EasyChair Preprint

№ 237

MIDAS: Multilinear Detection at Scale

Saliya Ekanayake, Jose Cadena, Udayanga Wickramasinghe and Anil Kumar Vullikanti

June 6, 2018

MIDAS: Multilinear Detection at Scale

Saliya Ekanayake*, Jose Cadena†, Udayanga Wickramasinghe‡ and Anil Kumar Vullikanti†
*Network Dynamics and Simulation Science Laboratory (NDSSL), Virginia Tech
Email: esaliya@vt.edu
†Department of Computer Science and Biocomplexity Institute, Virginia Tech
Email: {jcadena,vsakumar}@bi.vt.edu
‡Department of Computer Science, Indiana University, Bloomington
Email: uswickra@indiana.edu

***Abstract*—We focus on two classes of problems in graph mining here: (1) finding trees and (2) anomaly detection using network scan statistics in complex networks. These are fundamental problems in a broad class of applications. Most of the parallel algorithms for such problems are either based on heuristics, which do not scale very well, or use techniques like color coding, which have a high memory overhead. In this paper, we develop a novel approach for parallelizing both these classes of problems, using an algebraic representation of subgraphs as monomials—this methodology involves detecting multilinear terms in multivariate polynomials. Our algorithms show good scaling over a large regime, and they run on networks with close to half a billion edges. The resulting parallel algorithm for trees is able to scale to subgraphs of size 18, which has not been shown before, and it significantly outperforms the best prior color coding based method (FASCIA) by more than two orders of magnitude. Our algorithm for network scan statistics is the first such parallelization, and it is able to handle a broad class of scan statistics functions (both parametric and non-parametric), with the same approach.**

I. Introduction

Many problems in graph mining and social network analysis can be reduced to questions about different kinds of subgraphs; two important classes of such problems, which are the focus of our paper, are: (1) *Detecting and counting subgraphs*, such as paths and trees, of a given size k—these are used for characterizing different kinds of networks, especially in biological models [1], [2]. (2) *Anomaly detection in network data using graph scan statistics*, which involves finding connected subgraphs of size k, optimizing different kinds of anomaly score functions [3], [4], [5], [6]—this arises in a number of applications, such as social network analysis, epidemiology, finance, and bio-surveillance (see [7] for a survey).

Both problems are computationally very challenging. For instance, exact detection of paths is NP-hard and the corresponding counting problem is #P-hard. Development of parallel algorithms for these problems has been an active area of research. Many parallel algorithms exist for counting local subgraphs, such as triangles [8], [9], [10], [11]. Finding trees is much harder, and a number of heuristics have been developed. One of the few techniques that gives rigorous approximation guarantees is *color coding* [1], [12], [2]. Parallel adaptations have been developed using MapReduce [13] and MPI [14], [15]. The MPI based FASCIA algorithm [14], [15] is the state-of-the-art in terms of counting trees in massive networks, scaling to finding trees with up to 12 vertices in networks with one billion edges. However, it seems very challenging to scale the color coding method to larger subgraphs, even on small networks. The main reason is that the time and space complexity of the color coding technique both scale as 2^k, where k denotes the subgraph size. In this paper, we take the first steps towards beating this bound, which has remained a significant open problem since [15]. Our approach involves parallelization of a powerful algebraic technique for detecting multilinear terms in a multivariate polynomial, developed by Koutis [16] and Williams [17].

Optimizing network scan statistics leads to challenging optimization problems as well. Color coding has also been used to develop the first method with rigorous approximation guarantees [18]; however, this has not been parallelized because of its high memory overhead. In [19], we have developed a parallel adaptation of the multilinear detection technique using both GraphX and Giraph. However, none of these scaled beyond networks with 40 million edges.

In this paper, we develop a distributed algorithm for multilinear detection, which immediately leads to highly scalable algorithms for both paths and trees, and network scan statistics. Our contributions are:

1. MIDAS: Distributed algorithm for multilinear detection and applications to subgraph analysis. We develop MIDAS, a distributed MPI based algorithm for finding paths and trees through detection of multilinear terms with k variables of the form $x_{i_1}x_{i_2}\dots x_{i_k}$ (i.e., a term in which all variables have exponent 1) in a multivariate polynomial $P(x_1,\dots,x_n)$. The sequential algorithm uses a matrix representation of a group algebra (as will be described later) [16], [17], and its structure lends itself to a natural parallelization. We give rigorous bounds on the performance in terms of the time and space complexity, which scale as $O(2^k)$ and $O(k)$, respectively, compared with $O(2^k e^k)$ and $O(2^k)$ for color coding, respectively [13], [14], [15]. For random graphs, we show a rigorous scaling with N, the number of processors for $N \leq 2^k$.

2. Cache optimization and weak scalability. Our algorithm partitions the graph G into N_1 parts. The computation involves 2^k iterations, and N_1 processors together perform one iteration—this allows us to schedule N/N_1 such computations to occur in parallel. The total compute time exhibits good weak scaling. Additionally, our data structures for supporting Galois

field operations during the iterations support a temporal cache locality, which actually leads to a reduction in the compute time as N_1 increases. On the other hand, the communication cost increases with N_1, leading to an optimal value of N_1 for the best performance.

3. Experimental results. We evaluate our results on a number of real and synthetic networks with up to 250 million edges and subgraph sizes up to $k = 18$. The reduced memory footprint allows us to scale to paths of size 18, which has not been done before. Our algorithms for both problems show reasonable scaling up to 512 processors, supporting our theoretical analysis. The running time grows linearly with the network size and as 2^k for the subgraph size k.

4. Comparison with prior methods. Our algorithm for finding paths gives over two orders of magnitude improvement in time compared to FASCIA, the state of the art method based on color coding [14], [15]. Our algorithm for scan statistics improves on the Giraph based implementation [19] by over an order of magnitude, and it scales to significantly larger networks.

One additional advantage is that our parallel algorithm based on multilinear detection is conceptually much simpler than color coding based algorithms, though it requires the language of algebra. It also requires far less bookkeeping than color coding. As we discuss later in Section IV, the obvious ways to try to parallelize multilinear detection do not perform well; instead, we find that a careful interplay between the degree of partitioning, as well as batching a set of iterations and the data structures help yield the optimal results.

II. Preliminaries

A. Problem Formulation and Notation

We will focus primarily on the following two classes of problems in this paper; our approach can be extended more broadly.

1) Finding Paths and Trees: Given a graph $G = (V, E)$ with $n = |V|$, $m = |E|$, and a subgraph $H = (V^H, E^H)$, with $k = |V^H|$, find a mapping $f : V^H \rightarrow V$, such that $(i, j) \in E_H$ *only if* $(f(i), f(j)) \in E$. Such a mapping is referred to as a *non-induced embedding* of H in G.

Problem 1: (k-Tree) Given a weighted graph $G = (V, E)$ with a weight vector $\mathbf{w}$, and a tree denoted by $H = (V^H, E^H)$ with $|V^H| = k$, the objective is to determine if there exists an embedding of H in G.

We will consider the following approximate version of problem 1: determine if a non-induced embedding exists with probability at least $1 - \epsilon$, where $\epsilon \in (0, 1)$ is a parameter. Other common variants of this problem are: (1) counting all embeddings, and (2) finding a maximum weight embedding in a weighted version of the graph; our approach can be extended to all these variants.

2) Anomaly Detection Using Graph Scan Statistics: We use the notation of [18] here. We assume each node $v \in V$ has two associated values, which vary with time (we will not show the time to avoid complicating the notation): (1) a *baseline count*, $b(v)$, which indicates the count that we expect to see at the node v—e.g., the number of people in a county corresponding to node v—and (2) an *event count* or *weight*, $w(v)$, which indicates how many occurrences of an event of interest are seen at the node—e.g., the number of cases of a disease in a county.

Graph scan statistics are among the most commonly used methods for detecting anomalies or "hotspots" in network data [3], [4], [5], [20], [6]. Informally, this approach formalizes anomaly detection as a hypothesis testing problem. Under the null hypothesis H_0, it is *business as usual*, and the event counts for all nodes are generated proportionally to their baseline counts. Under the alternative hypothesis $H_1(S)$, counts of a majority of the vertices are generated (again) with rate proportional to the baseline counts, but there exists a small connected subset $S \subseteq V$ of vertices for which the counts are generated at a higher rate than expected. Then, the goal is to find a set of vertices S that maximizes an appropriate scan statistic function $F(S)$, typically a log-likelihood ratio that compares event counts to baseline counts. We define a scan statistic in terms of the event and baseline counts of a node set:

$$F(S) = F(W(S), B(S), \theta),$$

where $W(S) = \sum_{v \in S} w(v)$ is the total event count or *weight* of S, $B(S) = \sum_{v \in S} b(v)$ is the baseline count of the set, and θ represents possible additional arguments to F.

The graph anomaly detection problem can be posed as the following constrained optimization problem.

Problem 2: Given a graph $G = (V, E)$, a scan statistic $F(\cdot)$, the associated counts for vertices—$\mathbf{w}$ and $\mathbf{b}$—and a parameter k, find a connected subset $S \subseteq V$ that maximizes $F(S) = F(W(S), B(S), \theta)$ with $B(S) \leq k$.

Problem 2 is NP-hard, in general, as shown in [18]. We consider the following approximate version: find a subset S such that $B(S) \leq k$ and $F(S)$ equals the optimum, with probability at least $1 - \epsilon$, where $\epsilon \in (0, 1)$ is a parameter.

III. k-Multilinear Detection and Sequential Algorithms

We describe the sequential multilinear detection algorithm. This will start with some introduction to group algebras and Galois fields and end with the sequential algorithm for finding k-paths. Because of the limited space, we only describe the main ideas here and refer to [16], [17] for more details.

Let $X = x_1, \ldots, x_n$ be a set of variables, and let $P(X)$ be a polynomial, which is a sum of monomials on X. We will denote $P(X) = \sum_S \Pi_{i \in S} x_i$ as a monomial, where the sum is over multisets S. An example of a polynomial on six variables is $P(x_1, x_2, x_3, x_4, x_5, x_6) = x_1^2 x_2 + x_2 x_3 x_4 + x_3 x_4 x_5 + x_5 x_6$. A monomial is called *multilinear* or *square-free* if all its variables have exponent 1, and its *degree* is the sum of the exponents of all its variables. For instance, in the example above, $x_2 x_3 x_4$, $x_3 x_4 x_5$, and $x_5 x_6$ are multilinear monomials, but $x_1^2 x_2$ is not multilinear. Given variables $X = x_1, \ldots, x_n$ and a polynomial $P(X)$, the goal in the k-Multilinear Detection (k-MLD) problem is to decide whether or not $P(X)$ has a multilinear monomial of degree exactly k.

We note that the polynomial $P(X)$ may have an arbitrary number of terms—i.e., exponential on the size of n—therefore, the problem is not as simple as writing the polynomial explicitely and checking each term. Rather, we assume that $P(X)$ is given succintly in a recursive form, and the "yes"/"no" decision has to be made without unrolling this recursion. In general, we also have a weight w_S for each multinomial $\Pi_{i \in S} x_i$. Formally, we have the following problem.

Problem 3: (k-MLD problem) Given a polynomial $P(\cdot)$ defined recursively, in which each monomial has degree at most k and weight weight w_S, determine: (1) if $P(\cdot)$ has a multilinear term of degree k, and (2) the maximum weight of any multilinear term, if one exists.

A. Group Algebras

We discuss some notation from group algebras that is crucial for the paper. Let $\mathbb{Z}_2^k$ be the group formed by all the k-dimensional binary vectors, and define the group multiplication operation as entry-wise XOR. For example, $\mathbb{Z}_2^2$ consists of the vectors $v_0 = (0,0), v_1 = (0,1), v_2 = (1,0), v_3 = (1,1)$. We note that v_0 is the multiplicative identity of the group, and each element is its own multiplicative inverse: $v_i \cdot v_i = v_0$. Now, we define a group algebra $\mathbb{Z}_2[\mathbb{Z}_2^k]$. Each element in the group algebra is a sum of elements from $\mathbb{Z}_2^k$ with coefficients from $\mathbb{Z}_2$ (i.e., either 1 or 0): $\sum_{v \in \mathbb{Z}_2^k} a_v v$, where $a_v \in \{0,1\}$. The addition operator of the group algebra is

$$\sum_{v \in \mathbb{Z}_2^k} a_v v + \sum_{v \in \mathbb{Z}_2^k} b_v v = \sum_{v \in \mathbb{Z}_2^k} (a_v + b_v) v,$$

where the addition of the coefficients is modulo 2, and the multiplication is defined as

$$\left(\sum_{v \in \mathbb{Z}_2^k} a_v v \right) \left(\sum_{u \in \mathbb{Z}_2^k} b_u u \right) = \sum_{v \in \mathbb{Z}_2^k} (a_v \cdot b_u)(v \cdot u).$$

Example. For $k = 2$, the group algebra $\mathbb{Z}_2[\mathbb{Z}_2^k]$ has $2^{2^2} = 16$ elements, such as

$x_1 = 0 \cdot \begin{bmatrix}0\\0\end{bmatrix} + 1 \cdot \begin{bmatrix}0\\1\end{bmatrix} + 1 \cdot \begin{bmatrix}1\\0\end{bmatrix} + 0 \cdot \begin{bmatrix}1\\1\end{bmatrix}$, which we also write as $\begin{bmatrix}0\\1\end{bmatrix} + \begin{bmatrix}1\\0\end{bmatrix}$

$$x_2 = \begin{bmatrix}0\\0\end{bmatrix} + \begin{bmatrix}1\\0\end{bmatrix}$$

We have

$$x_1 + x_2 = \begin{bmatrix}0\\0\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix} + 2\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix}$$

$$x_1 x_2 = \left(\begin{bmatrix}0\\1\end{bmatrix} + \begin{bmatrix}1\\0\end{bmatrix} \right) \cdot \left(\begin{bmatrix}0\\0\end{bmatrix} + \begin{bmatrix}1\\0\end{bmatrix} \right) = \begin{bmatrix}0\\0\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix} + \begin{bmatrix}1\\0\end{bmatrix} + \begin{bmatrix}1\\1\end{bmatrix}$$

It is easy to check that

$$x_1^2 x_2 = 0 \cdot \begin{bmatrix}0\\0\end{bmatrix} + 0 \cdot \begin{bmatrix}0\\1\end{bmatrix} + 0 \cdot \begin{bmatrix}1\\0\end{bmatrix} + 0 \cdot \begin{bmatrix}1\\1\end{bmatrix} = \bar{0} \text{ (additive identify)}$$

B. Sequential algorithm for Multilinear Detection

We briefly discuss the algorithm of Koutis [16], which forms the basis of our paper. An important property is that for any $v_i \in \mathbb{Z}_2^k$, the square of the term $(v_0 + v_i) \in \mathbb{Z}_2[\mathbb{Z}_2^k]$ evaluates to 0:

$$(v_0 + v_i)^2 = v_0^2 + 2(v_0 \cdot v_i) + v_i^2 = v_0 + (0 \mod 2) v_i + v_0 = 2v_0 = 0.$$

We can show that, if we choose a $v_i \in \mathbb{Z}_2^k$ uniformly at random and set $x_i = v_0 + v_i$, then a multilinear monomial does **not** evaluate to $\bar{0}$ with high probability, whereas a monomial with squares is always $\bar{0}$ (as in the box above). The algorithm was later refined in [17] by evaluating the polynomial over the group algebra $GF(2^{3+\log_2 k})[\mathbb{Z}_2^k]$, where $GF(p)$ is the Galois field of order p [21], and this is the version that we implement. But, to simplify the discussion below, we assume that we are working on the group algebra $\mathbb{Z}_2[\mathbb{Z}_2^k]$.

A polynomial $P(x_1, \ldots, x_n)$ with variables from $\mathbb{Z}_2[\mathbb{Z}_2^k]$ can be evaluated in time $O(2^k poly(n))$ and space $O(2^k poly(n))$, resulting in Theorem 1.

Theorem 1 (Koutis [16] and Williams [17]) *There exists an algorithm that, given an instance $P(x_1, \ldots, x_n)$ of the k-*MLD *problem, correctly returns "no" if $P(X)$ does not contain a k multilinear term. Otherwise, if $P(X)$ has a k multilinear term, it returns "yes" with probability at least 1/5. The algorithm has time complexity $O(2^k poly(n))$ and space complexity $O(2^k poly(n))$.*

C. Implementation Using a Matrix Representation of $\mathbb{Z}_2[\mathbb{Z}_2^k]$

Theorem 1 performs operations in the group algebra $\mathbb{Z}_2[\mathbb{Z}_2^k]$, which takes $O(2^k poly(n))$ space. Koutis [16] showed that the space complexity can be reduced to $O(k poly(n))$ by using the idea of matrix representations. The main idea is that every element of the group algebra can be represented as a $2^k \times 2^k$ matrix, and the polynomial $P(X)$ evaluates to $\bar{0}$ if and only if the trace of its corresponding matrix representation is $0 \mod 2^{k+1}$. We can compute the trace by evaluating the polynomial over the group of all integers 2^k times, once for each element of the diagonal. For each variable $x_i = v_0 + v_i$, the tth diagonal element in the corresponding matrix representation is $1 + (-1)^{v^T t_{bin}}$, where t_{bin} is the k-bit binary representation of t.

D. Application to k-Path

As an example of the multilinear detection technique, we now describe a sequential algorithm for the k-Path problem, which is a special case of Problem 1. We are given a graph $G(V, E)$ and a parameter k, and the algorithm decides whether or not the graph has a path of length k. First, we reduce k-Path to a k-MLD instance (this follows from [16], [17]). Given a graph $G(V, E)$, let x_i denote a variable associated with each node $i \in V$. We define poynomials $P(i, j)$ for all $i \in V$, $j \leq k$ in the following manner.

- $P(i, 1) = x_i$ for all $i \in V$
- For $j > 1$, $P(i, j) = \sum_{u \in \text{NBR}(i)} P(i, 1) P(u, j-1)$
- Define the polynomial $P(x_1, \ldots, x_n) = \sum_i P(i, k)$

Intuitively, a polynomial $P(i, j)$ encodes all the possible walks of length j ending at node i. Each monomial in $P(i, j)$ corresponds to one walk. It can be verified that the graph G has a path of length k if and only if the polynomial $P(x_1, \ldots, x_n)$ has a multilinear term—i.e., a walk with no repeated vertices.

Algorithm 1 presents the full procedure. With the matrix representation, the polynomial for k-Path is evaluated over

2^k iterations (lines 6—12). In each iteration, we first initialize $P(i,1)$ (lines 7–8). From there, we compute recursively $P(i,j)$, a polynomial where each term contains x_i and has degree j (lines 9–11). The computation of $P(i,j)$ for a node i uses data from the immediate neighbors of i and all the polynomials of degree $j-1$, which have already been computed at this point. The two applications that we consider in Section V have this structure.

Algorithm 1 MULTILINEARDETECTPATH$(G(V,E),k)$.

1: **Input**: Graph $G(V,E)$ and parameter k
2: **Output**: "Yes" if G has a k-Path, "No" otherwise.
3:
4: For each node i, pick a random vector $v_i \in \mathbb{Z}_2^k$
5: $P = 0$
6: **for** $t = 0$ to 2^{k-1}
7: ▶ **Base case**
8: **for** $i \in V$ **do**
9: $P(i,1) = 1 + (-1)^{v_i^T \cdot t_{\text{bin}}}$
10: ▶ **Inductive step**
11: **for** $i \in V$, $j = 2$ to k **do**
12: $P(i,j) = \sum_{u \in \text{NBR}(i)} P(i,1)P(u,j-1)$
13: $P(k) = \sum_i P(i,k)$ for $i \in V$
14: $P = P + P(k) \mod 2^{k+1}$
15: **return** "Yes" if $P \neq 0$, else "No"

IV. PROPOSED PARALLEL ALGORITHM MIDAS FOR k-PATH

Opportunities and challenges for parallelization. Part of the outer for loop in lines 6–14 involves iterations which are uncoupled, in the sense that they can be done separately, as long as we are able to sum up the result P from each iteration, modulo 2^{k+1}. This gives us an easy source of parallelism, namely, run each iteration in parallel. However, this would not work if the graph does not fit in one processor's memory. The computation in the inductive step of Algorithm 1 has a *local* structure: a vertex only needs to data from its immediate neighbors in the graph. An alternative approach is to partition the graph into N parts, and then try to parallelize the local computation. However, neither extreme works well, and instead, MIDAS partitions the graph into N_1 parts, and runs N/N_1 iterations in parallel. This approach can lead to significant savings on the computation time, but has a high communication overhead. Since the values exchanged between nodes are small, we introduce an idea of combining the communications of multiple iterations together as a way to reduce the overhead.

A. Overview of Algorithm MIDAS

Let N denote the total number of processors or parallel units available. Quantities N_1 and N_2 are parameters for controlling the parallelism in different parts of the algorithm. We assume $2^k/N_2$ and N/N_1 are integers, in order to avoid cluttering the notation using ceiling and floor of these quantities. The algorithm involves solving a dynamic program 2^k times; these 2^k loops are independent, and we divide them into *phases* of size N_2 each, so that a total of $2^k/N_2$ phases have to be run. These are run in $2^k/(N_2N/N_1)$ *batches*, where each batch involves running N/N_1 phases. A phase involves a call to the subroutine PAREVALUATEPOLYNOMIALPATH. See Figure 1 for an illustration of this structure.

TABLE I: Summary of notation

Symbol	Description
N	Total number of processors
N_1	Number of parts in graph partitioning
N_2	Size of each phase
Phase	Group of N_2 iterations for which communication is done simultaneously
Batch	A set of N/N_1 phases
$\mathcal{P}$	Partition of G in N_1 parts, $G^1, \ldots, G^{N_1}$

We partition the graph G into N_1 parts, denoted by $\mathcal{P} = \{G^1, \ldots, G^{N_1}\}$; desirable properties of the partition will be discussed later. For a partition j, let $\text{DEG}(j)$ be the *degree* of j, defined as the number of edges connecting nodes in j to nodes in some other partition:

$$\text{DEG}(j) = |\{(u,v) : (u,v) \in G, u \in G^i, v \notin G^i\}|,$$

and let $\text{MAXDEG} = \max_j \text{DEG}(j)$. Also, let $\text{MAXLOAD} = \max_j |G^j|$ be the maximum "load" or number of vertices on any partition. We will analyze the performance of our algorithm in terms of MAXLOAD and MAXDEG.

We describe the main steps of MIDAS below.

1) The algorithm starts with the partitioning $\mathcal{P}$ of the graph G.
2) The algorithm runs $\log 1/\epsilon \log 5/4$ rounds, each of which involves 2^k iterations. Here, $\epsilon \in (0,1)$ is a parameter, which governs the success probability[1]. Each such round with 2^k iterations is partitioned into $2^k/N_2$ phases in the while loop in lines 8–12 of MIDAS, which are completely independent of other phases.
3) In the tth phase, Algorithm PAREVALUATEPOLYNOMIALPATH uses a vector of size N_2 to store polynomials $\langle P(i,tN_2,j), \ldots, P(i,(t+1)N_2-1,j)\rangle$ for each node i and $j \in [1,k]$. $P(i,q,j)$ corresponds to the polynomial of node i and degree j for the q^{th} diagonal element in the matrix representation.
4) In the tth phase, for each node i, we use the vector for each neighbor u of i to compute $\langle P(i,tN_2,j), \ldots, P(i,(t+1)N_2-1,j)\rangle$. If u is in the same partition, then its data is available on that processor. For every neighbor u in a different partition, u has to send a message with $\langle P(u,tN_2,j-1), \ldots, P(u,(t+1)N_2-1,j-1)\rangle$, introducing a communication overhead.
5) We use $\text{SUM}_t^\ell = \sum_{i \in V} P(i,tN_2,k) + \ldots + \sum_{i \in V} P(i,(t+1)N_2-1,k)$ to denote the sum of the polynomial evaluations for phase t, within round ℓ. These are summed up over all the phases within round ℓ to compute the total, denoted by P^ℓ.

[1] As per Theorem 1, Algorithm MULTILINEARDETECTPATH succeeds with probability 1/5, so we need to run it multiple times

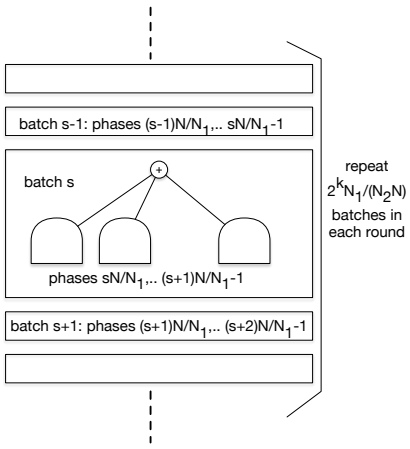

Fig. 1: Schematic structure of MIDAS: we run $(\log 1/\epsilon)/(\log 5/4)$ rounds. Each round is partitioned into $2^k N_1/(N_2 N)$ batches, and each batch involves N/N_1 phases being run simultaneously. Each phase involves an evaluation of the polynomial on N_2 iterations in algorithm PARMULTILINEARDETECTPOLYNOMIAL, which are then summed up.

Algorithm 2 MIDAS$(G, k, \epsilon, N_1, N_2)$.

1: **Input**: Graph $G = (V, E)$, parameter k, confidence parameter $\epsilon \in (0, 1)$, parameters N_1 and N_2, which guide the parallelism.
2: **Output**: "Yes" if G has a k-Path, "No" otherwise.
3:
4: Let $v_i \in \mathbb{Z}_2^k$ be a random vector for each node i
5: Let $P = 0$ be the polynomial
6: Let N_1 denote the number of processors used for each iteration. Let $\mathcal{P} = \{G^1, \ldots, G^{N_1}\}$ denote the corresponding partition of the graph into N_1 parts.
7: **for** $\ell = 1$ to $(\log 1/\epsilon)/(\log 5/4)$
8: $P^\ell = 0$
9: **while** $s \leq \frac{2^k/N_2}{N/N_1}$ **do**
10: **for** $t = sN/N_1$ to $(s+1)N/N_1$ **do in parallel**
11: $\text{SUM}_t^\ell = \text{PARMULTILINEARDETECTPOLYNOMIAL}(G, k, \mathbf{v}, t, N_2, N_1, \mathcal{P})$
12: MPIBARRIER
13: $P^\ell = P^\ell + \sum_{t=sN/N_1}^{(s+1)N/N_1} \text{SUM}_t^\ell \mod 2^{k+1}$ using MPIREDUCE
14: **if** $P^\ell \neq 0$ for some ℓ
15: **return** "Yes"
16: **else**
17: **return** "No"

B. Computation and Communication Complexity

Recall the definition of MAXDEG corresponding to the partitioning $\mathcal{P}$. Further, let c_1 and c_2 denote the time for unit computation at any node in G and the unit communication along any edge, respectively, in the Algorithm PARMULTILINEARDETECTPOLYNOMIALPATH. The time and communication complexity of algorithm MIDAS is summarized below in terms of these parameters.

Theorem 2: For any $\epsilon \in (0, 1)$, Algorithm MIDAS solves the k-PATH problem for an instance G, k with probability at least $1 - \epsilon$. The total time for computation and communication are $O\left(c_1 \frac{2^k N_1}{N} k \text{MAXLOAD} \log 1/\epsilon\right)$ and $O\left(c_2 \frac{2^k N_1}{N N_2} k \text{MAXDEG} \log 1/\epsilon\right)$, respectively.

Proof: (Sketch) First, we argue the correctness. The call to PARMULTILINEARDETECTPOLYNOMIALPATH evaluates the polynomial bottom up in parallel for all iterations in the tth phase, namely iterations $tN_2, \ldots, (t+1)N_2 - 1$. The vector $\langle P(tN_2, k), \ldots, P((t+1)N_2 - 1, k)\rangle$ is the final evaluation of the polynomial for each iteration in this phase. Each call to PARMULTILINEARDETECTPOLYNOMIALPATH in Algorithm MIDAS returns the sum of these values for phase t. Each round of Algorithm MIDAS, corresponding to the value of ℓ in the outer for loop in lines 6–12 in the sequential algorithm, goes over all phases, and calls PARMULTILINEARDETECTPOLYNOMIALPATH. Therefore, within round ℓ, P^ℓ denotes the sum of the polynomial evaluation over all the 2^k iterations within that round. If $P(\cdot)$ has a multilinear term, from [16], [17], $P^\ell \neq 0 \mod 2^{k+1}$ with probability at least $1/5$. This implies that $\Pr[P^\ell = 0,\ \forall \ell] = (\frac{4}{5})^{(\log 1/\epsilon)/(\log 5/4)} \leq \epsilon$, so that with probability at least $1 - \epsilon$, MIDAS returns "Yes" if G has a k-path. On the other hand, if $P(\cdot)$ has no multilinear term, then with probability 1, $P^\ell = 0$ for all ℓ. Therefore, MIDAS correctly solves the k-PATH problem with probability at least $1 - \epsilon$.

Next, we consider the computation and communication time complexity. The algorithm PARMULTILINEARDETECTPOLYNOMIALPATH computes the polynomial for each degree up to k within each iteration. Therefore, the computation time in a phase t is $O(c_1 k \max_j |G^j| N_2) = O(c_1 k \text{MAXLOAD} N_2)$, which is the maximum time for any processor. Therefore, the total compute time over all the rounds is $O(\frac{2^k/N_2}{N/N_1} c_1 k \text{MAXLOAD} N_2)$, which corresponds to the bound in the theorem. After the evaluation in the recursive step, the results have to be sent on all neighbors, for every pair of processors s, s'. Therefore, the maximum number of messages in one iteration of the loop in lines 8–15 is MAXDEG, and the total number of messages, over all the rounds, is $O(\frac{2^k/N_2}{N/N_1} \text{MAXDEG}) = O(\frac{2^k N_1}{N N_2} \text{MAXDEG})$.

Algorithm 3 PARMULTILINEARDETECTPOLYNOMIALPATH$(G, k, \mathbf{v}, t, N_2, N_1, \mathcal{P})$

1: **Input:** Graph G, parameter k, random assignment $\mathbf{v}$, phase number t, number of iterations within phase N_2, number of partitions N_1, and partitioning $\mathcal{P}$
2: **Output:** The value of the polynomial corresponding to k-path in the iterations within a phase
3:
4: **for** each processor s **do in parallel**
5: **Base case**
6: **for** node $i \in G^s$ and iteration $q \in [tN_2, (t+1)N_2 - 1]$ **do**
7: $P(i, q, 1) = 1 + (-1)^{v_i^T \cdot q_{\text{bin}}}$
8: **Recursive step**
9: **for** $j = 2$ to k **do**
10: **for** node $i \in G^s$ **do**
11: **for** all q set $P(i, q, j) = 0$
12: **for** each incoming message $\langle u, P(u, q, j-1)\rangle$ **do**
13: $P(i, q, j) = P(i, q, j) + P(i, q, 1)P(u, q, j-1)$
14: **Send result to neighbors**
15: **for** $u \in \text{NBR}(i) \setminus G^s$ **do**
16: **Send** $\langle i, P(i, q, j)\rangle$
17: MPIBARRIER
18: **return** $\sum_q \sum_i P(i, q, k)$

■

Memory Access: The recursive step in PAREVALUATEPOLYNOMIALPATH has some interesting properties of a highly memory bound region. Recall that polynomial multiplication terms are summed up for each of the incoming messages. This computation loop may be subjected locality effects of the memory sub-system and pipe-lining by the logical processor. Therefore, selecting appropriate values for N_2 is important to leverage fast memory access[2] and achieve desired parallel performance.

Lemma 1: For a graph $G = (V, E)$ drawn from the Erdös-Renyi model, $G(n, p)$, the computation and communication times for a random partition are $O(c_1 \frac{2^k nk}{N} \log 1/\epsilon)$ and $O(c_2 \log 1/\epsilon \frac{2^k mk}{NN_2})$, respectively, with high probability.

Proof: (Sketch) For a random partition into N_1 parts of equal size, we have $\text{MAXLOAD} = n/N_1$, and the bound for the total compute time follows from Theorem 2. Since $G \in G(n, p)$, it follows that $\text{MAXDEG} = O(\frac{n}{N_1}(n - \frac{n}{N_1}p)) = O(m/N_1)$, with high probability, and the Lemma follows. ■

V. PARALLEL ALGORITHMS FOR k-TREE AND NETWORK SCAN STATISTICS

We now describe how MIDAS for the k-path problem can be extended to parallel algorithms for finding trees and optimizing scan statistics. We discuss here how the corresponding polynomials are constructed recursively and evaluated in the subroutines PARTEVALUATEPOLYNOMIALTREE and PARTEVALUATEPOLYNOMIALSCANSTAT; the main Algorithm MIDAS remains unchanged. We recall the notation from Section II.

A. k-Tree

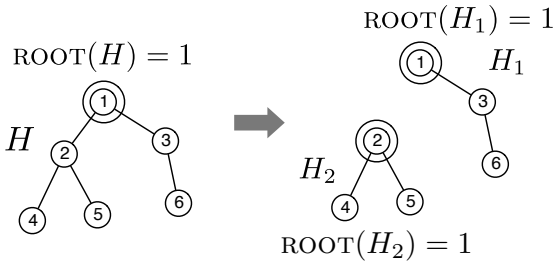

Fig. 2: Tree H with $\text{ROOT}(H) = 1$. It is decomposed into trees H_1 and H_2 by removing the edge $(1, 2)$. $\text{ROOT}(H_1) = 1$ and $\text{ROOT}(H_2) = 2$.

We describe how an instance of k-Tree with graph $G = (V, E)$ and tree $H = (V^H, E^H)$ is reduced to a k-MLD instance. We consider the tree H to be rooted, and let $\text{ROOT}(H)$ be the root node, selected arbitrarily. We consider a hierarchical structure among subtrees of H in the following manner: consider any node $u \in \text{NBR}(\text{ROOT}(H))$. Let H_1 and H_2 denote the subtrees or *children* obtained upon deleting the edge $(u, \text{ROOT}(H))$, with $\text{ROOT}(H) \in H_1$ and $u \in H_2$. We set $\text{ROOT}(H_1) = \text{ROOT}(H)$ and $\text{ROOT}(H_2) = u$. This process is illustrated in Figure 2. The subtrees H_1 and H_2 are further partitioned in a recursive manner until all trees have a single node. We define the polynomials $P(i, H')$, which will correspond to all layouts (not necessarily isomorphisms) of H' with $\text{ROOT}(H') = i$ for all nodes $i \in V$, in the following manner:

- If H' consists of a single node, $P(i, H') = x_i$
- Else, $P(i, H') = \sum_{u \in \text{NBR}(i)} P(i, H'_1)P(u, H'_2)$, where H'_1 and H'_2 are the children of H'.
- Finally, we have $P(x_1, \ldots, x_n) = \sum_{i \in V} P(i, H)$

By using ideas from [1], it can be verified that the tree H is a subgraph of G if and only if the polynomial $P(x_1, \ldots, x_n)$ has a multilinear term. Algorithm PARTEVALUATEPOLYNOMIALTREE evaluates this polynomial analogous to Algorithm 4 from Section IV. The performance of MIDAS, using PARTEVALUATEPOLYNOMIALTREE is summarized below.

Lemma 2: For any $\epsilon \in (0, 1)$, Algorithm MIDAS, using PARTEVALUATEPOLYNOMIALTREE, solves the k-TREE problem for an instance G, H with probability at least $1 - \epsilon$. The total time for computation and communication are $O\left(c_1 \frac{2^k N_1}{N} |\mathcal{T}| \text{MAXLOAD} \log 1/\epsilon\right)$ and $O\left(c_2 \frac{2^k N_1}{NN_2} |\mathcal{T}| \text{MAXDEG} \log 1/\epsilon\right)$, respectively.

Algorithm 4 PARTEVALUATEPOLYNOMIALTREE$(G(V, E), H(V^H, E^H), \mathbf{v}, t, N_2, N_1, \mathcal{P})$

Input: Graph $G(V, E)$, tree $H(V^H, E^H)$ with k vertices, random assignment $\mathbf{v}$, phase number t, number of iterations within phase N_2, number of partitions N_1, and partitioning $\mathcal{P}$
Output: The value of the polynomial corresponding to k-tree in the iterations within a phase

Let $\mathcal{T}$ be a collection of subtrees of H sorted by size
for each processor s**do in parallel**
  for each subtree $j \in \mathcal{T}$ **do**
    for node $i \in G^s$ and iteration $q \in [tN_2, (t+1)N_2 - 1]$ **do**
      if $|j| = 1$ **then**
        $P(i, q, j) = 1 + (-1)^{v_i^T \cdot q_{\text{bin}}}$
      else
        set $P(i, q, j) = 0$
        let j' and j'' be the children of subtree j
          for each incoming message $\langle u, P(u, q, j'')\rangle$ **do**
            $P(i, q, j) = P(i, q, j) + P(i, q, j')P(u, q, j'')$
      Send result to neighbors
      for $u \in \text{NBR}(i) \setminus G^s$ **do**
        Send $\langle i, P(i, q, j)\rangle$
MPIBARRIER
return $\sum_q \sum_i P(i, q, H)$

B. Scan Statistics

Let $W(V) = \sum_{i \in V} w(i)$ be the total weight of the nodes in G. For each node i, we define a variable x_i, and we construct a polynomial over the set of variables $\{x_i : i \in V\}$. Every term—i.e., monomial—in this polynomial will represent a connected subgraph of size at most k and weight at most $W(V)$. For $j \leq k$ and $z \leq W(V)$, let $P(i, j, z)$ be the polynomial corresponding to a subgraph (1) containing node i, (2) of size

[2]cache affinity in-terms of spatial and temporal locality results in fast memory access

j, and (3) total weight z. The following recurrence relations describe how the polynomials $P(i,j,z)$ are computed:

- $P(i,1,z) = x_i$ for all $i \in V$, $z = w(v)$
- For $i \in V$, $j = 2$ to k, $z = 0$ to $W(V)$, $P(i,j,z) = \sum_{u \in \text{NBR}(i)} \sum_{j'=1}^{j-1} \sum_{z'=0}^{z} (P(i,j',z') \cdot P(u, j-j', z-z'))$
- $P(j,z) = \sum_i P(i,j,z)$ for $j \leq k$, $z \leq W(V)$

Algorithm 5 maintains variables $P(i,q,j,z)$ for every node i, $j \leq k$, $z \leq W(V)$, and iteration q within phase t. It can be verified [19] that the input graph G has a connected subgraph S of size j and weight z if and only if the corresponding polynomial $P(j,z)$ has a multilinear term. We have the following lemma.

Lemma 3: For any $\epsilon \in (0,1)$, Algorithm MIDAS, using PAREVALUATEPOLYNOMIALSCANSTAT, solves the SCAN STATISTICS problem for an instance $G, k, \mathbf{w}$, with probability at least $1 - \epsilon$. The total time for computation and communication are $O\left(c_1 \frac{2^k N_1}{N} W(V)^2 k^2 \text{MAXLOAD} \log 1/\epsilon\right)$ and $O\left(c_2 \frac{2^k N_1}{N N_2} W(V)^2 k^2 \text{MAXDEG} \log 1/\epsilon\right)$, respectively.

We note that the performance for scan statistics can be improved significantly by rounding the weights, using a standard technique as in the Knapsack problem (see [19]).

Algorithm 5 PAREVALUATEPOLYNOMIALSCANSTAT$(G(V,E), k, \mathbf{w}, \mathbf{v}, t, N_2, N_1, \mathcal{P})$

1: **Input:** Graph $G(V,E)$, parameter k, node weights $\mathbf{w}$, random assignment $\mathbf{v}$, phase number t, number of iterations within phase N_2, number of partitions N_1, and partitioning $\mathcal{P}$
2: **Output:** The value of the polynomial corresponding to the scan statistics in the iterations within a phase
3:
4: **for** each processor s **do in parallel**
5: **for** node $i \in G^s$ and iteration $q \in [tN_2, (t+1)N_2 - 1]$ **do**
6: $P(i,q,1,w(v)) = 1 + (-1)^{v_i^T \cdot q_{\text{bin}}}$
7: **for** $j = 2$ to k and $z = 0$ to $W(V)$ **do**
8: **for** node $i \in G^s$ **do**
9: **for** all q set $P(i,q,j,z) = 0$
10: **for** each incoming message $\langle u, P(u,q,j-j',z-z') \rangle$ **do**
11: $P(i,q,j,z) = P(i,q,j,z) + P(i,q,j',z')P(u,q,j-j',z-z')$
12: **Send result to neighbors**
13: **for** $u \in \text{NBR}(i) \setminus G^s$ **do**
14: **Send** $\langle i, P(i,q,j,z) \rangle$
15: MPIBARRIER
16: **return** $\sum_q \sum_i P(i,q,k,z)$ for all $z \leq W(V)$

VI. EXPERIMENTS

The performance study of the proposed parallel algorithms covers the following topics.

- **Effect of partition size** investigates the performance variation with partition size as N_1 is varied (Section VI-B)
- **Scalability with subgraph size** depicts the total runtime as subgraph size is increased. (Section VI-C)
- **Strong scaling** looks at the total runtime as the number of parallel processors is increased, thereby reducing the computing workload per process. (Section VI-D)
- **MIDAS vs. FASCIA** presents the runtime comparison of our implementation compared to FASCIA. (Section VI-E)
- **Performance of Scan Statistics and its applications** provides performance results for the parallel Scan Statistics implementation and presents a real life application of it. (Section VI-F)

A. Experimental Setup

1) Hardware: Experiments were conducted on Juliet, an Intel Haswell HPC cluster. Up to 32 nodes were used for the evaluation, where each node has 36 cores (2 sockets x 18 cores each). A node consists of 128GB of main memory and 56Gbps Infiniband interconnect. We also tested on another HPC cluster, Shadowfax-Haswell, where we used 32 nodes each with 32 cores (2 sockets x 16 cores each). Memory and interconnect of this cluster are similar to those of Juliet.

2) Datasets: A summary of the datasets is provided in Table II. In addition, we perform experiments in two Erdos-Renyi networks of 1 and 10 million nodes with an expected number of edges of $n \log n$, where n is the number of nodes.

TABLE II: Datasets used in our experiments

Dataset	**Nodes** ($\times 10^6$)	**Edges** ($\times 10^6$)
miami	2.1	51.5
com-Orkut	3.1	234.3
random-1e6	1	13.8
random-1e7	10	161.8

B. Effect of Partition Size

The multilinear detection based parallel k-Path and Multilinear Scan algorithms exhibit two levels of parallelism: vertex and iterations. On one hand, the 2^k iterations are pleasingly parallel except for a global reduction at the end. The parallel vertex computation within each iteration, on the other hand, requires message passing between neighbors for $k-1$ steps (in the case of trees, this would be the number of sub templates instead of $k-1$).

Given these two levels of parallelism and a total of N processes, we can split the 2^k iterations among $a = N/N_1$ parallel phases. Each phase decomposes the graph across N_1 processes and performs the k-Path computation in parallel for $2^k/a$. To reduce the communication over computation cost, the algorithm packs a user defined N_2 number of iterations into one computation step, so each parallel phase only has to perform $2^k/(a * N_2)$ compute and communication phases.

To illustrate this with an example, consider the case of $k = 6$, $N = 128$, $N_1 = 32$, and $N_2 = 8$. The total number of iterations is $2^k = 64$. The number of parallel phases corresponding to $N_1 = 32$ is $128/32 = 4$. Each phase only needs to run $64/4 = 16$ iterations. Since $N_2 = 8$, the 16 iterations can be completed in just $16/8 = 2$ batches.

Increasing N_2, for example $N_2 = 16$ in the previous case, would allow us to finish the entire program in one compute and communicate batch. This results in higher parallel efficiency as the overhead of communication to computation is

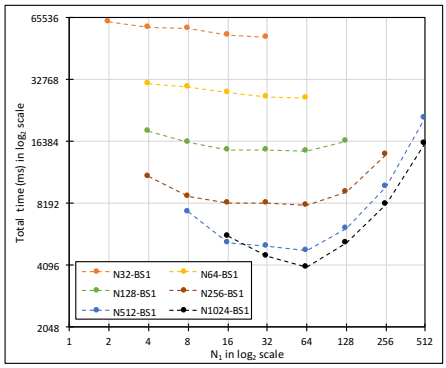

Fig. 3: k-path total runtime for random-1e6 dataset and varying N_1. Note. $BS1 = N_2 = 1$

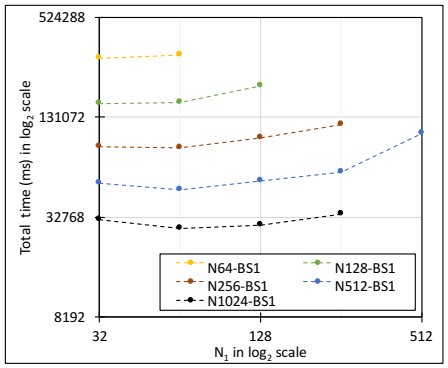

Fig. 4: k-path total runtime for com-Orkut dataset and varying N_1. Note. $BS1 = N_2 = 1$

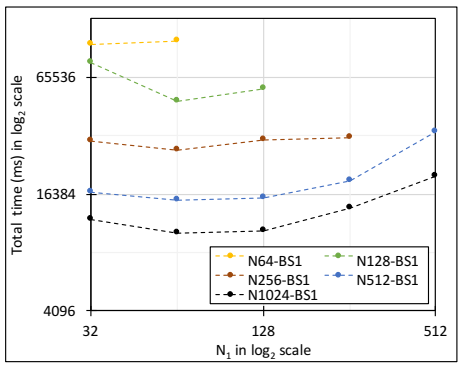

Fig. 5: k-path total runtime for miami dataset and varying N_1. Note. $BS1 = N_2 = 1$

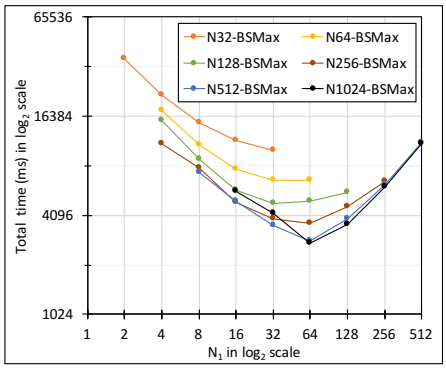

Fig. 6: k-path total runtime for random-1e6 dataset and varying N_1. Note. $BSMax = N_2 = 2^k N_1/N$

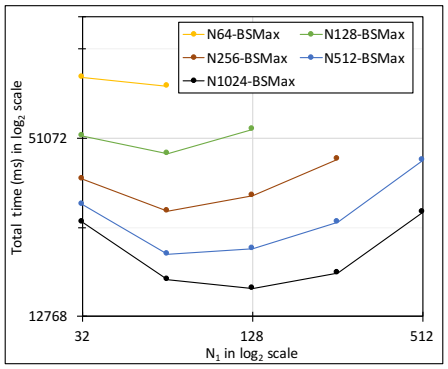

Fig. 7: k-path total runtime for com-Orkut dataset and varying N_1. Note. $BSMax = N_2 = 2^k N_1/N$

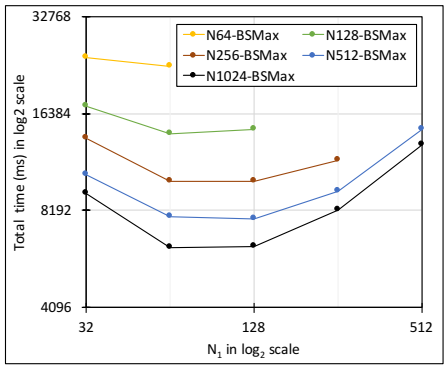

Fig. 8: k-path total runtime for miami dataset and varying N_1. Note. $BSMax = N_2 = 2^k N_1/N$

reduced. However, it increases the message size by N_2 factor[3]. Depending on the number of total processes, MPI may fail to accommodate very large message sizes requiring to reduce N_2 such that some form of chunking method may be required.

Figures 3, 4, 5 illustrate the performance of MIDAS on three different datasets when $N_2 = 1$. We observed that running times of MIDAS when N_2 is scaled for a fixed value of N for each problem size—this effectively tested our parallel algorithm for a large range of configurations. Our observations confirm the existence of an optimal point (i.e., a minimum) between two levels of parallelism discussed before. The communication cost gradually increases when moving from one extreme end of parallelism to the other because of the increase in number of messages exchanged [4]. However, at the optimal point, the cost of communication can be sufficiently amortized by the amount of parallelism gained. In other words, the optimal solution for MIDAS algorithm can be found in a point between vertex level and iteration based parallelism.

[3]Increasing message size for a communication step is not necessarily a bad occurrence. Reducing number of small messages in communication may lead to increased network performance. [22]

[4]Number of messages exchanged can be approximated to $O(\log N_1)$ for small message sizes where N_1 ranges from $N_1 = 1$ to $N_1 \to N$

Interestingly, when N_2 is increased (Figures 6, 7, 8) we observe further relative performance gains on the same set of experiments. The observed speedup (between ∼1x - ∼2x) is due to cache affinity effects on the main loop (Section IV-B) and reduction of communication phases by increasing the message size. Furthermore, speedups are evident for each experiment instance of the k-path problem when problem size is scaled.

C. Scalability with subgraph size

In Figure 11, we increase the subgraph size, k, in both FASCIA and MIDAS, while keeping N and N_1 fixed. Note, Figure 6 through Figure 8 suggest it is best to keep N_2 as high as possible to leverage the cache locality benefits discussed above. However, the total message size communicated out of a process increases with N_2 leading to diminishing returns. Therefore, we've kept $N_2 < 1024$.

D. MIDAS *Strong Scaling*

Strong scaling of MIDAS can be investigated in two ways. The first is to fix N_1 and change N, thereby increasing the parallel phases to split the 2^k iterations. We could observe the effect of this behavior by examining the values along a fixed N_1 value in Figures 3 through 8. Dividing the runtime

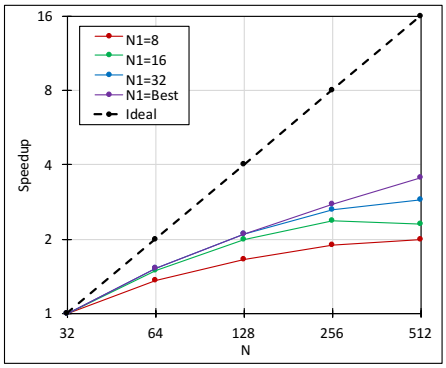

Fig. 9: MIDAS strong scaling for the k-path problem with increasing N, while N_1 is fixed.

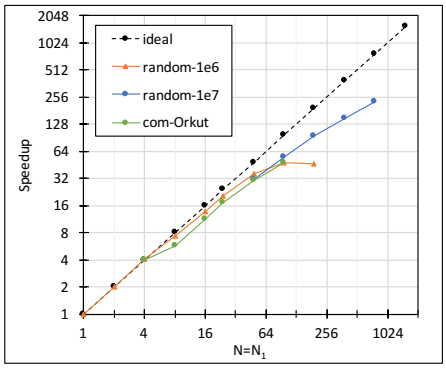

Fig. 10: MIDAS strong scaling for the k-path problem with increasing N and $N_1 = N$

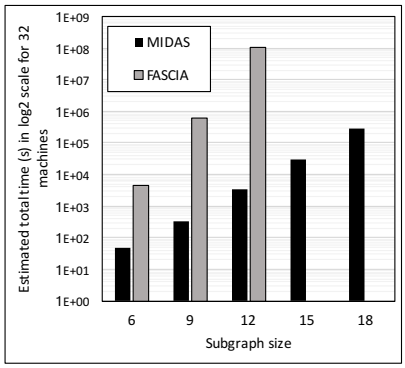

Fig. 11: MIDAS runtime compared to FASCIA for varying subgraph sizes of the k-path problem

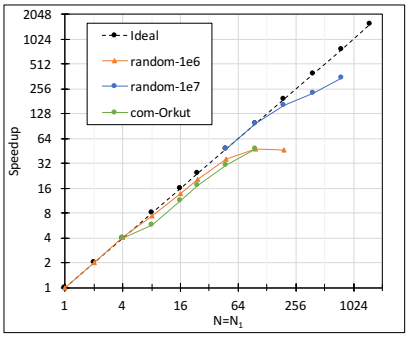

Fig. 12: MIDAS strong scaling for the Scan Statistics problem with increasing N and $N_1 = N$

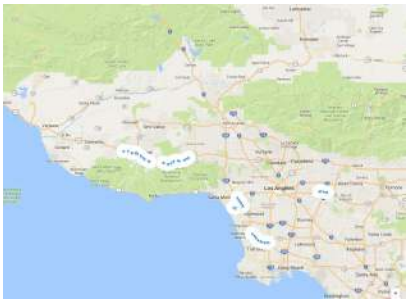

Fig. 13: Discovering highway segments with unexpected congesion in the Los Angeles road network.

corresponding to the minimum N (the top most line) by the given N gives speedup indicating the strong scalability of MIDAS.

Figure 9 presents such speedup for a set of N_1 values over varying N. We observe the results do not necessarily scale linearly due to the fact that communication within a phase is dominant. We get the best speedup by going along points in Figures 3 through 8 that gave the minimum runtime. This is shown as the $N1 = Best$ line in Figure 9.

The other form of strong scalability we can test is by setting $N1 = N$. This produces a single phase and is the classic strong scaling of parallel graph algorithms. Figure 10 presents the speedp values for different datasets. While, the speedups are less than ideal, they still scale well up to a considerable number of processes.

E. MIDAS *vs. FASCIA*

Figure 11 compares the running time of FASCIA to MIDAS for varying subgraph sizes. We see FASCIA fails to support beyond subgraphs of size 12 on this random-1e6 dataset, whereas MIDAS scales to well over 18. Also, MIDAS shows a significant improvement over FASCIA in runtime.

F. Performance of Scan Statistics and Its Applications

In Figure 12, we present strong scaling results for Scan Statistics problem where N_1 is set to N. We do this for multiple datasets and observe they show considerable strong scalability similar to k-Path problem in Figure 10.

Congested Highways Clusters in Road Networks We apply our algorithm for scan statistics to find clusters with unexpectedly low-moving traffic in the highway network of Los Angeles County[5]. Nodes in the graph are sensors next to the road that record the average speed and the number of vehicles passing through. We have 30-minute snapshots for May 2014. We assume that the average speed recorded by each sensor follows a normal distribution. Then, the p-value of a node i is the cumulative distribution function of a normal distribution with mean $\mu_i^{[1,t-1]}$ and standard deviation $\sigma_i^{[1,t-1]}$, where $\mu_i^{[1,t-1]}$ and $\sigma_i^{[1,t-1]}$ are, respectively, the sample mean and standard deviation for node i from snapshots 1 to $t-1$.

We use our algorithm with $k = 12$ on this dataset. In Figure 13, we show with blue dots highway segments that our algorithm identifies as having *unexpectedly* low average speed during rush hour (16:00 to 19:00) on Friday May 9, 2014. These segments are not necessarily the ones with most congestion. For instance, the center of Los Angeles city has higher congestion; however, such activity is normal on Friday afternoons according to the previous snapshots. The clusters

[5]http://pems.dot.ca.gov/

shown in the map are selected because they have significantly lower average speeds than in previous observations.

VII. Related Work

There is a huge literature on a variety of subgraph analysis problems, arising out of a number of applications, such as bioinformatics, security, social network analysis, epidemiology and finance (see [7] for a survey). We discuss some of three main directions here: subgraph isomorphism and clique enumeration (for which parallel algorithms exist), and anomaly detection (for which there has been limited work on parallel algorithms).

The basic frequent subgraph detection problem involves finding subgraphs having frequency higher than a threshold. Parallel approaches for this problem involve a "bottom-up" candidate generation approach, combined with careful pruning, which builds embeddings of larger subgraphs using all possible embeddings of smaller subgraphs[23], [24]. While these results allow scaling to very large networks with millions of nodes, they give no guarantees on the performance. Our work is more closely related to the use of the color coding technique for finding tree-like subgraphs [1], [2], which guarantees a fully polynomial time approximation to the number of embeddings with running time and space of $O(2^k e^k m \log n)$ and $O(2^k m)$, respectively. This has been parallelized using MapReduce [13] and OpenMP [14], [15], enabling subgraph counting in graphs with tens of millions of nodes with rigorous guarantees. Slota et al. [14], [15] use threading and techniques for reducing the memory footprint of the color coding dynamic programming tables, in order to scale.

Another area where parallel algorithms have been developed is for dense subgraph enumeration. There are several implementations for finding maximal cliques in parallel by careful partitioning, pruning and backtracking heuristics [9], [25], [10], [26], [11]. Our results do not extend to the clique enumeration problem.

Finally, there are only two prior works on parallel graph scan statistics [19], [25]. However, these do not scale to very large instances.

VIII. Conclusions

We present a new class of distributed MPI-based algorithms for various subgraph detection problems based on the recent multilinear detection technique. Even with a naive partitioning scheme, we observe significant performance improvement over the state-of-the-art color coding based methods. Our algorithms are conceptually much simpler than color coding and scale to subgraphs of size 18, which hasn't been done before. Our method combines parallelization of two different parts of the sequential multilinear algorithm, with a batched communication, and a data structure that gives cache locality.

References

[1] N. Alon, P. Dao, I. Hajirasouliha, F. Hormozdiari, and S. C. Sahinalp, "Biomolecular network motif counting and discovery by color coding," *Bioinformatics*, 2008.

[2] F. Hüffner, S. Wernicke, and T. Zichner, "Algorithm engineering for color-coding with applications to signaling pathway detection," *Algorithmica*, vol. 52, no. 2, pp. 114–132, 2008.

[3] S. Speakman *et al.*, "Scalable detection of anomalous patterns with connectivity constraints," *Jl Comp Graphical Stat*, 2015.

[4] M. Leiserson *et al.*, "Pan-cancer network analysis identifies combinations of rare somatic mutations across pathways and protein complexes," *Nature genetics*, vol. 47, no. 2, pp. 106–114, 2015.

[5] T. Hansen and F. Vandin, "Finding mutated subnetworks associated with survival in cancer," *arXiv preprint arXiv:1604.02467*, 2016.

[6] F. Chen and D. Neill, "Non-parametric scan statistics for event detection and forecasting in heterogeneous social media graphs," in *KDD*, 2014.

[7] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data Mining and Knowledge Discovery*, 2014.

[8] S. Arifuzzaman, M. Khan, and M. Marathe, "Patric: A parallel algorithm for counting triangles in massive networks," in *Proc. CIKM*, 2013.

[9] M. Schmidt, N. Samatova, K. Thomas, and B. Park, "A scalable, parallel algorithm for maximal clique enumeration," *Journal of Parallel and Distributed Computing*, vol. 69, no. 4, pp. 417–428, 2009.

[10] D. Aparicio, P. Ribeiro, and F. A. da Silva, "Parallel subgraph counting for multicore architectures," in *IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, 2014.

[11] N. Du, B. Wu, L. Xu, B. Wang, and P. Xin, "Parallel algorithm for enumerating maximal cliques in complex network," *Mining Complex Data*, pp. 207–221, 2009.

[12] N. Alon, R. Yuster, and U. Zwick, "Color-coding," *Journal of the ACM (JACM)*, 1995.

[13] Z. Zhao, G. Wang, A. R. Butt, M. Khan, V. A. Kumar, and M. V. Marathe, "Sahad: Subgraph analysis in massive networks using hadoop," in *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*. IEEE, 2012, pp. 390–401.

[14] G. M. Slota and K. Madduri, "Fast approximate subgraph counting and enumeration," in *Proc. ICPP*, 2013.

[15] ——, "Complex network analysis using parallel approximate motif counting," in *Proc. IPDPS*, 2014.

[16] I. Koutis, "Faster algebraic algorithms for path and packing problems," in *Proc. ICALP*, 2008.

[17] R. Williams, "Finding paths of length k in o(k2) time," *Information Processing Letters*, vol. 109, no. 6, pp. 315–318, 2009.

[18] J. Cadena, F. Chen, and A. Vullikanti, "Near-optimal and practical algorithms for graph scan statistics," in *SIAM Data Mining (SDM)*, 2017.

[19] S. Ekanayake, J. Cadena, and A. Vullikanti, "Fast graph scan statistics optimization using algebraic fingerprints," in *Proc. IEEE BigData*, 2017.

[20] J. Neil, C. Hash, A. Brugh, M. Fisk, and C. B. Storlie, "Scan statistics for the online detection of locally anomalous subgraphs," *Technometrics*, vol. 55, no. 4, pp. 403–414, 2013.

[21] G. Mullen and C. Mummert, "Finite fields and applications, volume 41 of student mathematical library," *American Mathematical Society*, vol. 3, pp. 19–20, 2007.

[22] M. J. Koop, T. Jones, and D. K. Panda, "Reducing connection memory requirements of mpi for infiniband clusters: A message coalescing approach," in *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*. IEEE, 2007, pp. 495–504.

[23] E. Abdelhamid, I. Abdelaziz, P. Kalnis, Z. Khayyat, and F. Jamour, "Scalemine: Scalable parallel frequent subgraph mining in a single large graph," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2016, p. 11.

[24] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis, "Grami: Frequent subgraph and pattern mining in a single large graph," in *Proc. International Conference on Very Large Data Bases*, 2014.

[25] J. Zhao, J. Li, B. Zhou, F. Chen, P. Tomchik, and W. Ju, "Parallel algorithms for anomalous subgraph detection," *Concurrency and Computation: Practice and Experience*, 2016.

[26] J. Cheng, L. Zhu, Y. Ke, and S. Chu, "Fast algorithms for maximal clique enumeration with limited memory," in *Proc. SIGKDD*, 2012.