory and a limited cache of fast memory of size M elements, where the result of each operation in the computation graph is a single element. Every operation in the computation graph must be evaluated. When a vertex v is evaluated, the parents of v must be loaded into fast memory from slow memory if they are not already present. As in [3, 8, 16], we disallow recomputation of the same vertex: therefore, if a computed result is needed elsewhere in the computation graph and is about to be evicted, the result must first be written to slow memory.

I/O can be separated into trivial (reading inputs and writing outputs) and non-trivial I/O. We focus on *non-trivial* I/O: we thus do not directly include the cost of reading inputs or writing outputs. Instead, we assume that inputs can be read from the user directly into fast memory, and outputs are reported to the user immediately as they are computed. However: if an input is evicted from fast memory and is still needed elsewhere in the computation, it must be written to slow memory. This assumption is inherent in the proof in [3, 9]. Because we seek lower bounds, we do not constrain the eviction policy of fast memory. I/O is incurred when, during computation, an element is written to slow memory from fast memory or read from slow memory into fast memory.

An evaluation order is then the order that operations are evaluated in the graph. Since a vertex can only be evaluated after its parents, a valid evaluation order must be topological with respect to the graph. We thus seek lower bounds on the I/O incurred by the optimal evaluation order.

3.1 Optimization Task

Formally, let $G = (V, E)$ be a computation graph with vertices V and edges E. Let $n = |V|$ be the number of operations in the graph, and let M be the size of fast memory. Note that each vertex in the graph is evaluated exactly once; therefore, the total computation takes exactly n time-steps.

We formalize an evaluation order on G as a permutation matrix $X \in \mathbb{R}^{n \times n}$, where X_{ij} is one if v_j is computed at time-step i. Let O_G be the set of valid topological orders on G. Since vertices must be evaluated after their operands, $X \in O_G$.

An I/O is incurred every time an element must be read into fast memory from slow memory or written to slow memory from fast memory. Let $J_G(X)$ be the number of nontrivial I/Os that were incurred by evaluating G in the order specified by X on G. We seek a lower bound on J_G^*, the optimal I/O incurred by any evaluation order:

$$J_G^* = \inf_{X \in O_G} J_G(X).$$

4 SPECTRAL BOUNDS VIA THE GRAPH LAPLACIAN

In this section, we find a lower bound based on the eigenvalues of the graph Laplacian. We first link the problem to the edge expansion of the graph, by counting the number of edges that cross boundaries over a graph partition as in [3]. We frame this problem as a quadratic program (QP) with respect to the graph Laplacian. Finally we use the Laplacian's spectra to find a lower bound on the solution to the QP.

Notation: For $v \in V$, let $d_{in}(v), d_{out}(v)$, and $d(v)$ be the in-degree, out-degree, and total degree of v respectively. Finally, for any subset $S \subseteq V$, we define ∂S as the edge boundary of S: $\partial S = \{(u, v) \in E \mid (u \in S \land v \notin S) \lor (v \in S \land u \notin S)\}$.

4.1 Counting Edges over Graph Partitions

For any evaluation X on G, we can choose a partition $P \subseteq 2^V$ that divides V into disjoint subsets of vertices so each $S \in P$ is contiguously ordered by X. P thus defines breakpoints on X. Figure 2 depicts an example of a partition on a graph. The numbers on the vertices indicate the evaluation order determined by X. The graph is then partitioned into green, yellow, and blue segments. Each segment is contiguous with respect to the order.

Let $\mathcal{P}_X$ be the set of valid partitions on X according to the ordering constraint. We leverage the following key lemma from [3], which divides the I/O cost of a subset of a computation graph into reads (edges entering the subgraph), and writes (edges leaving the subgraph). For each subset $S \in P$, define the following sets:

$$R_S = \{v \in V \mid v \notin S, \exists (v, u) \in E \text{ s.t } u \in S\},$$
$$W_S = \{v \in V \mid v \in S, \exists (v, u) \in E \text{ s.t } u \notin S\}.$$

R_S is the vertices not in S with an edge into S, and W_S is the vertices in S with an edge outside of S. Ballard et. al in [3] then present the following lemma:

Lemma 1 (Equation 6 from [3]).

$$J_G(X) \geq \max_{P \in \mathcal{P}_X} \left(\sum_{S \in P} |R_S| + |W_S| \right) - 2M|P|.$$

Proof. We summarize the proof of their lemma here. To evaluate the nodes in S, the vertices in R_S must be read into fast memory (or were already in fast memory before beginning computation of S). Similarly, the vertices in W_S are freshly computed and needed elsewhere in the evaluation and thus must be written out or left in fast memory at the end of S. (Figure 3). Since the fast memory size is only M, at least $|R_S| + |W_S| - 2M$ I/O's are incurred by evaluating the nodes in S.

Summing over all $S \in P$ leads to a bound on the IO incurred by G. Any P is valid so long as P splits V into components contiguous in X. Specifically, if $\mathcal{P}_X$ is the set of valid partitions with respect to X:

$$J_G(X) \geq \max_{P \in \mathcal{P}_X} \left(\sum_{S \in P} |R_S| + |W_S| \right) - 2M|P|.$$

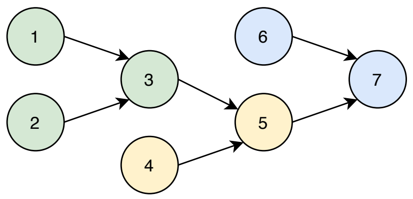


Figure 2: A computation graph: the numbers indicate the evaluation order and the colors are a valid partition.

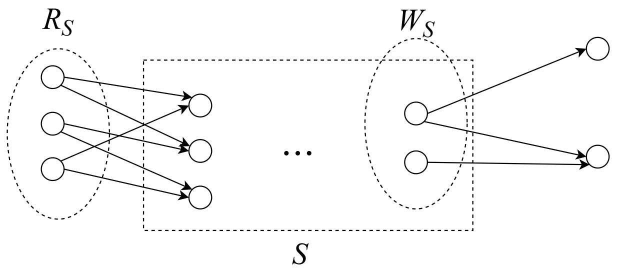


Figure 3: We identify sets R_S and W_S that cause I/O for each component S in partition P.

□

It is easier to compute the number of edges crossing into and out of S rather than the vertex sets R_S and W_S. Ballard et. al use this lemma to bound the I/O of the computation graph for Strassen matrix multiplication. However, they make several assumptions that weaken the bound for general graphs. Firstly, rather than computing a bound for all segments in the partition, they derive a bound for any single $n/|P|$ sized sub-graph within the Strassen computation graph. They then compute this bound specifically for Strassen-like graphs, and multiply this bound by $|P|$ to achieve a bound for the entire graph. This approach succeeds for the Strassen graph where the I/O is evenly distributed across the graph. However, this relaxation can be loose for graphs where the I/O is concentrated in a small portion of the vertices. Secondly, they deal strictly with regular graphs by adding loops to the computation graph. As a result, they link the size of $|R_S| + |W_S|$ to the size of the edge boundary by dividing by the maximum undirected degree, i.e $|R_S| + |W_S| \geq \frac{1}{d_{max}(u)}|\partial S|$. While this assumption is convenient for closed form bounds, it is not necessary for automatic methods where we can retain access to the graph.

The following theorem links the partition to the I/O cost. We diverge from [3] by bounding over all segments and maintaining access to the individual degrees of the vertices.

Theorem 2. *For fast memory size M and graph G, the optimal I/O is lower bounded by:*

$$J_G^* \geq \min_{X \in O_G} \max_{P \in \mathcal{P}_X} \left(\sum_{S \in P} \sum_{(u,v) \in \partial S} \frac{1}{d_{out}(u)} \right) - 2M|P|. \tag{1}$$

Proof. We bound $|R_S|$ and $|W_S|$ as:

$$|R_S| \geq \sum_{(u,v) \in E} \frac{\mathbb{I}\{u \notin S, v \in S\}}{d_{out}(u)}, \quad |W_S| \geq \sum_{(u,v) \in E} \frac{\mathbb{I}\{u \in S, v \notin S\}}{d_{out}(u)}.$$

Summing reads and writes, we have:

$$|R_S| + |W_S| \geq \sum_{(u,v)\in\partial S} \frac{1}{d_{out}(u)}.$$

Minimizing over all X, we get the full bound

$$J_G^* \geq \min_{X\in O_{\mathcal{G}}} \max_{P\in\mathcal{P}_X} \left(\sum_{S\in P} \sum_{(u,v)\in\partial S} \frac{1}{d_{out}(u)} \right) - 2M|P|. \qquad (2)$$

□

Intuitively, an adversary picks some evaluation order X on G. We pick a hard partition P on X to maximize the I/O incurred. In the next section, we formalize Theorem 2 as a quadratic program using the graph Laplacian of an out-degree normalized graph. We then lower bound the I/O cost via the eigenvalues of the Laplacian.

4.2 Formulation via the Graph Laplacian

In Theorem 2, we solved for the minimum order over a maximum partition. However, since any partition will give us a lower bound, we can choose to split our graph into evenly sized segments. We pick some $k \leq n$ as our number of segments: splitting into k subsets of as equally as possible (such that the first $n \mod k$ segments have $\lfloor n/k \rfloor + 1$ vertices and the rest have $\lfloor n/k \rfloor$ vertices). For an evaluation order X, let $P^{(X,k)} \in P_X$ be the k-partition described above. If $P^{(I,k)}$ would be the above partition assuming an identity evaluation order $X = I_k$, then we can define $\hat{W}^{(k)} \in \mathbb{R}^{n\times k}$ as $(\hat{W}^{(k)})_{ij} = \mathbb{I}\{i \in P_j^{(I,k)}\}$. Then $X\hat{W}^{(k)} \in \mathbb{R}^{n\times k}$ is the partition matrix for the k-partition $P^{(X,k)}$.

We transform our graph directed G into a weighted undirected graph as follows: for each directed edge $(u, v) \in G$, we add the undirected edge (u, v) to $\tilde{G}$ with weight $\frac{1}{d_{out}(u)}$. Henceforth, we indicate the degree function, degree matrix, and adjacency matrix of the original G as $d(v), D$, and A respectively; we analogously denote $\tilde{d}, \tilde{D}, \tilde{A}$ as the degree function, degree matrix, and adjacency matrix of $\tilde{G}$.

Let $\tilde{L} = \tilde{D} - \tilde{A}$ be the graph Laplacian of $\tilde{G}$. $\tilde{L}$ is positive semi-definite, so all of its eigenvalues are nonnegative. The Laplacian is convenient for expressing the edge boundaries of vertex subsets. Specifically, for subset $S \subseteq V$, let $x \in \mathbb{R}^n$ be the one-hot encoding of S (i.e $x_i = \mathbb{I}\{v_i \in S\}$). Then:

$$x^T \tilde{L} x = x^T \tilde{D} x - x^T \tilde{A} x = \sum_{(u,v)\in\partial S} \frac{1}{d_{out}(u)}. \qquad (3)$$

Using this property we can bound the edge crossing as:

$$\mathrm{tr}((\hat{W}^{(k)})^T X^T \tilde{L} X \hat{W}^{(k)}) = \sum_{S\in P^{(X,k)}} \sum_{(u,v)\in\partial S} \frac{1}{d_{out}(u)}.$$

Letting $W^{(k)} = \hat{W}^{(k)} \hat{W}^{(k)T}$, and rewriting Equation 1 leads to the following quadratic program:

Theorem 3 (I/O Bound via Graph Laplacian). *For a computation graph G and any $k \leq n$ with $\tilde{L}$ and $W^{(k)}$ defined as above, J_G^* is lower bounded by the solution of:*

$$\begin{aligned} \textit{minimize}_X \quad & \max_k \mathrm{tr}(X^T \tilde{L} X W^{(k)}) - 2kM \\ & X \in O_G. \end{aligned}$$

In the next section, we relax the above optimization problem to find a lower bound on the objective using the eigenvalues of $\tilde{L}$ and $W^{(k)}$.

4.3 Spectral Bounds

We derive the following eigenvalue bound:

Theorem 4 (Spectral Method).

$$J_G^* \geq \left\lfloor \frac{n}{k} \right\rfloor \sum_{i=1}^{k} \lambda_i(\tilde{L}) - 2kM. \qquad (4)$$

Proof. We relax the topological constraint $X \in O_G$, and instead constrain over orthogonal X. We thus have for any k:

$$J_G^* \geq \mathrm{tr}(X^T \tilde{L} X W^{(k)}) - 2kM \quad \text{s.t } X^T X = X X^T = I.$$

For symmetric $\tilde{L}, W$ and orthogonal matrix X, where $\lambda_1, ..., \lambda_n$ and $\mu_1, ..., \mu_n$ are the eigenvalues in increasing order of $\tilde{L}$ and W respectively, we have $\mathrm{tr}(X^T \tilde{L} X W) \geq \sum_{i=1}^{n} \lambda_i \mu_{n-i}$, or the minimal dot product of λ and μ (see [10], Theorem 3). Here $W^{(k)}$ is a block diagonal matrix, with $n - k$ zero eigenvalues and k eigenvalues that are at least $\lfloor n/k \rfloor$. Therefore, we apply our lower bound as a sum of the first k eigenvalues of $\tilde{L}$:

$$J_G^* \geq \max_k \mathrm{tr}(X^T \tilde{L} X W^{(k)}) - 2kM \geq \sum_{i=1}^{k} \lfloor n/k \rfloor \lambda_i(\tilde{L}) - 2kM,$$

□

This bound can be found in $O(n^3)$ time. We first find the eigenvalues $\lambda(\tilde{L})$ in $O(n^3)$. We then iterate over possible values of k which takes constant time per iteration to find the best eigenvalue. However, we generally only need small number of eigenvalues to find a good k. Since any value of k is a lower bound, it suffices to find the h smallest eigenvalues of L. These values can be found using a method such as Lanczos-Arnoldi with time complexity $O(hn^2)$: this complexity decreases even further with sparse L using sparse eigenvalue solvers.

For closed form analysis, sometimes the exact form of the original Laplacian spectra $\lambda(L)$ are known, but the spectra of our out-degree normalized Laplacian $\lambda(\tilde{L})$ are not. While $\lambda(\tilde{L})$ can be easily computed automatically, they can be harder to derive for closed form analysis. We can naturally loosen the bound in Theorem 4 to be in terms of L rather than $\tilde{L}$.

Theorem 5 (Spectral Method with Original Graph Laplacian).

$$J_G^* \geq \frac{1}{\max_{v\in V} d_{out}(v)} \left\lfloor \frac{n}{k} \right\rfloor \sum_{i=1}^{k} \lambda_i(L) - 2kM. \qquad (5)$$

Proof. We follow the same steps of Theorem 4, but we bound Equation 1 as

$$J_G^* \geq \min_{X\in O_{\mathcal{G}}} \max_{P\in\mathcal{P}_X} \left(\sum_{S\in P} \frac{|\partial S|}{\max_{v\in V} d_{out}(v)} \right) - 2M|P|.$$

We can then reframe Equation 3 in terms of L, noting that, if $x \in \mathbb{R}^n$ is the one-hot encoding of $S \subseteq V$, then $x^T L x = |\partial S|$. Using the same partitioning argument and definition of $W^{(k)}$, we can reframe the quadratic program in Theorem 3 with L instead of $\tilde{L}$:

$$\text{minimize}_X \quad \max_k \frac{\operatorname{tr}(X^T L X W^{(k)})}{\max_{v\in V} d_{out}(v)} - 2kM$$
$$X \in \mathcal{O}_G.$$

Then, following the same spectral argument as in Theorem 4, we arrive at a looser, but more convenient bound:

$$J_G^* \geq \frac{1}{\max_{v\in V} d_{out}(v)} \left\lfloor \frac{n}{k} \right\rfloor \sum_{i=1}^{k} \lambda_i(L) - 2kM.$$

□

4.4 Parallel Spectral Bounds

We generalize Theorem 4 to the parallel setting as follows. Suppose that we have p processors, each with memory M. As in [3, 12] we count I/O as the communication with a processor to slow memory or between processors. We make no assumptions about the distribution of the workload.

Theorem 6 (Parallel Spectral Bound). *For a computation graph G distributed across p processors, at least one of the processors has I/O J_G^* lower bounded by:*

$$J_G^* \geq \left\lfloor \frac{n}{kp} \right\rfloor \sum_{i=1}^{k} \lambda_i(\tilde{L}) - 2kM. \quad (6)$$

Proof. For a given evaluation of G, each vertex in G is evaluated by one processor. Let $V_1, ..., V_p$ be the vertex sets associated with each processor. Then given the optimal evaluation order X we can define p evaluation orders $X_1, ..., X_p$ where X_i indicates the evaluation order of processor i on its vertex set V_i. Since memory is local to each processor, we can use the same graph partitioning machinery in Theorem 4 per-processor. For processor i, if $\mathcal{P}_{X_i}$ is the set of valid partitions over X_i, the I/O of processor i is lower bounded by

$$J_G(X_i) \geq \max_{P\in\mathcal{P}_{X_i}} \left(\sum_{S\in P} |R_S| + |W_S| \right) - 2M|P|.$$

There must exist one processor i^* for which $|V_{i^*}| \geq n/p$. For this processor, we can partition V_{i^*} into k equal parts of $\frac{n}{kp}$ vertices each (call this partition P). Since any partition of V_{i^*} is a lower bound, we have

$$J_G(X_{i^*}) \geq \left(\sum_{S\in P} \sum_{(u,v)\in\partial S} \frac{1}{d_{out}(u)} \right) - 2kM.$$

However, as a looser lower bound, instead of restricting P to equal partitions of V_{i^*}, we can consider the set of equal partitions of the entire graph into kp parts of $\frac{n}{kp}$, and simply pick the k sections incurring the least I/O cost. This is then equivalent to the bound in Theorem 4 with kp partitions, but we only count I/O from the first k parts (which correspond to the smallest k eigenvalues). We then have

$$J_G^* \geq \left\lfloor \frac{n}{kp} \right\rfloor \sum_{i=1}^{k} \lambda_i(\tilde{L}) - 2kM.$$

□

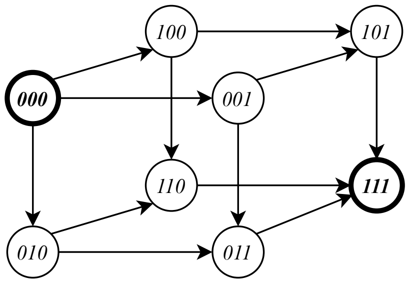


Figure 4: Bellman-Held-Karp computation graph with 3 cities. The graph is a 3 dimensional boolean hypercube, with starting point 000 and ending point 111.

5 ANALYTICAL BOUNDS FOR SPECIFIC GRAPHS

For computation graphs with known eigenvalues, we can compute the bound in Theorem 5 directly. We perform this analysis for the Bellman-Held-Karp algorithm for the Traveling Salesman Problem and as well as the Fast Fourier Transform, which have a hypercube and butterfly computation graph respectively. For both of these problems, we consider solely *nontrivial I/O*, which does not count reading inputs or writing outputs. In the process, we derive the spectrum of the FFT graph; to our knowledge, this is the first closed form of the spectrum of the unwrapped butterfly graph that includes multiplicity. Finally, we present a probabilistic bound on the I/O of a random Erdős Rényi graph.

Previously, [14] found an asymptotically tight bound of $\Omega(\frac{l2^l}{\log M})$ for a 2^l point FFT through manual inspection of $2S$ partitions. We find that our spectral bound of $\Omega(\frac{l2^l}{\log^2 M})$ is only a factor of $1/\log M$ off from this published tight bound.

5.1 Hypercube Graph

The hypercube is a computation graph that appears as a result of many hard dynamic programming problems [1]. For example, consider the well-known Bellman-Held-Karp algorithm which uses dynamic programming to solve the traveling salesman problem [4, 11]. The approach solves for the optimal path visiting a subset of the cities by leveraging the computed optimal paths through adjacent subsets with one fewer city.

The Bellman-Held-Karp algorithm with l cities can be naturally formulated as an iteration on the vertices of a boolean hypercube. We encode "cities visited" as a length l binary string. Let Q_l be a boolean l-dimensional hypercube, where each vertex k is a length l binary string, and $(k_1, k_2) \in E$ if k_2 can be constructed by setting a 0 in k_1 to 1. Let $S(k, i)$ be the shortest path visiting all the cities active in k and ending up at the i'th city. Then to solve TSP, we seek to find the solution set $Y[k] = \{S(k, i) \mid k[i] = 1\}$ for each vertex k in the boolean hypercube Q_l. For example, $Y[01101]$ contains three paths that have traversed cities 2, 3, and 5, where each path has a different ending point. $Y[k]$ can be easily computed given the results of k's incoming neighbors in Q_l, and $Y[\{1\}^l]$ gives the solution to the TSP. Thus Q_l represents the computation graph for the under this formulation of the Bellman-Held-Karp algorithm. An example of this graph can be seen in Figure 4.

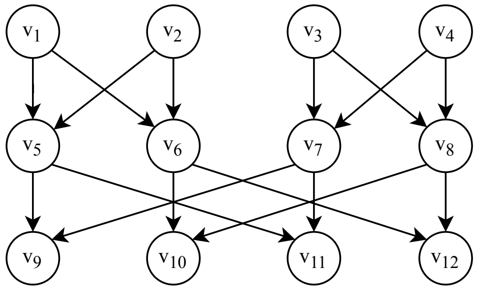


Figure 5: Computation graph for a $2^l = 4$ point Fast Fourier Transform.

The I/O bound for this formulation of the Bellman-Held-Karp algorithm can then be found via our spectral method, because the hypercube has relatively simple eigenvalues. The l-dimensional hypercube has $n = 2^l$ vertices and Laplacian eigenvalues $2i$ for $i = 0, \ldots, l$ with multiplicity $\binom{l}{i}$. If we choose k to encompass the top eigenvalues up to $i = \alpha$, we have $k = \sum_{i=0}^{\alpha} \binom{l}{i}$. The maximal out degree is l. For any $\alpha < 2^l$:

$$J_G^* \geq \frac{1}{\hat{d}_{out}} \frac{2^{l+1}}{\sum_{i=0}^{\alpha} \binom{l}{i}} \sum_{i=0}^{\alpha} i \binom{l}{i} - 2M \sum_{i=0}^{\alpha} \binom{l}{i}$$
$$= \sum_{i=0}^{\alpha} \binom{l}{i} \left(i \frac{2^{l+1}}{l \sum_{i=0}^{\alpha} \binom{l}{i}} - 2M \right).$$

While any $\alpha < l$ would be a lower bound, for simplicity we here choose $\alpha = 1$ (i.e $k = l + 1$):

$$J_G^* \geq \frac{2^{l+1}}{(l+1)} - 2M(l+1).$$

For a tighter bound we can optimize more specifically over α. We see that this bound is nontrivial as long as $M \leq \frac{2^l}{(l+1)^2}$.

5.2 Fast Fourier Transform Graph

The Fast Fourier Transform (FFT) computation graph is a butterfly graph. For a 2^l point FFT, the butterfly graph B_l has $(l+1)2^l$ vertices, which can be arranged into $l + 1$ columns. A $2^l = 4$ point FFT can be seen in Figure 5. The butterfly graph can be defined inductively: allow B_0 to be defined as a single vertex. Then B_l can be constructed as two copies of B_{l-1} that are joined by an extra final column of $(l + 1)$ vertices.

We derive the eigenvalues of the Laplacian of B_l (see the Appendix of the full version of this paper, [13]). To our knowledge, closed forms with multiplicities were only previously known for the wrapped butterfly graph [7]. The eigenvalues we derive are:

$$4 - 4\cos\left(\frac{\pi j}{l+1}\right), \forall j = 0, \ldots, l;\ \text{multiplicity } 1$$
$$4 - 4\cos\left(\frac{\pi(2j+1)}{2i+1}\right) \forall i = 1, \ldots, l; j = 0, \ldots, i-1;\ \text{multiplicity } 2^{l-i+1}$$
$$4 - 4\cos\left(\frac{j\pi}{i+1}\right) \forall i = 1, \ldots, l-1; j = 1, \ldots, i; \text{multiplicity } (l-i)2^{l-i-1}.$$

The smallest eigenvalue is 0 (from the first expression), but the next eigenvalues are governed by the second expression with $j = 0$ as long as $2i + 1 \geq l + 1$. We choose some α and set $k = 2^{\alpha+1}$. We compute the lowest k eigenvalues of the Laplacian of B_l. Of these eigenvalues, 2^α have (with $i = l - \alpha$):

$$\lambda = 4 - 4\cos\left(\frac{\pi}{2(l-\alpha)+1}\right).$$

To compute our lower bound, we assume the other eigenvalues are 0. We note that $n = (l + 1)2^l$. Then we have (dividing by our maximal out-degree 2):

$$J_G^* \geq (l+1)2^l \left(1 - \cos\left(\frac{\pi}{2(l-\alpha)+1}\right)\right) - 2^{\alpha+2}M.$$

Suppose that we set $\alpha = l - \log_2 M$, under the assumption that $M \ll l$. Then:

$$J_G^* \geq (l+1)2^l \left(1 - \cos\left(\frac{\pi}{2\log_2 M + 1}\right) - \frac{4}{l+1}\right).$$

To see how this behaves, we can use the small angle approximation $\theta^2/2 \approx 1 - \cos(\theta)$ for small θ to get:

$$J_G^* \geq (l+1)2^l \left(\frac{\pi^2}{8\log_2^2 M} - \frac{4}{l+1}\right).$$

Thus, for large M and l where $M \ll l$, our bound behaves at least as well as $\Omega(\frac{l2^l}{\log^2 M})$. This bound is only a $1/\log_2 M$ factor worse than the tight lower bound for butterfly graphs: $\Omega(\frac{l2^l}{\log M})$, which is computed by inspection on the specific graph using S-partitions [14].

5.3 Random Graphs

The spectral bound is flexible, and can perform well on most graphs with high connectivity regardless of its structure. In the following section, we characterize the performance of this spectral bound given a random graph, and show that the spectral bound provides nontrivial results for as long as the graph is well-connected. While this graph is not a specific computation graph, examining random graphs allows us to understand the performance of the bound as we increase the connectivity of the graph.

We consider an Erdős Rényi graph $G(n, p)$ on n vertices, where each edge is determined by flipping a coin with probability p. We will only deal with the regime where $p \geq \frac{\log n}{n}$, where the graph is almost surely connected [15].

We begin with the case where $p = \Theta(\frac{\log n}{n})$, but the graph is still connected. More specifically, we specify $p = p_0 \frac{\log n}{n-1}$ for some $p_0 > 6$. By [15], in this regime:

$$\lambda_2 \sim p_0 \log n \left(1 - \sqrt{\frac{2}{p_0}} + O(\frac{1}{p_0}) + O(\frac{1}{\sqrt{p_0 \log n}})\right).$$

We first concentrate the maximum degree of the graph using Chernoff's bound as in [6]. We first note that the degree d of a single vertex is governed by the sum of $n - 1$ Bernoulli random variables with probability p. The expected degree is $\mu = p(n - 1) = p_0 \log n$. Then using Chernoff's bound, we have:

$$P(d \geq (1+\delta)\mu) \leq \exp(-\mu\delta^2/3)$$
$$P(d \geq (1+\delta)p_0 \log n) \leq \exp(\frac{-\delta^2 p_0 \log n}{3}).$$

If we set $\delta = \sqrt{6/p_0}$, we concentrate individual degrees as $P(d \geq (1+\sqrt{6/p_0})p_0 \log n) \leq 1/n^2$. Then, using the union bound, we can concentrate the maximum degree as:

$$P(d_{max} \geq (1+\sqrt{6/p_0})p_0 \log n) \leq 1/n$$

Thus, with high probability ($1/n \to 0$ as $n \to \infty$), we have

$$d_{max} \geq (1+\sqrt{6/p_0})p_0 \log n.$$

Setting $k = 2$ in Theorem 5, we have with high probability:

$$J_G^* \leq \frac{n}{(1+\sqrt{6/p_0})}\left(1 - \sqrt{\frac{2}{p_0}} + O(\frac{1}{p_0}) + O(\frac{1}{\sqrt{p_0 \log n}})\right) - 4M.$$

As $n \to \infty$ this bound scales roughly with $n + \frac{n}{\sqrt{\log n}}$, and is linear in M. Our bound becomes weaker, but still nontrivial, when we consider regimes with higher p. This is because as p increases, the maximum degree scales to almost np (and our bound requires dividing by the maximum out degree). For example, consider the case where $\frac{np}{\log n} \to \infty$, as in this regime the graph is essentially regular with degree np. Then from [15], we have that with high probability as $n \to \infty$:

$$\lambda_2(L) = np + O(\sqrt{np \log n}).$$

We then can apply Theorem 5 to lower bound the non-trivial I/O (setting $k = 2$) and dividing by the max degree np:

$$J_G^* \geq \frac{n}{2}(1 + O(\sqrt{\frac{\log n}{np}})) - 4M.$$

As $n \to \infty$, $O(\sqrt{\frac{\log n}{np}})$ will decay to zero resulting in a bound linear in n.

6 EVALUATION

6.1 Solver

We evaluate our two lower bounds on four common computation graphs. To facilitate our evaluation, we develop a solver that traces operations during a Python computation and thus extracts a computation graph [1]. The solver inter-operates with standard arithmetic operations and supports the inclusion of custom operations.

When computing Theorem 4, any choice of $k < n$ produces a valid bound. We set $h = 100$, computing up to the first 100 values of the graph Laplacian, and choose the optimal k from $k \in \{2 \ldots h\}$. We discuss this choice in Section 6.5.

6.2 Evaluation Computation Graphs

We evaluate the following graphs. Examples of these graphs can be found in Figure 6.

(1) *Fast Fourier Transform (FFT):* We evaluate the l level FFT of an 2^l element array, which is a butterfly graph. This graph has a published bound [14] of

$$\Omega\left(l2^l/\log M\right).$$

[1]Our code can be found at https://github.com/stanford-futuredata/graphIO

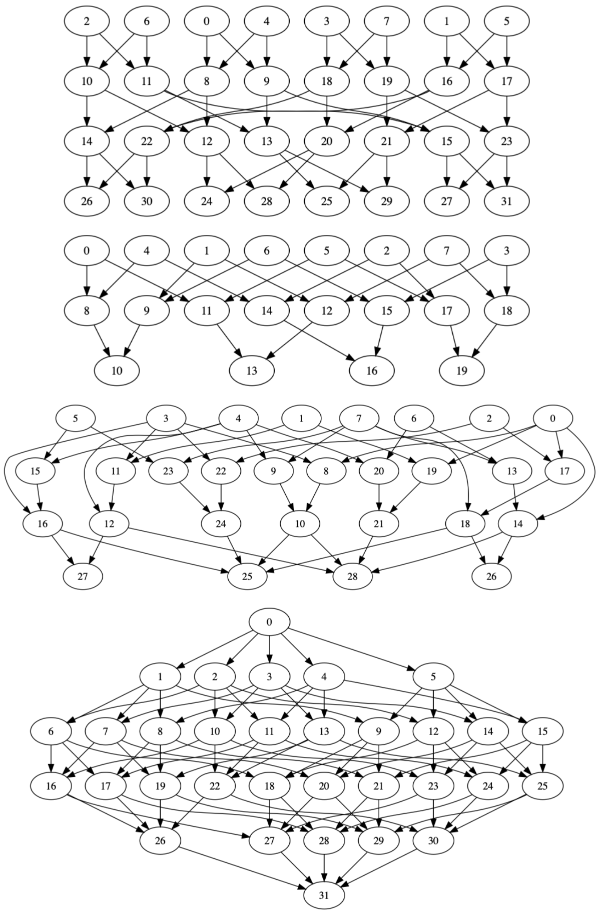


Figure 6: Computation Graphs for (top to bottom): 8 point FFT, 2×2 Naive Multiplication, 2×2 Strassen Multiplication, 5 city Bellman-Held-Karp algorithm for TSP

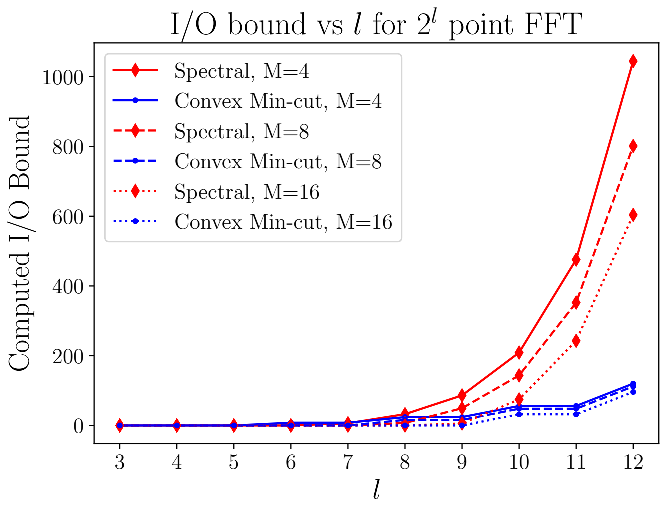
(2) *Naive Matrix Multiplication:* We evaluate the graph formed by naive multiplication of two $n \times n$ matrices $C = AB$. Specifically, we compute C_{ij} as the dot product of the ith row of A and the jth column of B. This graph has a published bound [12]

$$\Omega\left(n^3/\sqrt{M}\right).$$

(3) *Strassen Multiplication* We evaluate the graph formed by multiplying to $n \times n$ matrices $C = AB$ via Strassen's method. Since Strassen's method is a recursive method that splits matrices into quadrants, we evaluate on values of n that are powers of 2. This graph has a published bound [3] of

$$\Omega\left(\left(n/\sqrt{M}\right)^{\log_2 7} M\right).$$

(4) *Bellman-Held-Karp* We evaluate the hypercube computation graph formed by performing the Bellman-Held-Karp algorithm for a l city TSP. We could not find a prior I/O bound for this problem in the current literature. However, in Section



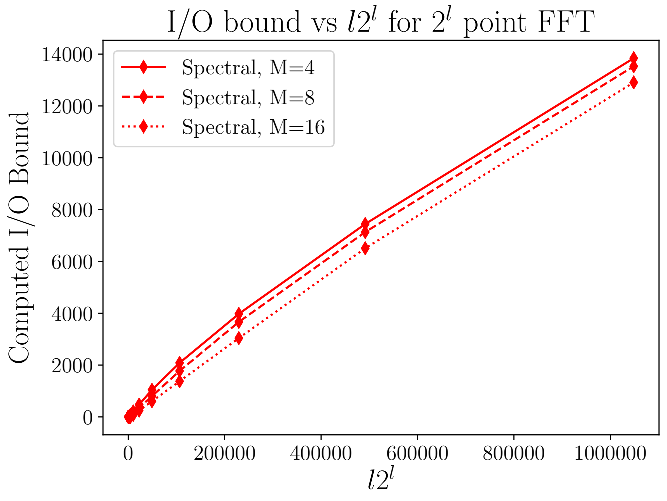
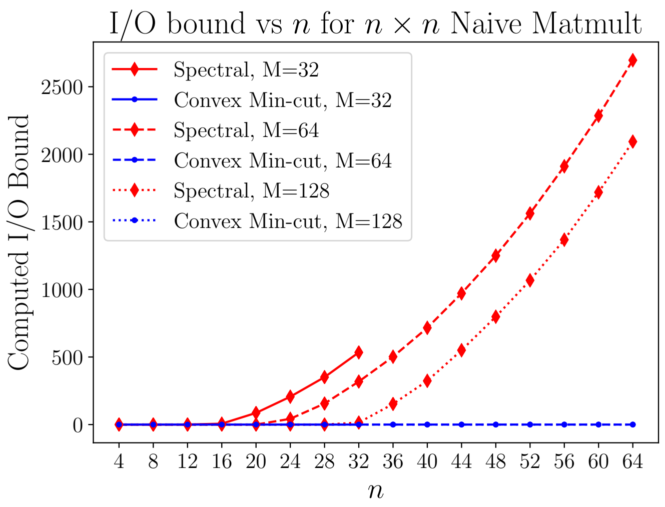


Figure 7: FFT: Bound vs l (top) and $l2^l$ (bottom) for $M = 4, 8, 16$; l = FFT Level. Max in-degree 2



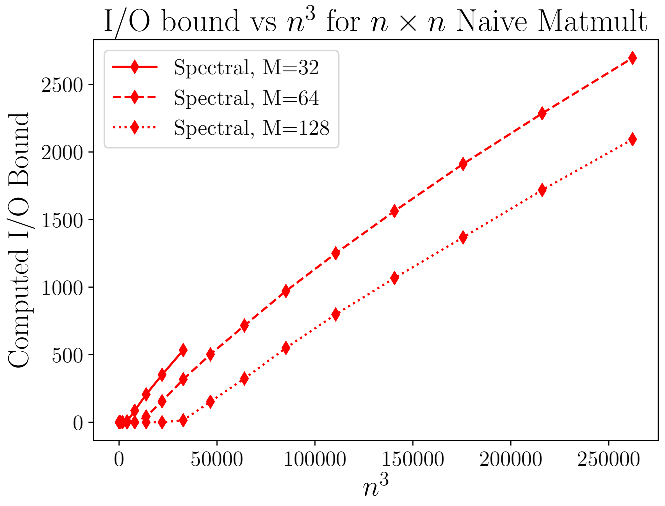


Figure 8: Naive matrix multiplication: Bound vs n (top) and n^3 (bottom) for $M = [32, 64, 128]$; n = side length. Max in-degree n.

5.1 we derive using the spectral method a bound of:

$$\Omega\left(\left(2^l/l\right) - 2Ml\right).$$

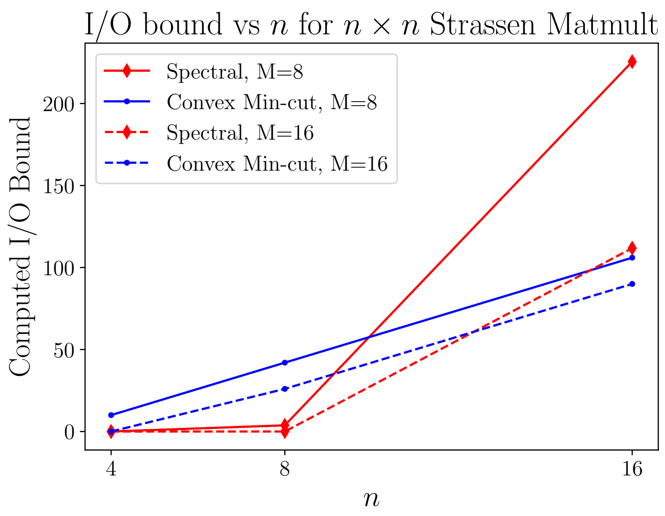
6.3 Baselines

The only current methods for creating automatic lower bounds for any arbitrary graph that we could find are the convex min-cut method [9] and the $2S$ partition method [8] (see Section 2). The $2S$ partition method involves solving a Mixed Integer Program, and is combinatorial in complexity: we could thus not perform this method for large graphs. The convex min-cut method is polynomial in time but still extremely expensive at $O(n^5)$. We evaluate the convex min-cut method for as large graphs as possible, cutting off evaluation at 1 day.

Convex-Min Cut: The convex min-cut method transforms the graph with respect to a vertex v into a flow problem and then finds the minimum s-t cut of the transformed graph. The method maximizes over all v in the graph. The method decomposes trivial (reading inputs and writing outputs) and non-trivial I/O, and thus fits well with our problem set-up. The runtime of this bound is $O(n^5)$ where n is the number of vertices in the graph. In order to reduce runtime, the authors suggest loosening the bound by partitioning the graph into smaller sub-graphs using METIS and running convex min-cut on each sub-graph. If $C(v, G)$ is the minimum convex cut for G transformed with respect to v, and $\mathcal{P}$ is the partition reported by METIS, the authors report the bound:

$$J_G^* \geq \sum_{P \in \mathcal{P}} \max_{v \in P} \max(0, 2 * (C(v, G) - M)).$$

More details can be found in their paper. The authors suggest that each sub-graph have at most $2 * M$ vertices, and evaluate their bounds on a series of small, simple computation graphs with very uniform structure. However, when evaluating on more complex



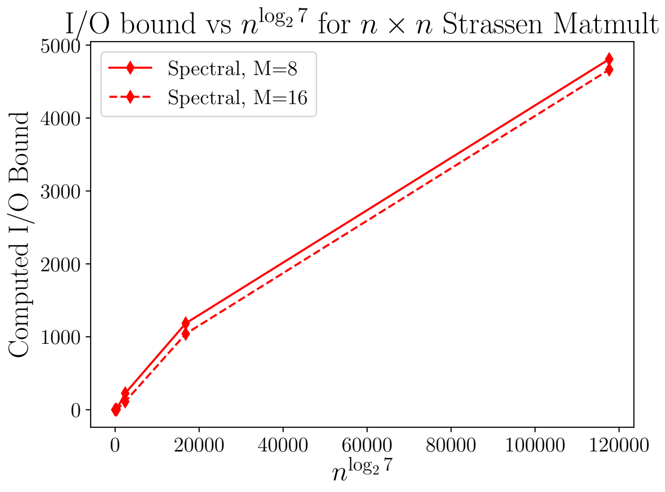
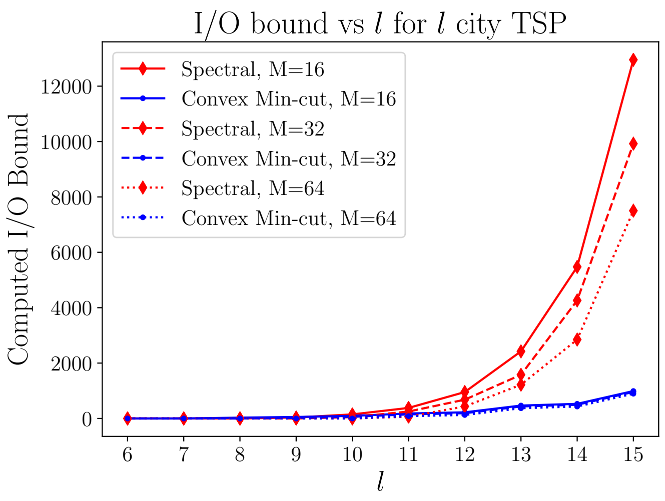


Figure 9: Strassen: Bound vs n (top) and $n^{\log_2 7}$ (bottom) for $M = [8, 16]$; n = side length. Max in-degree 4.



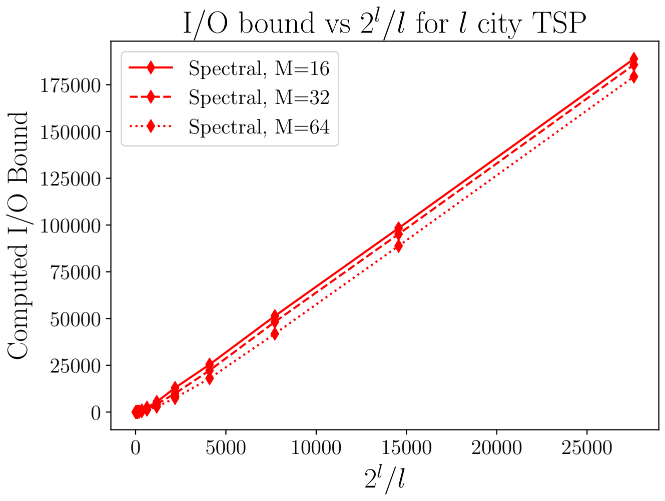


Figure 10: Bellman-Held-Karp for TSP: Bound vs l (top) and $2^l/l$ (bottom) for $M = 16, 32, 64$; l = number of cities. Max in-degree l

computation graphs such as matrix multiplication or FFT, we found that the above bound gave trivial results ($J^* \geq 0$) for every one of the graphs. We hypothesize that the prescribed sub-graph size of $2 * M$ is too small for more complex graphs . In our evaluation, we display results of the convex min-cut method run over the entire graph (without partitioning):

$$J^*_G \geq \max_{v \in V} \max(0, 2 * (C(v, G) - M)).$$

The above bound is linear in M for any graph. In the worst case, this bound can take $O(n^5)$ time to compute.

6.4 Bound Behavior vs Graph Sizes

We examine graph behavior for varying graph sizes and varying M. We plot the the spectral method and the convex min-cut method against the graph size parameter (l for the 2^l FFT, the matrix side length n for matrix multiplication, and the number of cities for the TSP).

To compare against the published bounds, we also plot the computed I/O for the spectral bound against the graph parameter term in the analytical bounds in Section 6.2. For example, we plot the computed I/O for the 2^l point FFT graph against $l2^l$. If our bounds follow the growth patterns of the analytical bounds, then these plots should be roughly linear. We do not display points where the maximum in-degree is greater than M, because then the computation of some operations would not fit all their operands inside fast memory.

For all four graphs, we find that the bound computed from the spectral method is both tighter and more scalable than the convex min-cut method. In particular, the convex min-cut method is trivial for the naive matrix multiplication graph.

Moreover, we find that our bounds roughly match the analytical growth of the published bounds, as the I/O vs the published bound is roughly linear for all four graphs. Finally, we note that our bound

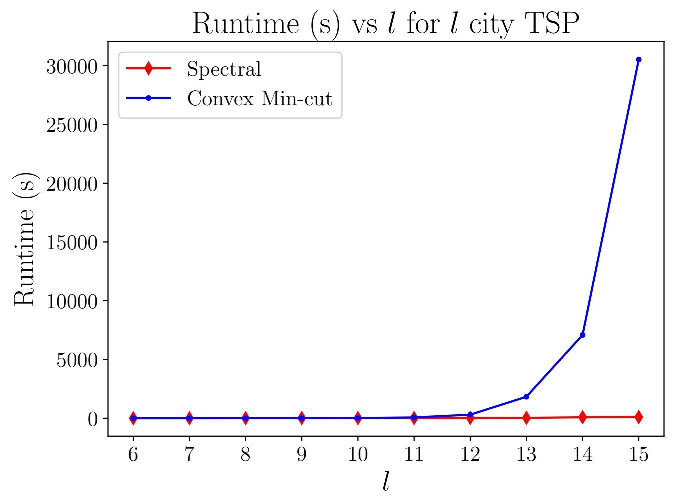


Figure 11: Runtime in seconds for computing the lower bound for Bellman-Held-Karp on a l city TSP for various l.

does not significantly degrade with M, and can thus be computed for large memory sizes.

6.5 Scalability

Our spectral method is fast to compute, with a runtime complexity of $O(hn^2)$ where $h \leq n$ is the number of eigenvalues computed. Since any number of partitions k in Theorem 4 gives a valid lower bound, h can be set to trade off the bound strength with runtime complexity, with a maximum runtime of $O(n^3)$ by computing all the eigenvalues of the graph Laplacian. However, empirically we found that even when computing all of the eigenvalues, the best k is usually far below 100 even for large graphs, so the higher level eigenvalues remain unused: we therefore can set $h = 100$ without losing bound strength. In contrast, the convex min-cut method has runtime complexity of $O(n^5)$, which does not scale well to large graphs. In Figure 11, we plot the runtime in seconds of computing the convex min-cut and the spectral method for successively larger l for evaluating Bellman-Held-Karp on an l city TSP. We find that the convex-min-cut runtime explodes, taking close to 8.5 hours for the 15 city TSP, while our spectral method takes 98 seconds.

7 CONCLUSION

Finding I/O bounds for general computations remains a challenging problem. In this paper, we propose a novel method to find I/O bounds for computation graphs, using the spectra of the graph Laplacian. The spectra can be computed efficiently even for large graphs and can also be computed in closed form for some graphs, yielding a proof technique to find new closed-form bounds. We used the spectral method to derive closed-form bounds for several graphs, including the hypercube for the Bellman-Held-Karp algorithm and the butterfly graph for the Fast Fourier Transform. We evaluated our method empirically on four computation graphs and showed that it finds tighter bounds than previous automated methods at a fraction of the runtime and behaves similarly to published analytical bounds.

ACKNOWLEDGMENTS

We thank Pratiksha Thaker, Moses Charikar, and Guillermo Angeris for their advice and feedback on this work.

This research was supported in part by affiliate members and other supporters of the Stanford DAWN project—Ant Financial, Facebook, Google, Infosys, NEC, and VMware—as well as Cisco, SAP, and the NSF under CAREER grant CNS-1651570. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Andris Ambainis, Kaspars Balodis, Jānis Iraids, Martins Kokainis, Krišjānis Prūsis, and Jevgēnijs Vihrovs. 2019. Quantum speedups for exponential-time dynamic programming algorithms. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1783–1793.

[2] Grey Ballard, Aydin Buluc, James Demmel, Laura Grigori, Benjamin Lipshitz, Oded Schwartz, and Sivan Toledo. 2013. Communication optimal parallel multiplication of sparse random matrices. In *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*. ACM, 222–231.

[3] Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. 2012. Graph expansion and communication costs of fast matrix multiplication. *Journal of the ACM (JACM)* 59, 6 (2012), 32.

[4] Richard Bellman. 1962. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM (JACM)* 9, 1 (1962), 61–63.

[5] Gianfranco Bilardi and Franco P Preparata. 1999. Processor-Time Tradeoffs under Bounded-Speed Message Propagation: Part II, Lower Bounds. *Theory of Computing Systems* 32, 5 (1999), 531–559.

[6] Fan Chung and Mary Radcliffe. 2011. On the spectra of general random graphs. *the electronic journal of combinatorics* (2011), P215–P215.

[7] Francesc Comellas, Miquel Angel Fiol, Joan Gimbert, and Margarida Mitjana. 2003. The spectra of wrapped butterfly digraphs. *Networks: An International Journal* 42, 1 (2003), 15–19.

[8] Venmugil Elango. 2016. *Techniques for Characterizing the Data Movement Complexity of Computations*. Ph.D. Dissertation. The Ohio State University.

[9] Venmugil Elango, Fabrice Rastello, Louis-Noël Pouchet, J Ramanujam, and P Sadayappan. 2013. *Data access complexity: The red/blue pebble game revisited.* Technical Report. Technical Report OSU-CISRC-7/13-TR16, Ohio State University.

[10] Gerd Finke, Rainer E Burkard, and Franz Rendl. 1987. Quadratic assignment problems. In *North-Holland Mathematics Studies*. Vol. 132. Elsevier, 61–82.

[11] Michael Held and Richard M Karp. 1962. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied mathematics* 10, 1 (1962), 196–210.

[12] Dror Irony, Sivan Toledo, and Alexander Tiskin. 2004. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel and Distrib. Comput.* 64, 9 (2004), 1017–1026.

[13] Saachi Jain and Matei Zaharia. 2020. Spectral Lower Bounds on the I/O Complexity of Computation Graphs. *arXiv preprint arXiv:1909.09791* (2020).

[14] Hong Jia-Wei and Hsiang-Tsung Kung. 1981. I/O complexity: The red-blue pebble game. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*. ACM, 326–333.

[15] Theodore Kolokolnikov, Braxton Osting, and James Von Brecht. 2014. Algebraic connectivity of Erdös-Rényi graphs near the connectivity threshold. *Manuscript in preparation* (2014).

[16] Jacob Scott, Olga Holtz, and Oded Schwartz. 2015. Matrix multiplication I/O-complexity by path routing. In *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures*. ACM, 35–45.