Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

Proposal for Thesis Research in Partial Fulfillment
of the Requirements for the Doctoral Degree

Title: Bridging Theory and Practice in Parallel Clustering

Submitted by: Jessica Shi
32 Vassar St, G728
Cambridge, MA 02139

Signature of author: ______________________

Date of Submission: May 12, 2022

Expected Date of Completion: May 5, 2023

Laboratory: CSAIL

Thesis statement:

> Developing shared-memory parallel clustering algorithms with strong theoretical guarantees and using performance engineering techniques can lead to highly scalable, efficient, and cost-effective implementations on real-world datasets.

Bridging Theory and Practice in Parallel Clustering

Jessica Shi
Massachusetts Institute of Technology
jeshi@mit.edu

May 12, 2022

1 Introduction

A fundamental tool in modern data mining is graph and metric clustering, which has wide-ranging applications including in social network analysis [81], bioinformatics [106], and machine learning [100]. As the need to analyze larger and larger data sets increases, designing scalable algorithms that can handle billions of edges or points while maintaining fast speed and high quality becomes crucial. Current infrastructure for large-scale clustering in production settings relies not only on distributed algorithms with the ability to process trillion-size datasets [23, 202], but also on shared-memory parallel algorithms with the ability to process billion-size datasets [188]. Importantly, large distributed clusters are prohibitively expensive in many use-cases, while commodity multi-core machines are widely available to the average consumer, with much lower per-hour costs and generous machine sizes of up to 24 TB of memory [1]. Moreover, sequential algorithms do not take advantage of the multiple cores available and are prohibitively slow for datasets of these sizes. Thus, it is essential to develop efficient and performant shared-memory parallel algorithms and implementations that can scale to up to datasets of sizes in the hundreds of billions, from both a cost-savings and accessibility point of view.

Achieving parallel implementations that take full advantage of multiple cores is non-trivial. The problems that we study in this proposal are work-intensive, so highly parallel solutions that significantly increase the asymptotic work complexity lead to poor performance, especially for large datasets and where a limited number of cores is available. Moreover, the state-of-the-art sequential algorithms often have sequential dependencies that limit the amount of parallelism that can be naively introduced, particularly when scaling these solutions to increasing numbers of cores. Some of these problems are also known to be (or we show that they are) P-complete, indicating that polylogarithmic span solutions are highly unlikely, theoretically limiting the scalability of solutions to these problems. Especially in these scenarios, we must additionally take into account practical considerations in designing parallel implementations. Factors including cache performance and memory contention have significant effects on performance (for instance, see [52]), notably in large-scale multicore algorithms that involve frequent memory accesses and updates to shared locations.

We tackle these problems from two angles. First, we focus on developing algorithms with strong theoretical guarantees, which often translates to the most significant performance improvements in practice, over algorithms without theoretical bounds or with exponentially worse theoretical guarantees. By focusing specifically on work-efficient algorithms with low span, our implementations start from a strong theoretical grounding. We then improve our theoretically-efficient implementations with practical optimizations, using performance engineering techniques to achieve fast implementations on real-world datasets. We investigate parallel data structures and techniques that offer tradeoffs between performance and space-usage, as well as heuristics that optimize for cache-efficiency and minimize parallel overheads.

In this thesis, we study the following topics relating to fast and scalable parallel clustering:

1. the efficient discovery and enumeration of small subgraphs, which has applications in clustering metrics and graph statistics,

2. decomposition algorithms to discover hierarchical dense substructures based on higher order subgraphs, which form hierarchical clusters in themselves, but can also be used for clustering metrics and in preprocessing tasks, and

3. a generalized framework for graph and metric clustering, including an exploration of the interplay between metric datasets and graph building techniques.

In more detail, we have developed work-efficient parallel cycle and clique counting algorithms, and we have obtained experimental results demonstrating that these outperform previous state-of-the-art implementations [97,189,192]. We plan to generalize the techniques used in these algorithms to other types of small subgraphs, as well as subgraphs of larger sizes, for which some theoretical barriers exist [26]. Also, using our efficient subgraph counting algorithms, we have developed corresponding decomposition algorithms that hierarchically discover dense substructures based on subgraph motifs [189, 190, 192]. Our algorithms are work-efficient with low depth, and while we show that the exact problems are often P-complete, we provide approximation algorithms as well that have polylogarithmic depth. We plan to explore dynamic algorithms and other approximation algorithms for these problems.

Furthermore, we have developed an efficient shared-memory parallel correlation clustering implementation based on an exploration of several heuristic optimizations [188]. Our goal is to develop a generalized graph and metric clustering framework, to allow users to explore multiple state-of-the-art clustering algorithms in a single easy-to-use package. Moreover, metric and graph clustering techniques are often linked, and a metric clustering problem can be translated into a graph clustering problem through the use of graph building techniques on pointsets, such as k-nearest neighbors graphs [56]. We plan to explore these commonalities in our framework and offer a pipeline to translate seamlessly from metric clustering problems to graph clustering problems. Additionally, we plan to develop new parallel clustering implementations, including a high-dimensional k-means implementation.

Thesis statement: Developing shared-memory parallel clustering algorithms with strong theoretical guarantees and using performance engineering techniques can lead to highly scalable, efficient, and cost-effective implementations on real-world datasets.

2 Preliminaries

We consider graphs $G = (V, E)$ to be simple and undirected, and we let $n = |V|$ be the number of vertices and $m = |E|$ be the number of edges. For any vertex v, $\deg(v)$ denotes the degree of v. The *arboricity (α)* of a graph is the minimum number of spanning forests needed to cover the graph. α is upper bounded by $O\left(\sqrt{m}\right)$ and lower bounded by $\Omega\left(1\right)$ [40]. Closely related to arboricity is the *degeneracy (d)* of a graph G, or the smallest k such that every subgraph of G contains a vertex of degree at most k. It is known that $d = \Theta(\alpha)$ [151]. We note that having theoretical bounds parameterized by α, and as a result by d, is desirable since most real-world graphs have low arboricity [55]. A *c-orientation* of an undirected graph is a total ordering on the vertices, where the oriented out-degree of each vertex (the number of its neighbors higher than it in the ordering) is bounded by c.

For analysis, we use the work-span model [46, 103]. The *work* W of an algorithm is the total number of operations, and the *span* S is the longest dependency path. We can execute a parallel computation in $W/P + S$ running time using P processors [31]. We aim for *work-efficient* parallel algorithms in this model, that is, an algorithm with work complexity that asymptotically matches the best-known sequential time complexity for the problem. We assume concurrent reads and writes and atomic adds are supported in the model in $\mathrm{O}(1)$ work and span. We say $O(f(n))$ *with high probability* (*whp*) to indicate $O(cf(n))$ with probability at least $1 - n^{-c}$ for $c \geq 1$, where n is the input size.

Throughout our work, we use primitives from the Graph Based Benchmark Suite (GBBS) [58] and ParlayLib [29]. Notably, we use the efficient work-stealing scheduler from ParlayLib, which, as shown in [29], provides on average a 1.43x speedup over Intel's Parallel STL library.

3 Subgraph Counting and Listing

In the absence of ground-truth data, one of the key methods to determining the quality of graph clusterers is by investigating the makeup of small subgraphs within these clusters in relation to the overarching graph [25, 209, 212, 222]. Additionally, graphlet kernels and subgraph-based feature extraction for graphs are important steps in many machine learning pipelines for graph classification and community detection [88, 95, 173, 187]. Discovering specific subgraphs, such as k-cliques, can also contribute directly to community detection and graph partitioning tasks, as subroutines to more complex algorithms [25, 212]. Outside of clustering-related applications, subgraph counting in general has widespread applications in network analytics across various domains including bioinformatics and social network analysis [34, 71, 94]. We also note that cycle counting specifically has further important applications, including spam and fraud detection [24], and link classification and recommendation [211].

However, subgraph counting is a difficult problem particularly for subgraphs of larger sizes. The number of possible subgraphs grows exponentially with the subgraph size, so as such, discovering and counting these subgraphs naively can be computationally expensive. For instance, five-cycle counting is particularly computationally intensive and takes up between 25–58% of the total running time of the state-of-the-art serial Efficient Subgraph Counting Algorithmic PackagE (ESCAPE) [167] for general five-vertex subgraph counting. ESCAPE is unable to complete five-cycle counting on a graph with 200 million edges (com-orkut [127]) in 5.5 hours on an 18-core machine with 144 GiB of main memory. Additionally, for certain graphs, even state-of-the-art subgraph counting implementations for specific subgraphs, notably cliques, encounter memory limitations. Jain and Seshadhri's Pivoter [102], Danisch et al.'s kClist [48], Mhedhbi and Salihoglu's worst-case optimal join algorithm [148], and Lai *et al.*'s binary join algorithm [122] all run out of memory when counting four-cliques on graphs with billions to hundreds of billions of edges (ClueWeb [47], Hyperlink2014 [147], Hyperlink2012 [147]) on a machine with 80 cores and 3844 GiB of main memory.

We overcome these difficulties by exploiting common structural characteristics of real-world graphs. In particular, real-world graphs are often sparse and exhibit low arboricity. We have developed algorithms to efficiently obtain acyclic low out-degree orientations of these graphs, which we can then use to prune the search space of directed subgraphs [135, 189]. Even with the use of low out-degree orderings, though, we note that there are theoretical and practical barriers to efficiently counting large subgraphs in parallel in practice. We address these barriers in various ways. We explore the tradeoffs between space usage and performance in both our theoretical guarantees and our implementations, and we introduce approximation algorithms with unbiased estimators that offer much more significant speedups [189, 192]. We also explore different data structures and orientation algorithms, which may not be work-efficient but which offer better performance with lower parallel overheads in practice [97, 189, 192]. Our implementations are able to complete five-cycle counting on a graph with 200 million edges (com-orkut [127]) in under 3 minutes, and four-clique counting on graphs with billions to hundreds of billions of edges (ClueWeb [47], Hyperlink2014 [147], Hyperlink2012 [147]) in 2 hours, 4 hours, and 45 hours respectively, on the same corresponding machines as previously mentioned. Finally, maintaining subgraph counts on evolving or dynamic graphs poses an additional computational challenge, and we plan to explore this problem in our future work.

3.1 Related Work

There have been a wealth of related work on sequential and parallel subgraph counting, both for specific subgraphs or subgraph classes of interest, and for general subgraphs [171]. Additionally, certain algorithms focus on subgraph counting on graphs in specific graph classes, such as bipartite graphs and trees. Other variations include local subgraph counting per-vertex or per-edge. Many algorithms have been designed for finding 4- and 5-vertex subgraphs (e.g., [6, 159, 166, 167, 174, 220]) as well as estimating larger subgraph counts (e.g., [32, 33, 225]). Worst-case optimal join algorithms from the database literature [3, 122, 148, 158] have also been developed for small subgraph counting.

In the special case of k-cliques, a trivial algorithm enumerates k-cliques in $O(n^k)$ work, and using a thresholding argument improves the work for counting to $O(m^{c/2})$ [8]. The current fastest combinatorial algorithms for k-clique enumeration for sparse graphs are based on the seminal results of Chiba and Nishizeki [40], who show that all k-cliques can be enumerated in $\mathrm{O}(m\alpha^{k-2})$ work, where α is the arboricity of the graph. For arbitrary graphs, the fastest theoretical algorithm uses matrix multiplication, and counts $3l$ cliques in $\mathrm{O}(n^{l\omega})$ time where ω is the matrix multiplication exponent [153]. The k-clique counting problem is a canonical hard problem in the FPT literature, and is known to be $W[1]$-complete when parametrized by k [60]. We refer the reader to [213], which surveys other theoretical algorithms for this problem. The current state-of-the-art practical algorithms for k-clique counting are all based on the Chiba-Nishizeki algorithm [48, 130, 189]. The special case of counting and listing triangles ($k = 3$) has received a huge amount of attention over the past two decades (e.g., [12, 15, 91, 143, 161, 164, 165, 194, 205, 206, 210, 211, 232], among many others).

Additionally, in the special case of k-cycle counting, efficient algorithms have been developed to count k-cycles for $k \leq 5$. Notably, Alon et al. [8] developed algorithms for efficiently finding a k-cycle for general k, but these translate to efficient k-cycle counting algorithms only for planar graphs where $k \leq 5$. For $k = 4$, Chiba and Nishizeki [40] proposed algorithms that take $O(m\alpha)$ work, and Sanei-Mehri *et al.* [176] also introduce approximate counting algorithms based on sampling and graph sparsification. More recently, Bera *et al.* [26] analyzed the subgraph counting problem for $k = 5$ and gave an algorithm where the five-cycle counting subroutine takes $O(m\alpha^3)$ work. Kowalik [117] improves this with a five-cycle counting algorithm that takes $O(m\alpha^2)$ work. The current state-of-the-art practical algorithms for four-cycle counting are, as in k-clique counting, all based on the Chiba-Nishizeki algorithm [192, 217].

Moreover, there has been much recent work on the problem of maintaining subgraph counts on dynamic graphs. Eppstein and Spiro [66] present sequential dynamic algorithms for maintaining 3-vertex subgraph counts in amortized time proportional to the h-index of the graph, and Eppstein *et al.* [65] extend these results to 4-vertex subgraphs. Hanauer *et al.* [92] prove lower bounds on dynamic 4-vertex subgraph counting and give new efficient algorithms for certain 4-vertex subgraphs; they also show that Eppstein *et al.*'s [65] dynamic 4-vertex subgraph counting algorithm cannot be improved by a polynomial factor in the general case. Ammar *et al.* [10] obtain parallel dynamic worst-case optimal join algorithms for subgraph counting, but do not obtain improvements over their static worst-case complexity. Very recently, Chen *et al.* [38] introduce a practically efficient parallel dynamic subgraph enumeration algorithm that uses graph orientations to avoid double-counting, although their algorithm is not work-efficient. In the special case of triangle and k-clique counting, Kara *et al.* [113] and Dvorak and Tuma [62] respectively give sequential dynamic algorithms. In the parallel setting and specifically for triangle counting, Ediger *et al.* [63] and Makkar *et al.* [143] present batch-dynamic algorithms, but these take linear work per update in the worst case. Dhulipala *et al.* [57] develop theoretically efficient parallel batch-dynamic k-clique counting algorithms with fast and practical implementations.

3.2 Butterfly Counting

Triangles are core substructures in unipartite graphs, and indeed triangle counting is a core metric with widespread applications in areas including social network analysis [155], spam and fraud detection [24], and link classification and recommendation [211]. However, bipartite graphs do not contain triangles; instead, *butterflies*, also known as (2, 2)-bicliques, 4-cycles, or rectangles, are the smallest non-trivial dense subgraph in bipartite graphs.

In our work, we present a framework, ParButterfly, that provides different implementations for butterfly counting with strong theoretical guarantees [192]. The main procedure for butterfly counting involves finding *wedges*, or 2-paths, and combining them to count butterflies. In more detail, we find all wedges originating from each vertex, and then aggregate the counts of wedges incident to every distinct pair of vertices forming the endpoints of the wedge. The ParButterfly framework provides different ways to aggregate wedges in parallel, including sorting, hashing, histogramming, and batching. We further speed up the counting procedure by ranking vertices and only considering wedges formed by a particular ordering of the vertices. ParButterfly supports different parallel ranking methods, including side-ordering, approximate and exact degree-ordering, and approximate and exact complement-coreness ordering, all of which can be combined with any of the aggregation methods. Furthermore, we present parallel approximate butterfly counting algorithms and implementations within this framework, via graph sparsification based on ideas by Sanei-Mehri *et al.* [176] for the sequential setting. We additionally integrate a cache optimization for butterfly counting by Wang *et al.* [217].

We prove theoretical bounds showing that certain variants of our counting algorithms are work-efficient and take polylogarithmic span. Specifically, ParButterfly gives a counting algorithm that takes $O(\alpha m)$ expected work, $O(\log m)$ span *whp*, and $O(\min(n^2, \alpha m))$ additional space. Additionally, ParButterfly gives an approximate counting algorithm that takes $O((1 + \alpha' p)m)$ expected work, $O(\log m)$ span *whp*, and $O(\min(n^2, (1 + \alpha' p)m))$ space, where α' is the arboricity of the sparsified graph and p is the sampling probability. We use two sparsification methods, edge sparsification and colorful sparsification, both of which give unbiased estimates of the total butterfly count.

Moreover, we present a comprehensive experimental evaluation of all of the different variants of counting algorithms in ParButterfly, and we show on a 48-core machine with 2-way hyperthreading, our counting algorithms achieve self-relative speedups of up to 39x and outperform the previous fastest sequential baseline by up to 14x [176]. Compared to PGD [6], the previous state-of-the-art parallel subgraph counting solution that can count butterflies as a special case, ParButterfly is 350–5169x faster.

3.3 Five-cycle Counting

Cycle counting specifically is an important problem in fraud detection [169], and of the 5-vertex subgraphs, five-cycles are significantly more difficult to count because they are the only such pattern that requires first counting all directed three-paths. Indeed, the Efficient Subgraph Counting Algorithmic PackagE (ESCAPE), a software package by Pinar *et al.* that serially counts all 5-vertex subgraphs [167], spends between 25–58% of the total 5-vertex subgraph counting runtime on counting five-cycles alone based on our measurement. We present two new parallel five-cycle counting algorithms that not only have strong theoretical guarantees, but are also demonstrably fast in practice [97]. These algorithms are based on two different serial algorithms, namely by Kowalik [117] and from ESCAPE by Pinar *et al.* [167].

Kowalik studied k-cycle counting in graphs for $k \leq 6$ and proposed a five-cycle counting algorithm that runs in $O(md^2) = O(m\alpha^2)$ time for d-degenerate graphs [117], where α is the arboricity of the graph. Pinar *et al.*'s ESCAPE [167] contains contains a five-cycle counting algorithm that, with an important modification that we make, achieves the same asymptotic complexity of $O(m\alpha^2)$. The main procedure in both algorithms, and the essential modification to ESCAPE, is to first compute an appropriate arboricity orientation of the graph in parallel, where the vertices' out-degrees are upper-bounded by $O(\alpha)$. We use parallel orientation algorithms that we developed for parallel k-clique counting [189], which we discuss in more detail in Section 3.4. This orientation then enables the efficient counting of directed two-paths and three-paths, which are then appropriately aggregated to form five-cycles. Notably, the counting and aggregation steps can each be efficiently parallelized. The two algorithms differ fundamentally in the ways in which they use the orientations of these path substructures to eliminate double-counting.

We prove theoretical bounds that show that both of our algorithms match the work of the best sequential algorithms, taking $O(m\alpha^2)$ work and $O(\log^2 n)$ span *whp*. Additionally, we present two approximate five-cycle counting algorithms based on counting five-cycles in a sparsified graph, and we prove that both approximation algorithms give unbiased estimates on the global five-cycle count. We show that both algorithms take $O(pm\alpha^2 + m)$ expected work and $O(\log^2 n)$ span *whp* for a sampling probability p.

We present optimized implementations of our algorithms, which use thread-local data structures, fast resetting of arrays, and a new work scheduling strategy to improve load balancing. We provide a comprehensive experimental evaluation of our five-cycle counting algorithms. On a 36-core machine with 2-way hyperthreading, our best exact parallel algorithm achieves between 10–46x self-relative speedup, and between 162–818x speedups over the fastest prior serial five-cycle counting implementation, which is from ESCAPE [167]. We also implement our own serial versions of the two exact algorithms, which are 7–39x faster than ESCAPE's algorithm due to improved theoretical work complexities. Our best parallel algorithm achieves between 10–32x speedups over our best serial algorithm. Moreover, we show the tradeoffs between error and running time of our approximate five-cycle counting algorithms. In particular, we are able to approximate five-cycle counts with 12% error with a 9–189x speedup over exact five-cycle counting on the same graphs.

3.4 k-clique Counting and Listing

Finding k-cliques in a graph is a fundamental graph-theoretic problem with a long history of study in both theory and practice. In recent years, k-clique counting and listing have been widely applied in practice due to their many applications, including in learning network embeddings [173], understanding the structure and formation of networks [212, 227], identifying dense subgraphs for community detection [69, 87, 182, 209], and graph partitioning and compression [73].

We design a new parallel k-clique counting algorithm ARB-COUNT [189] that matches the work of Chiba-Nishezeki [40] (which is the best known sequential algorithm for k-clique counting for sparse graphs), has polylogarithmic span, and has improved space complexity compared to kClist [48], the state-of-the-art parallel k-clique counting algorithm. Our algorithm is able to significantly outperform kClist and other competitors, and scale to larger graphs than prior work. ARB-COUNT is based on using low out-degree orientations of the graph to reduce the total work. Assuming that we have a low out-degree ranking of the graph, we show that for a constant k we can count or list all k-cliques in $\mathrm{O}(m\alpha^{k-2})$ work, and $\mathrm{O}(k \log n + \log^2 n)$ span *whp* where α is the arboricity of the graph. Theoretically, ARB-COUNT requires $\mathrm{O}(\alpha)$ extra space per processor; in contrast, the kClist algorithm requires $\mathrm{O}(\alpha^2)$ extra space per processor. Furthermore, kClist does not achieve polylogarithmic span.

We also design an approximate k-clique counting algorithm based on counting on a sparsified graph. We show that our approximate algorithm produces unbiased estimates and runs in $\mathrm{O}(pm\alpha^{k-2} + m)$ work and $\mathrm{O}(k \log n + \log^2 n)$ span *whp* for a sampling probability of p.

In order to obtain the low out-degree orientations used in our k-clique counting algorithms, we present two new parallel algorithms for efficiently ranking the vertices. We show that a distributed algorithm by Barenboim and Elkin [22] can be implemented in linear work and polylogarithmic span. We also parallelize an external-memory algorithm by Goodrich and Pszona [86] and obtain the same complexity bounds.

We perform a thorough experimental study on a 30-core machine with 2-way hyperthreading and compare to prior work. We show that on a variety of real-world graphs and different k, our k-clique counting algorithm ARB-COUNT achieves 1.31–9.88x speedup over the state-of-the-art parallel kClist algorithm [48] and self-relative speedups of 13.23–38.99x. We also compared our k-clique counting algorithm to other parallel k-clique counting implementations including Jain and Seshadhri's Pivoter [102], Mhedhbi and Salihoglu's worst-case optimal join algorithm (WCO) [148], Lai *et al.*'s implementation of a binary join algorithm (BinaryJoin) [122], and Pinar *et al.*'s ESCAPE [167], and demonstrate speedups of up to several orders of magnitude.

Furthermore, by integrating state-of-the-art parallel graph compression techniques, we can process graphs with tens to hundreds of billions of edges, significantly improving on the capabilities of existing implementations. *As far as we know, we are the first to report* 4*-clique counts for Hyperlink2012, the largest publicly-available graph, with over two hundred billion undirected edges.*

We study the accuracy-time tradeoff of our sampling algorithm, and show that is able to approximate the clique counts with 5.05% error 5.32–6573.63 times more quickly than running our exact counting algorithm on the same graph. We compare our sampling algorithm to Bressan *et al.*'s serial MOTIVO [33], and demonstrate 92.71–177.29x speedups.

3.5 Future Work

We aim to incorporate our specialized subgraph counting algorithms into a more general framework for small subgraph counting. We plan to develop theoretically efficient parallel algorithms for general k-vertex subgraph counting where $k \leq 5$, based on the state-of-the-art serial algorithms [26,167]. In particular, our acyclic orientation algorithms [189] can be used to reduce the subgraph isomorphisms that we must discover for other classes of subgraphs, and for $k \leq 5$, we plan to incorporate these algorithms in a modularized fashion into a k-vertex subgraph counting framework. Other systems for general subgraph counting include AutoMine [146], Pangolin [39], and Peregrine [104, 105]. Most recently, Chen *et al.*'s Sandslash [38] is a practically efficient parallel implementation for subgraph enumeration that uses graph orientations to avoid double-counting, but does not have strong theoretical guarantees and is not work-efficient. We plan to improve upon their running times by exploring theoretically efficient algorithms, and combining these with additional practical optimizations.

Additionally, Bera *et al.* [26] showed that under the Triangle Detection Conjecture [2], there exists a constant $\gamma > 0$ such that for any $k \geq 6$ and any function $f : \mathbb{N} \to \mathbb{N}$, there is no expected $o(f(\alpha)m^{1+\gamma})$ algorithm for k-cycle counting. While practical implementations for six-cycle counting have been developed (standalone or as part of general subgraph counting systems) [3, 226], these results do not include strong theoretical guarantees. We plan to develop practically efficient k-cycle counting algorithms on sparse graphs for $k \geq 6$ with strong theoretical guarantees. A challenging barrier to this problem is efficiently aggregating and combining long directed paths, as well as capturing all isomorphisms of these k-cycles under acyclic orientations.

4 Subgraph Decomposition

The discovery of dense substructures, particularly through the use of higher order structures, is a fundamental topic in graph mining. In particular, understanding the distributions of and relationships between dense substructures has important applications in graph visualization tasks [9, 236], gene correlation and DNA motif detection in biological networks [79,230], motif detection in financial networks [61], and community detection in social networks and web graphs [68, 131]. Constructing hierarchies of dense substructures can form the basis of clustering pipelines, either as a preprocessing step or as a clusterer in itself [25, 83, 126, 212].

However, these dense substructure discovery algorithms often rely first on finding and listing certain subgraphs as a subroutine, and then maintaining these subgraphs through a decomposition algorithm in order to construct these hierarchies. This general process is computationally intensive, with many theoretical and practical barriers, due to P-completeness results and memory limitations, which we explore in our work [189, 190]. For instance, the previous state-of-the-art serial implementations for butterfly peeling [180] takes over 4 hours on a graph with 5.7 million edges (discogs_style [121]) on a machine with 48 cores and 384 GiB of main memory, and the previous state-of-the-art serial implementations for finding the k-clique densest subgraph [48,69] take over 5 hours for $k = 8$ on a graph with 200 million edges (com-orkut [127]) on a machine with 30 cores and 240 GiB of main memory. Moreover, for the same graph and on the same machine, the previous state-of-the-art parallel implementation for computing the $(3, 4)$-nucleus decomposition [181, 182] takes over an hour.

We use efficient data structures from GBBS [58] as well as our efficient subgraph counting subroutines in order to provide practical decomposition implementations [189,190,192]. We also present new efficient data structures, notably a new multi-level hash table structure to store information on cliques space-efficiently and a technique for traversing this structure cache-efficiently for the nucleus decomposition problem [190]. Moreover, we develop various approximation algorithms that take polylogarithmic span and are more scalable alternatives to the exact algorithms [189, 192]. Our implementations are able to perform butterfly peeling on a graph with 5.7 million edges (discogs_style [121]) in under 0.5 seconds on the previously mentioned machine with 48 core and 384 GiB of main memory. Also, we are able to compute the k-clique densest subgraph on a graph with 200 million edges (com-orkut [127]) in 2.5 hours, and perform $(3, 4)$-nucleus decomposition on the same graph in under 15 minutes, on the same machine as previously mentioned with 30 cores and 240 GiB of main memory.

4.1 Related Work

These subgraph decomposition problems are inspired by and closely related to the k-core problem, which was defined independently by Seidman [185], and by Matula and Beck [145]. The k-core of a graph is the maximal subgraph of the graph where the induced degree of every vertex is at least k. The *coreness* of a vertex is the maximum value of k such that the vertex participates in a k-core. Matula and Beck provided a linear time algorithm based on peeling vertices that computes the coreness value of all vertices [145].

Butterfly, or four-cycle, peeling is an algorithm for hierarchically discovering dense subgraphs in bipartite graphs, and was first developed by Zou [238] and Sariyüce and Pinar [180]. While these algorithms can be directly applied to unipartite graphs, in the special case of bipartite graphs, butterflies are (2,2)-bicliques and are the arguable equivalent to triangles, or three-cliques, in unipartite graphs; in this sense, butterflies are the smallest non-trivial subgraph that form the building blocks for dense subgraphs in bipartite graphs. In more detail, there are two variations of butterfly peeling, namely per vertex (tip decomposition) or per edge (wing decomposition), depending on whether vertices or edges are used respectively to define these dense subgraphs. Zou [238] introduces the latter, and Sariyüce and Pinar [180] introduce the former. Wang *et al.* [218] present a sequential algorithm for butterfly edge peeling that improves over the algorithm by Sariyüce and Pinar [180] in practice, and uses an index that takes $O(\alpha m)$ space. Additionally, Lakhotia *et al.* [123] develop an improved practically efficient parallel butterfly vertex peeling implementation that offers tradeoffs between work and scalability, and Wang *et al.* [219] develop a fast parallel butterfly edge peeling implementation that uses batch-based optimizations. A different approach to discovering dense substructures in bipartite graphs is the (α, β) decomposition, introduced by Liu *et al.* [134], which is a more direct generalization of k-core for bipartite graphs based on the degrees of vertices of each bipartition in the graph.

In general graphs, k-cliques are notable dense subgraphs of interest, and importantly, the k-clique densest subgraph problem, introduced by Tsourakakis [209], is a generalization of the densest subgraph problem. Tsourakakis presents a sequential $1/k$-approximation algorithm based on iteratively peeling the vertex with the minimum k-clique count, and a parallel $1/(k(1+\epsilon))$-approximation algorithm based on a parallel densest subgraph algorithm of Bahmani *et al.* [18]. Sun *et al.* [204] give additional approximation algorithms that converge to produce the exact solution over further iterations; these algorithms are more sophisticated and demonstrate the tradeoff between running times and relative errors. Recently, Fang *et al.* [69] propose algorithms for finding the largest (j, Ψ)-core of a graph, or the largest subgraph such that all vertices have at least j subgraphs Ψ incident on them. They propose an algorithm for Ψ being a k-clique that peels vertices with larger clique counts first and show that their algorithm gives a $1/k$-approximation to the k-clique densest subgraph.

Many other concepts capturing dense near-clique substructures have been proposed, including k-trusses (or triangle-cores), k-plexes [186], and n-clans and n-clubs [149]. In particular, k-trusses were proposed independently by Cohen [44], Zhang *et al.* [234], and Zhou *et al.* [236] with the goal of efficiently obtaining dense clique-like substructures. Unlike other near-clique substructures like k-plexes, n-clans, and n-clubs, which are computationally intractable to enumerate and count, k-trusses can be efficiently found in polynomial-time. Many parallel, external-memory, and distributed algorithms have been developed in the past decade for k-cores [54, 70, 109, 115, 150, 223] and k-trusses [28, 35, 37, 45, 110, 137, 195, 216, 238], and computing all trussness values of a graph is one of the challenge problems in the yearly MIT GraphChallenge [175]. Additional work has focused on how to maintain k-cores and k-trusses in dynamic graphs [7,11,96,98,107,108,132,133,140,141,178, 203,223,233,235]. The concept of a (r, s) nucleus decomposition was first proposed by Sariyüce *et al.* as a principled approach to discovering dense substructures in graphs that generalizes k-cores and k-trusses [182]. They also proposed an algorithm for efficiently finding the hierarchy associated with a (r, s) nucleus decomposition [179]. Sariyüce *et al.* later proposed parallel algorithms for nucleus decomposition based on local computation [181]. Recent work has studied nucleus decomposition in probabilistic graphs [67]. Very recently, Sariyüce proposed a motif-based decomposition, which generalizes the connection between r-cliques and s-cliques in nucleus decomposition to any pair of subgraphs [177].

4.2 Butterfly Peeling

As discussed in Section 3.2, butterflies, or $(2, 2)$-bicliques, are fundamental dense building blocks of bipartite graphs, and in particular, they naturally lend themselves to finding dense subgraph structures in bipartite networks. Zou [238] and Sariyüce and Pinar [180] developed peeling algorithms to hierarchically discover dense subgraphs. We present in our framework ParButterfly new parallel algorithms for butterfly peeling, with strong theoretical bounds [192]. Our peeling algorithms iteratively remove the vertices (tip decomposition) or edges (wing decomposition) with the lowest butterfly count until the graph is empty. Each iteration removes vertices (edges) from the graph in parallel and updates the butterfly counts of neighboring vertices (edges) using the parallel wedge aggregation techniques that we developed for counting, discussed in Section 3.2. We use a parallel bucketing data structure by Dhulipala *et al.* [54] and a new parallel Fibonacci heap to efficiently maintain the butterfly counts.

Our parallel Fibonacci heap improves upon the work bounds for vertex-peeling from Sariyüce and Pinar's sequential algorithms, which take work proportional to the maximum number of per-vertex butterflies. ParButterfly gives a vertex-peeling algorithm that takes $\mathrm{O}\big(\min(\text{max-b}_v, \rho_v \log n) + \sum_{v\in V} \deg(v)^2\big)$ expected work, $\mathrm{O}\big(\rho_v \log^2 n\big)$ span *whp*, and $\mathrm{O}\big(n^2 + \text{max-b}_v\big)$ additional space, and an edge-peeling algorithm that takes $\mathrm{O}\big(\min(\text{max-b}_e, \rho_e \log n) + \sum_{(u,v)\in E} \sum_{u'\in N(v)} \min(\deg(u), \deg(u'))\big)$ expected work, $\mathrm{O}\big(\rho_e \log^2 m\big)$ span *whp*, and $\mathrm{O}\big(m + \text{max-b}_e\big)$ additional space, where max-b$_v$ and max-b$_e$ are the maximum number of per-vertex and per-edge butterflies and ρ_v and ρ_e are the number of vertex and edge peeling iterations required to remove the entire graph. Additionally, given a slightly relaxed work bound, we can improve the space bounds in both algorithms; specifically, we have a vertex-peeling algorithm that takes $\mathrm{O}\big(\rho_v \log n + \sum_{v\in V} \deg(v)^2\big)$ expected work, $\mathrm{O}\big(\rho_v \log^2 n\big)$ span *whp*, and $\mathrm{O}\big(n^2\big)$ additional space, and we have an edge-peeling algorithm that takes $\mathrm{O}\big(\rho_e \log n + \sum_{(u,v)\in E} \sum_{u'\in N(v)} \min(\deg(u), \deg(u'))\big)$ expected work, $\mathrm{O}\big(\rho_e \log^2 m\big)$ span *whp*, and $O(m)$ additional space.

We present a comprehensive experimental evaluation of all of the different variants of butterfly peeling algorithms in the PARBUTTERFLY framework. On a 48-core machine with 2-way hyper-threading, our peeling algorithms achieve self-relative speedups of between 1.0–10.7x for vertex peeling and between 2.3–10.4x for edge peeling. Moreover, due to their improved work complexities, our peeling algorithms outperform the fastest sequential baseline [180] by between 1.3–30696x for vertex peeling and between 3.4–7.0x for edge peeling. Our speedups are highly variable because they depend heavily on the peeling complexities and the number of empty buckets processed.

4.3 k-clique Densest Subgraph

The k-clique densest subgraph problem is a generalization of the densest subgraph problem, which was first introduced by Tsourakakis [209], that captures higher-order k-clique structures in a graph in order to discover large near-cliques. This problem admits a natural $1/k$-approximation by peeling vertices in order of their incident k-clique counts. We present a new parallel peeling algorithm, ARB-PEEL, that peels all vertices with the lowest k-clique count on each round and uses ARB-COUNT as a subroutine [189]. The expected amortized work of ARB-PEEL is $O(m\alpha^{k-2} + \rho_k(G)\log n)$ and the span is $O(\rho_k(G)k\log n + \log^2 n)$ *whp*, where $\rho_k(G)$ is the number of rounds needed to completely peel the graph and α is the arboricity of the graph. We also prove that the problem of obtaining the hierarchy given by this process is P-complete for $k > 2$, indicating that a polylogarithmic-span solution is unlikely.

Tsourakakis also shows that naturally extending the Bahmani et al. [18] algorithm for approximate densest subgraph gives an $1/(k(1+\epsilon))$-approximation in $O(\log n)$ parallel rounds, although they were not concerned about work. We present an $O(m\alpha^{k-2})$ work and polylogarithmic-span algorithm, ARB-APPROX-PEEL, for obtaining a $1/(k(1+\epsilon))$-approximation to the k-clique densest subgraph problem. We obtain this work bound using our k-clique counting algorithm as a subroutine. Danisch *et al.* [48] use their k-clique counting algorithm as a subroutine to implement these two approximation algorithms for the k-clique densest subgraph as well, but their implementations do not have provably-efficient bounds.

We implement both ARB-PEEL and ARB-APPROX-PEEL with practical optimizations, and perform an experimental study on a 30-core machine with 2-way hyperthreading. We show that our parallel approximation algorithms for k-clique densest subgraph are able to outperform the parallel KCLIST [48] by up to 29.59x and achieve 1.19–13.76x self-relative speedup. We demonstrate up to 53.53x speedup over Fang *et al.*'s serial COREAPP [69] as well.

4.4 Nucleus Decomposition

Sariyüce *et al.* [182] introduced the nucleus decomposition problem, which generalizes the influential notions of k-cores and k-trusses to k-(r,s) nuclei, and can better capture higher-order structures in the graph. Informally, a k-(r,s) nucleus is the maximal induced subgraph such that every r-clique in the subgraph is contained in at least k s-cliques. The goal of the (r,s) nucleus decomposition problem is to identify for each r-clique in the graph, the largest k such that it is in a k-(r,s) nucleus.

Solving the (r,s) nucleus decomposition problem is a significant computational challenge for several reasons. First, simply counting and enumerating s-cliques is a challenging task, even for modest s. Second, storing information for all r-cliques can require a large amount of space, even for relatively small graphs. Third, engineering fast and high-performance solutions to this problem requires taking advantage of parallelism due to the computationally-intensive nature of listing cliques. There are two well-known parallel paradigms for approaching the (r, s) nucleus decomposition problem, a global peeling-based model and a local update model that iterates until convergence [181]. The former is inherently challenging to parallelize due to sequential dependencies and necessary synchronization steps [181], which we address in our work, and we demonstrate that the latter requires orders of magnitude more work to converge to the same solution and is thus less performant [190].

Lastly, it is unknown whether existing sequential and parallel algorithms for this problem are theoretically efficient. Notably, existing algorithms perform more work than the fastest theoretical algorithms for k-clique enumeration on sparse graphs [40, 189], and it is open whether one can solve the (r, s) nucleus decomposition problem in the same work as s-clique enumeration.

We address the computational challenges by designing a theoretically efficient parallel algorithm for (r, s) nucleus decomposition that nearly matches the work for s-clique enumeration, along with new techniques that improve the space and cache efficiency of our solutions. The key to our theoretical efficiency is a new combinatorial lemma bounding the total sum over all k-cliques in the graph of the minimum degree vertex in this clique, which enables us to to provide a strong upper bound on the overall work of our algorithm. As a byproduct, we also obtain the most theoretically-efficient serial algorithm for (r, s) nucleus decomposition. We provide several new optimizations for improving the practical efficiency of our algorithm, including a new multi-level hash table structure to space efficiently store data associated with cliques, a technique for efficiently traversing this structure in a cache-friendly manner, and methods for reducing contention and further reducing space usage. Finally, we experimentally study our parallel algorithm on various real-world graphs and (r, s) values, and find that it achieves between 3.31–40.14x self-relative speedup on a 30-core machine with 2-way hyperthreading. The only existing parallel algorithm for nucleus decomposition is by Sariyüce *et al.* [181], but their algorithm requires much more work than the best sequential algorithm. Our algorithm achieves between 1.04–54.96x speedup over the state-of-the-art parallel nucleus decomposition of Sariyüce *et al.*, and our algorithm can scale to larger (r, s) values, due to our improved theoretical efficiency and our proposed optimizations. We are able to compute the (r, s) nucleus decomposition for $r > 3$ and $s > 4$ on several million-scale graphs for the first time.

4.5 Future Work

We plan to develop parallel approximation algorithms for (r, s) nucleus decomposition, based on prior work on approximate k-clique peeling [189, 209] algorithms; importantly, we aim to obtain theoretically efficient approximation algorithms with polylogarithmic span, in order to significantly speed up the (r, s) nucleus decomposition implementations with relatively low percentage errors. Additionally, developing an approximate algorithm is an essential first step to obtaining an efficient dynamic (r, s) nucleus decomposition algorithm. Even in the special case of the k-core decomposition, the state-of-the-art prior work for maintaining the k-core decomposition in the sequential setting does not take polylogarithmic update time per single edge update. Indeed, a single edge update may cause all coreness values of all vertices to change; for instance, in the case of a cycle with one edge removed, repeatedly adding and removing this edge causes the coreness values of all vertices to repeatedly change. Thus, the goal is to extend upon our work on parallel batch-dynamic algorithms for approximate k-core [135] to obtain amortized polylogarithmic time bounds for approximate (r, s) nucleus decomposition, which hopefully translate to improved performance in practice over exact dynamic algorithms as well.

In both of these settings as well as the exact static setting, we also plan to develop algorithms to return not only the nuclei of each r-clique, but also the (r, s) nucleus decomposition hierarchy [179]. The (r, s) nucleus decomposition hierarchy encodes connectivity information, which can be retrieved from the nuclei, but is non-trivial to compute work-efficiently with low depth in parallel and is non-trivial to maintain dynamically. In particular, a naive algorithm would recompute connectivity across all r-cliques per nucleus number, which would not be work-efficient.

5 Parallel Clustering Framework and Algorithms

One of the main directions of our work is to develop a shared-memory parallel graph and metric clustering framework that is scalable and fast for graphs or pointsets with up to tens to hundreds of billions of edges or points respectively. While there exists a wealth of graph and metric clustering algorithms with different theoretical guarantees and use-cases [4,144,184], one fundamental question is how performant and how effective these algorithms are on different types of real-world datasets, especially with ground-truth clusters. We plan to offer in our framework scalable implementations of graph and metric clustering algorithms, and our overarching goal is to provide an easy-to-use API to compare various clustering algorithms and better understand the use-cases of each. In this spirit, we plan to also develop a pipeline for evaluating clustering algorithms, with classic statistics relating to clustering density and quality as well as comparisons to ground-truth data.

We additionally plan to introduce new highly efficient clustering algorithms as a part of this framework, and demonstrate their effectiveness in terms of speed and quality on real-world data. Different clustering algorithms offer different challenges in terms of effective parallelization and scalability, such as unavoidable sequential dependencies and objective functions that are NP-hard to optimize [59, 188]. In our work, we explore approximation guarantees and practical heuristics to approach these difficulties [56, 188]. Notably, we have developed a highly scalable parallel correlation clustering implementation that can cluster graphs with hundreds of millions of edges in under 10 minutes on a 30-core machine with 240 GiB of main memory. Our implementation heuristically relaxes convergence and consistency guarantees to achieve these speeds, but matches the precision and recall of its sequential counterpart on ground truth data [188]. Our plan is to develop a parallel high-dimensional k-means algorithm and implementation as well. We further note that our subgraph decomposition algorithms and implementations [189, 190, 192] also naturally output hierarchical clusters and communities of graphs, and as such can be viewed as clusterers in their own right. We plan to incorporate these works into our framework.

5.1 Related Work

Graph clustering is an expansive topic with many approaches, including modularity and correlation optimization [43, 156, 215], hierarchical clustering [116, 124, 154, 196], spectral clustering [157, 191, 224], partitional clustering [114,136,142], clique percolation [162], k-core decomposition [83], among others. Surveys of various algorithms can be found in Fortunato [78] and Fang *et al.* [68], and existing libraries with implementations for multiple community detection algorithms include the Stanford Network Analysis Project (SNAP) [129], NetworKit [197], and Neo4j [152]. Moreover, there exist a multitude of standalone scalable graph clustering implementations with demonstrably good quality on real-world datasets with various tradeoffs, including affinity clustering [23], Scalable Community Detection (SCD) [168], correlation clustering [188], and hierarchical agglomerative clustering [56, 202].

Moreover, once a clustering is obtained, there exists the important question of how to measure the quality of a clustering, either with or without ground-truth clusters available. There are a variety of well-known unsupervised quality metrics, including objective functions such as modularity [84] and conductance [111, 191], density metrics such as edge and clique density [85, 209], and the Dasgupta cost for hierarchical clusterings [51]. We note that many of these metrics can be trivially computed in parallel. Notably, Leskovec *et al.* [128] introduce the network community profile (NCP) plot, which captures the communities with the smallest conductances over different sizes, revealing the underlying structures of "good" clusters in these graphs. Shun *et al.*'s [193] work on fast parallel algorithms for local graph clustering allows for efficient generation of NCP plots.

In the specific example of correlation clustering, the LambdaCC framework by Veldt *et al.* [215] optimizes for a general objective that spans the classic modularity [84] and correlation clustering [21] objectives. Veldt *et al.* show that LambdaCC framework unifies several quality measures, including modularity, sparsest cut, cluster deletion, and a general version of correlation clustering. Optimization for correlation clustering has been studied empirically in the case of complete graphs, which is equivalent to LambdaCC objective with resolution $\gamma = 0.5$ [64, 163]. In this restricted setting, several scalable parallel implementations have been obtained based on the KwikCluster algorithm [41, 80, 163]. We observe that KwikCluster typically obtains a negative LambdaCC objective, which significantly limits its practical applicability. In the special case of the modularity objective, scalable modularity clustering has been extensively studied both in the shared-memory [72,76,90,138,197,229] and distributed memory [82,170,172,183] settings. The two fastest implementations that we identify are NetworKit [197] and Grappolo [82, 90]. Both of them offer comparable performance, but we observed the NetworKit typically computes solutions with slightly larger objective, and thus we compare to NetworKit in our empirical evaluation.

In the metric setting, k-means clustering is a well-known local search algorithm that partitions points into a fixed number of clusters by minimizing the sum of squared distances between each point and its cluster center [77, 136, 142, 198]. Theoretically, optimizing for the k-means objective is NP-hard, but there are a multitude of algorithms that achieve constant factor approximations [5,42,101, 112,125]. Notably, while k-means clustering can be exponential in the worst-case [13,214], Ostrovsky *et al.* [160] and Arthur and Vassilvitskii [14] independently showed that a careful selection of initial cluster centers leads to faster convergence in practice while maintaining theoretical guarantees on quality. Parallel k-means clustering is well-studied in a variety of practical settings, including in MapReduce and Message Passing Interface (MPI) models in distributed environments [27, 50, 53, 118, 119, 231, 237]. Other methods to speed up k-means clustering in parallel include using smaller coresets to represent larger datasets [16, 17, 20, 36, 74, 75, 93, 120, 139], reducing redundant distance computations [89, 119, 200], and intelligently initializing cluster centroids in parallel [19, 27].

5.2 Correlation Clustering

A major challenge in graph clustering is to design algorithms that can achieve fast speed at high scale while retaining high quality as evaluated on data sets with ground truth. Many graph clustering algorithms have been proposed to address this challenge, and our goal is to develop a state-of-the-art algorithm from both speed and quality perspectives. In particular, in our work [188], we adopt a new LambdaCC framework, introduced by Veldt *et al.* [215], which provides a general objective encompassing modularity [84] and correlation clustering [21]. Modularity is a widely-used objective that is formally defined as the fraction of edges within clusters minus the expected fraction of edges within clusters, assuming random distribution of edges. The goal of correlation clustering is to maximize agreements or minimize disagreements, where agreements and disagreements are defined based on edge weights indicating similarity and dissimilarity.

It is NP-hard to approximate modularity within a constant factor [59], so optimizing for modularity, and by extension optimizing for the LambdaCC objective, is inherently difficult. The most successful and widely-used modularity clustering implementations focus on heuristic algorithms, notably the popular Louvain method [30]. The Louvain method has been well-studied for use in modularity clustering, with highly optimized heuristics and parallelizations that allow them to scale to large real-world networks [138, 197, 207, 208].

We design, implement, and evaluate a generalized sequential and shared-memory parallel framework for Louvain-based algorithms including modularity and correlation clustering. We optimize the LambdaCC objective with state-of-the-art empirical performance, scaling to graphs with billions of edges. We also show that there is an inherent bottleneck to efficiently parallelizing the Louvain method, in that the problem of obtaining a clustering matching that given by the Louvain method on the LambdaCC objective, is P-complete. As such, we explore heuristic optimizations and relaxations of the Louvain method, and demonstrate their quality and performance trade-offs for the LambdaCC objective.

As part of our comprehensive empirical study, we show that our sequential implementation is orders of magnitude faster than the proof-of-concept implementation of Veldt *et al.* [215]. We note that for both LabmdaCC and correlation clustering objective, we are unaware of any existing implementation that would scale to even million-edge graphs and achieve comparable quality. We further show that our parallel implementations obtain up to 28.44x speedups over our sequential baselines on a 30-core machine with 2-way hyperthreading.

Moreover, we show that optimizing for the correlation clustering objective is of particular importance, by studying cluster quality with respect to ground truth data. We observe that optimizing for correlation clustering yields higher quality clusters than the ones obtained by optimizing for the celebrated modularity objective. In addition, we compare our implementation to two other prominent scalable algorithms for community detection: Tectonic [212] and SCD [168] and in both cases obtain favorable results, improving both the performance and quality. Finally, even in the highly competitive and extensively studied area of optimizing for modularity, we obtain an up to 3.5x speedup over a highly optimized parallel shared-memory modularity clustering implementation in NetworKit [197].

5.3 Future Work

Open-source clustering. We plan to develop a shared-memory parallel graph and metric clustering framework that demonstrates good quality on real-world data. Importantly, while there exist scalable graph clustering implementations with different theoretical guarantees and optimizing for different objectives [23, 56, 168, 188, 202], it is unclear how these implementations compare to one another and in which settings each implementation is preferable. We plan to benchmark these existing implementations, as well as incorporate our own fast implementations into an easy-to-use API with a streamlined parallel evaluation pipeline that can collect metrics and comparisons to ground-truth data. Graph clustering algorithms of interest include affinity clustering [23] and Scalable Community Detection (SCD) [168], correlation clustering and modularity clustering [188], hierarchical agglomerative clustering [56], clique percolation [189] and (r, s) nucleus decomposition [190], Tectonic [212], among others. Clustering metrics of interest include network community profile (NCP) plots [128], modularity [84], conductance [111, 191], edge and clique density [85, 209], Dasgupta cost and purity [51], Adjusted Rand Index (ARI) [99, 199] and Normalized Mutual Information (NMI) [49, 201], precision and recall, among others.

Moreover, we plan to explore metric clustering algorithms and implementations, including DBSCAN [221], hierarchical agglomerative clustering [228], k-means clustering [19], among others. We also plan to incorporate graph building techniques such as k-NN to allow us to explore the quality and performance of graph clustering algorithms on pointsets.

k-means clustering. We additionally plan to develop scalable parallel algorithms and implementations for high-dimensional k-means clustering, particularly for large k. While there is a wealth of prior work on parallel k-means clustering, approaches using coresets do not scale to large k [16, 17, 20, 36, 74, 75, 93, 120, 139] and techniques for fast centroid initialization do not explore other optimizations for speedup up the main Lloyd iterations [19, 27]. We plan to explore optimizations that reduce distance computations through use of graph building techniques and efficient locality-sensitive hashing (LSH).

6 Schedule

- Summer 2022 – Fall 2022: Finish work on developing parallel algorithms and implementations for a (r, s) nucleus decomposition hierarchy, approximate (r, s) nucleus decomposition, and dynamic (r, s) nucleus decomposition. Also, finish work on an open-source graph clustering framework, integrated with the Google graph clustering library and with benchmarking experiments. Give seminar talks on ongoing work at various venues.

- Fall 2022: Work with a SuperUROP student on finishing various (r, s) nucleus decomposition algorithms. Also, apply for jobs.

- Summer 2022 – Winter 2022: Finish work on scalable k-means clustering algorithms with Google.

- Winter 2022 – Spring 2023: Work on a parallel general subgraph counting framework with modularized graph orientation algorithms and practical optimizations. Look into developing theoretically efficient parallel algorithms for k-vertex subgraph counting where $k \geq 6$. Also, work on an open-source metric clustering framework, integrated with the Google metric clustering library and with benchmarking experiments.

- Spring 2023: Write and defend thesis.

References

[1] Amazon EC2 instance types. `https://aws.amazon.com/ec2/instance-types/`. Accessed: 2022-03-28.

[2] A. Abboud and V. V. Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 434–443, 2014.

[3] C. R. Aberger, A. Lamb, S. Tu, A. Nötzli, K. Olukotun, and C. Ré. EmptyHeaded: A relational engine for graph processing. *ACM Trans. Database Syst.*, 42(4):20:1–20:44, 2017.

[4] C. C. Aggarwal and H. Wang. *A Survey of Clustering Algorithms for Graph Data*, pages 275–301. Springer US, Boston, MA, 2010.

[5] S. Ahmadian, A. Norouzi-Fard, O. Svensson, and J. Ward. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 61–72, 2017.

[6] N. K. Ahmed, J. Neville, R. A. Rossi, N. G. Duffield, and T. L. Willke. Graphlet decomposition: framework, algorithms, and applications. *Knowl. Inf. Syst.*, 50(3):689–722, 2017.

[7] E. Akbas and P. Zhao. Truss-based community search: A truss-equivalence based indexing approach. *Proc. VLDB Endow.*, 10(11):1298–1309, Aug. 2017.

[8] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

[9] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. Large scale networks fingerprinting and visualization using the k-core decomposition. In *Proceedings of the 18th International Conference on Neural Information Processing Systems*, NIPS'05, page 41–50, Cambridge, MA, USA, 2005. MIT Press.

[10] K. Ammar, F. McSherry, S. Salihoglu, and M. Joglekar. Distributed evaluation of subgraph queries using worst-case optimal low-memory dataflows. *Proc. VLDB Endow.*, 11(6):691–704, Feb. 2018.

[11] S. Aridhi, M. Brugnara, A. Montresor, and Y. Velegrakis. Distributed k-core decomposition and maintenance in large dynamic graphs. In *ACM International Conference on Distributed and Event-Based Systems*, pages 161–168, 2016.

[12] S. Arifuzzaman, M. Khan, and M. Marathe. PATRIC: A parallel algorithm for counting triangles in massive networks. In *ACM Conference on Information and Knowledge Management (CIKM)*, pages 529–538, 2013.

[13] D. Arthur and S. Vassilvitskii. How slow is the k-means method? In *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry*, SCG '06, page 144–153, New York, NY, USA, 2006. Association for Computing Machinery.

[14] D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics.

[15] A. Azad, A. Buluç, and J. Gilbert. Parallel triangle counting and enumeration using matrix algebra. In *IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 804–811, 2015.

[16] O. Bachem, M. Lucic, and A. Krause. Scalable k -means clustering via lightweight coresets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 1119–1127, New York, NY, USA, 2018. Association for Computing Machinery.

[17] O. Bachem, M. Lucic, and S. Lattanzi. One-shot coresets: The case of k-clustering. 11 2017.

[18] B. Bahmani, R. Kumar, and S. Vassilvitskii. Densest subgraph in streaming and MapReduce. *Proc. VLDB Endow.*, 5(5), Jan. 2012.

[19] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii. Scalable k-means++. *Proc. VLDB Endow.*, 5(7):622–633, mar 2012.

[20] M. F. Balcan, S. Ehrlich, and Y. Liang. Distributed k-means and k-median clustering on general topologies. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 1995–2003, Red Hook, NY, USA, 2013. Curran Associates Inc.

[21] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Mach. Learn.*, 56(1–3):89–113, June 2004.

[22] L. Barenboim and M. Elkin. Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition. *Distributed Computing*, 22(5), 2010.

[23] M. Bateni, S. Behnezhad, M. Derakhshan, M. Hajiaghayi, R. Kiveris, S. Lattanzi, and V. Mirrokni. Affinity clustering: Hierarchical clustering at scale. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[24] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 16–24, 2008.

[25] A. R. Benson, D. F. Gleich, and J. Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016.

[26] S. K. Bera, N. Pashanasangi, and S. Comandur. Linear time subgraph counting, graph degeneracy, and the chasm at size six. *ArXiv*, abs/1911.05896, 2020.

[27] J. Bhimani, M. Leeser, and N. Mi. Accelerating k-means clustering with parallel implementations and gpu computing. In *2015 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, 2015.

[28] M. Blanco, T. M. Low, and K. Kim. Exploration of fine-grained parallelism for load balancing eager k-truss on gpu and cpu. In *IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, 2019.

[29] G. E. Blelloch, D. Anderson, and L. Dhulipala. Parlaylib - a toolkit for parallel algorithms on shared-memory multicore machines. In *ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA, page 507–509, New York, NY, USA, 2020. Association for Computing Machinery.

[30] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *J. Stat. Mech. Theory Exp.*, 2008(10):P10008, Oct. 2008.

[31] R. P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21(2):201–206, Apr. 1974.

[32] M. Bressan, F. Chierichetti, R. Kumar, S. Leucci, and A. Panconesi. Motif counting beyond five nodes. *TKDD*, 12(4):48:1–48:25, 2018.

[33] M. Bressan, S. Leucci, and A. Panconesi. Motivo: Fast motif counting via succinct color coding and adaptive sampling. *Proc. VLDB Endow.*, 12(11), July 2019.

[34] R. S. Burt. Structural holes and good ideas. *American Journal of Sociology*, 110(2):349–399, 2004.

[35] Y. Che, Z. Lai, S. Sun, Y. Wang, and Q. Luo. Accelerating truss decomposition on heterogeneous processors. *Proc. VLDB Endow.*, 13(10):1751–1764, June 2020.

[36] K. Chen. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM Journal on Computing*, 39(3):923–947, 2009.

[37] P.-L. Chen, C.-K. Chou, and M.-S. Chen. Distributed algorithms for k-truss decomposition. In *IEEE International Conference on Big Data (BigData)*, pages 471–480, 2014.

[38] X. Chen, R. Dathathri, G. Gill, L. Hoang, and K. Pingali. Sandslash: A two-level framework for efficient graph pattern mining. In *Proceedings of the ACM International Conference on Supercomputing*, ICS '21, page 378–391, New York, NY, USA, 2021. Association for Computing Machinery.

[39] X. Chen, R. Dathathri, G. Gill, and K. Pingali. Pangolin: An efficient and flexible graph mining system on cpu and gpu. *Proc. VLDB Endow.*, 13(8):1190–1205, apr 2020.

[40] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, Feb. 1985.

[41] F. Chierichetti, N. Dalvi, and R. Kumar. Correlation clustering in mapreduce. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD, page 641–650, New York, NY, USA, 2014. Association for Computing Machinery.

[42] D. Choo, C. Grunau, J. Portmann, and V. Rozhoň. K-means++: Few more steps yield constant approximation. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[43] A. Clauset, M. Newman, and C. Moore. Finding community structure in very large networks. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 70:066111, 01 2005.

[44] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*, 16(3.1), 2008.

[45] A. Conte, D. De Sensi, R. Grossi, A. Marino, and L. Versari. Discovering k-trusses in large-scale networks. In *IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–6, 2018.

[46] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.).* MIT Press, 2009.

[47] B. Croft and J. Callan. The Lemur project. `https://www.lemurproject.org/`, 2016.

[48] M. Danisch, O. Balalau, and M. Sozio. Listing k-cliques in sparse real-world graphs*. In *WWW*, 2018.

[49] L. Danon, J. Duch, A. Diaz-Guilera, and A. Arenas. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, 2005, 06 2005.

[50] A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: Scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, page 271–280, New York, NY, USA, 2007. Association for Computing Machinery.

[51] S. Dasgupta. A cost function for similarity-based hierarchical clustering. In D. Wichs and Y. Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 118–127. ACM, 2016.

[52] T. David, R. Guerraoui, and V. Trigonakis. Everything you always wanted to know about synchronization but were afraid to ask. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 33–48, 2013.

[53] I. S. Dhillon and D. S. Modha. A data-clustering algorithm on distributed memory multiprocessors. In M. J. Zaki and C.-T. Ho, editors, *Large-Scale Parallel Data Mining*, pages 245–260, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[54] L. Dhulipala, G. Blelloch, and J. Shun. Julienne: A framework for parallel graph algorithms using work-efficient bucketing. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 293–304, 2017.

[55] L. Dhulipala, G. E. Blelloch, and J. Shun. Theoretically efficient parallel graph algorithms can be fast and scalable. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 393–404, 2018.

[56] L. Dhulipala, D. Eisenstat, J. Łącki, V. Mirrokni, and J. Shi. Hierarchical agglomerative graph clustering in nearly-linear time. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2676–2686. PMLR, 18–24 Jul 2021.

[57] L. Dhulipala, Q. C. Liu, J. Shun, and S. Yu. Parallel batch-dynamic k-clique counting. In *APOCS*, 2021.

[58] L. Dhulipala, J. Shi, T. Tseng, G. E. Blelloch, and J. Shun. The graph based benchmark suite (GBBS). In *Joint Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, pages 11:1–11:8. ACM, 2020.

[59] T. N. Dinh, X. Li, and M. T. Thai. Network clustering via maximizing modularity: Approximation algorithms and theoretical limits. In *IEEE International Conference on Data Mining (ICDM)*, pages 101–110, 2015.

[60] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM J. Comput.*, 24(4), 1995.

[61] X. Du, R. Jin, L. Ding, V. E. Lee, and J. H. Thornton. Migration motif: A spatial - temporal pattern mining approach for financial markets. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, page 1135–1144, New York, NY, USA, 2009. Association for Computing Machinery.

[62] Z. Dvořák and V. Tůma. A dynamic data structure for counting subgraphs in sparse graphs. In F. Dehne, R. Solis-Oba, and J.-R. Sack, editors, *Algorithms and Data Structures*, pages 304–315, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[63] D. Ediger, K. Jiang, J. Riedy, and D. A. Bader. Massive streaming data analytics: A case study with clustering coefficients. In *Workshop on Multithreaded Architectures and Applications (MTAAP)*, pages 1–8, 2010.

[64] M. Elsner and W. Schudy. Bounding and comparing methods for correlation clustering beyond ilp. In *Workshop on Integer Linear Programming for Natural Langauge Processing*, pages 19–27, 2009.

[65] D. Eppstein, M. T. Goodrich, D. Strash, and L. Trott. Extended dynamic subgraph statistics using h-index parameterized data structures. *Theoretical Computer Science*, 447:44–52, 2012. Combinational Algorithms and Applications (COCOA 2010).

[66] D. Eppstein and E. S. Spiro. The h-index of a graph and its application to dynamic subgraph statistics. In F. Dehne, M. Gavrilova, J.-R. Sack, and C. D. Tóth, editors, *Algorithms and Data Structures*, pages 278–289, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[67] F. Esfahani, V. Srinivasan, A. Thomo, and K. Wu. Nucleus decomposition in probabilistic graphs: Hardness and algorithms. *arXiv preprint arXiv:2006.01958*, 2020.

[68] Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, and X. Lin. A survey of community search over big graphs. *The VLDB Journal*, 29(1):353–392, jan 2020.

[69] Y. Fang, K. Yu, R. Cheng, L. V. S. Lakshmanan, and X. Lin. Efficient algorithms for densest subgraph discovery. *Proc. VLDB Endow.*, 12(11), July 2019.

[70] M. Farach-Colton and M.-T. Tsai. Computing the degeneracy of large graphs. In *Latin American Symposium on Theoretical Informatics*, pages 250–260, 2014.

[71] K. Faust. A puzzle concerning triads in social networks: Graph constraints and the triad census. *Social Networks*, 32(3):221–233, 2010.

[72] M. Fazlali, E. Moradi, and H. Tabatabaee Malazi. Adaptive parallel louvain community detection on a multicore platform. *Microprocessors and Microsystems: Embedded Hardware Design (MICPRO)*, 54:26 – 34, 2017.

[73] T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. *Journal of Computer and System Sciences*, 51(2):261 – 272, 1995.

[74] D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, STOC '11, page 569–578, New York, NY, USA, 2011. Association for Computing Machinery.

[75] D. Feldman, M. Schmidt, and C. Sohler. Turning big data into tiny data: Constant-size coresets for k-means, pca and projective clustering. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, page 1434–1453, USA, 2013. Society for Industrial and Applied Mathematics.

[76] A. Fender, N. Emad, S. Petiton, and M. Naumov. Parallel modularity clustering. *Procedia Computer Science*, 108:1793–1802, 2017.

[77] E. W. Forgy. Cluster analysis of multivariate data : efficiency versus interpretability of classifications. *Biometrics*, 21:768–769, 1965.

[78] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[79] E. Fratkin, B. T. Naughton, D. L. Brutlag, and S. Batzoglou. MotifCut: Regulatory motifs finding with maximum density subgraphs. *Bioinformatics*, 22(14):e150–e157, 2006.

[80] D. García-Soriano, K. Kutzkov, F. Bonchi, and C. Tsourakakis. Query-efficient correlation clustering. In *The Web Conference*, WWW, page 1468–1478, New York, NY, USA, 2020. Association for Computing Machinery.

[81] U. Gargi, W. Lu, V. Mirrokni, and S. Yoon. Large-scale community detection on youtube for topic discovery and exploration. In *AAAI Conference on Weblogs and Social Media (ICWSM)*, volume 5, pages 486–489, 2011.

[82] S. Ghosh, M. Halappanavar, A. Tumeo, A. Kalyanaraman, H. Lu, D. Chavarrià-Miranda, A. Khan, and A. Gebremedhin. Distributed louvain algorithm for graph community detection. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 885–895, 2018.

[83] C. Giatsidis, F. Malliaros, D. Thilikos, and M. Vazirgiannis. Corecluster: A degeneracy based graph clustering framework. *Proceedings of the National Conference on Artificial Intelligence*, 1, 07 2014.

[84] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences (PNAS)*, 99(12):7821–7826, 2002.

[85] A. V. Goldberg. Finding a maximum density subgraph. Technical Report UCB/CSD-84-171, EECS Department, University of California, Berkeley, 1984.

[86] M. T. Goodrich and P. Pszona. External-memory network analysis algorithms for naturally sparse graphs. In *European Symposium on Algorithms*, 2011.

[87] E. Gregori, L. Lenzini, and S. Mainardi. Parallel k-clique community detection on large-scale networks. *IEEE Trans. Parallel Distrib. Syst.*, 24(8), Aug 2013.

[88] L. M. Gunderson and G. Bravo-Hermsdorff. Introducing graph cumulants: What is the variance of your social network? 2020.

[89] A. Gürsoy and I. Cengiz. Parallel pruning for k-means clustering on shared memory architectures. In *Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing*, Euro-Par '01, page 321–325, Berlin, Heidelberg, 2001. Springer-Verlag.

[90] M. Halappanavar, H. Lu, A. Kalyanaraman, and A. Tumeo. Scalable static and dynamic community detection using grappolo. In *IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2017.

[91] S. Han, L. Zou, and J. X. Yu. Speeding up set intersections in graph algorithms using simd instructions. In *ACM SIGMOD International Conference on Management of Data*, pages 1587–1602, 2018.

[92] K. Hanauer, M. Henzinger, and Q. C. Hua. Fully Dynamic Four-Vertex Subgraph Counting. *arXiv e-prints*, page arXiv:2106.15524, June 2021.

[93] S. Har-Peled and A. Kushal. Smaller coresets for k-median and k-means clustering. In *Proceedings of the Twenty-First Annual Symposium on Computational Geometry*, SCG '05, page 126–134, New York, NY, USA, 2005. Association for Computing Machinery.

[94] P. W. Holland and S. Leinhardt. A method for detecting structure in sociometric data. *American Journal of Sociology*, 76(3):492–513, 1970.

[95] T. Horváth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, page 158–167, New York, NY, USA, 2004. Association for Computing Machinery.

[96] Q. Hua, Y. Shi, D. Yu, H. Jin, J. Yu, Z. Cai, X. Cheng, and H. Chen. Faster parallel core maintenance algorithms in dynamic graphs. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 31(6):1287–1300, 2020.

[97] L. R. Huang, J. Shi, and J. Shun. Parallel Five-Cycle Counting Algorithms. In D. Coudert and E. Natale, editors, *19th International Symposium on Experimental Algorithms (SEA 2021)*, volume 190 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[98] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *ACM SIGMOD International Conference on Management of Data*, page 1311–1322, 2014.

[99] L. J. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2:193–218, 1985.

[100] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, Sept. 1999.

[101] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, mar 2001.

[102] S. Jain and C. Seshadhri. The power of pivoting for exact clique counting. In *ACM WSDM*, 2020.

[103] J. Jaja. *Introduction to Parallel Algorithms.* Addison-Wesley Professional, 1992.

[104] K. Jamshidi, R. Mahadasa, and K. Vora. Peregrine: A pattern-aware graph mining system. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery.

[105] K. Jamshidi and K. Vora. A deeper dive into pattern-aware subgraph exploration with peregrine. *SIGOPS Oper. Syst. Rev.*, 55(1):1–10, jun 2021.

[106] B. Jiang, J. Wang, J. Xiao, and Y.-C. Wang. Gene prioritization for type 2 diabetes in tissue-specific protein interaction networks. volume 11, 09 2009.

[107] W. Jiang, J. Qi, J. X. Yu, J. Huang, and R. Zhang. HyperX: A scalable hypergraph framework. volume 31, pages 909–922, May 2019.

[108] H. Jin, N. Wang, D. Yu, Q. Hua, X. Shi, and X. Xie. Core maintenance in dynamic graphs: A parallel approach based on matching. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 29(11):2416–2428, 2018.

[109] H. Kabir and K. Madduri. Parallel k-core decomposition on multicore platforms. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1482–1491, May 2017.

[110] H. Kabir and K. Madduri. Parallel k-truss decomposition on multicore systems. In *IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, 2017.

[111] R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad and spectral. *J. ACM*, 51(3):497–515, may 2004.

[112] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. A local search approximation algorithm for k-means clustering. *Computational Geometry*, 28(2):89–112, 2004. Special Issue on the 18th Annual Symposium on Computational Geometry - SoCG2002.

[113] A. Kara, H. Q. Ngo, M. Nikolic, D. Olteanu, and H. Zhang. Counting triangles under updates in worst-case optimal time. *CoRR*, abs/1804.02780, 2018.

[114] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[115] W. Khaouid, M. Barsky, V. Srinivasan, and A. Thomo. k-core decomposition of large networks on a single PC. *Proc. VLDB Endow.*, 9(1):13–23, 2015.

[116] B. King. Step-wise clustering procedures. *Journal of the American Statistical Association*, 69:86–101, 1967.

[117] Ł. Kowalik. Short cycles in planar graphs. In *Proceedings of the International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 284–296, 2003.

[118] P. Kraj, A. Sharma, N. Garge, R. Podolsky, and R. Mcindoe. Parakmeans: Implementation of a parallelized k-means algorithm suitable for general laboratory use. *BMC bioinformatics*, 9:200, 02 2008.

[119] J. Kumar, R. Mills, F. Hoffman, and W. Hargrove. Parallel k-means clustering for quantitative ecoregion delineation using large data sets. volume 4, 06 2011.

[120] S. Kumari, A. Maheshwari, P. Goyal, and N. Goyal. Parallel framework for efficient k-means clustering. In *Proceedings of the 8th Annual ACM India Conference*, Compute '15, page 63–71, New York, NY, USA, 2015. Association for Computing Machinery.

[121] J. Kunegis. KONECT: the Koblenz network collection. pages 1343–1350, 05 2013.

[122] L. Lai, Z. Qing, Z. Yang, X. Jin, Z. Lai, R. Wang, K. Hao, X. Lin, L. Qin, W. Zhang, Y. Zhang, Z. Qian, and J. Zhou. Distributed subgraph matching on timely dataflow. *Proc. VLDB Endow.*, 12(10), June 2019.

[123] K. Lakhotia, R. Kannan, V. Prasanna, and C. A. F. De Rose. Receipt: Refine coarse-grained independent tasks for parallel tip decomposition of bipartite graphs. *Proc. VLDB Endow.*, 14(3):404–417, nov 2020.

[124] G. N. Lance and W. T. Williams. A general theory of classificatory sorting strategies 1. Hierarchical systems. *Computer Journal*, 9(4):373–380, Feb. 1967.

[125] S. Lattanzi and C. Sohler. A better k-means++ algorithm via local search. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3662–3671. PMLR, 09–15 Jun 2019.

[126] V. E. Lee, N. Ruan, R. Jin, and C. Aggarwal. *A Survey of Algorithms for Dense Subgraph Discovery*, pages 303–336. Springer US, Boston, MA, 2010.

[127] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, 2019.

[128] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, page 695–704, New York, NY, USA, 2008. Association for Computing Machinery.

[129] J. Leskovec and R. Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1, 2016.

[130] R.-H. Li, S. Gao, L. Qin, G. Wang, W. Yang, and J. X. Yu. Ordering heuristics for k-clique listing. *Proc. VLDB Endow.*, 13(12):2536–2548, 2020.

[131] R.-H. Li, L. Qin, F. Ye, J. X. Yu, X. Xiao, N. Xiao, and Z. Zheng. Skyline community search in multi-valued networks. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 457–472, New York, NY, USA, 2018. Association for Computing Machinery.

[132] X. Li, D. G. Anderson, M. Kaminsky, and M. J. Freedman. Algorithmic improvements for fast concurrent cuckoo hashing. In *European Conference on Computer Systems (EuroSys)*, pages 27:1–27:14, 2014.

[133] Z. Lin, F. Zhang, X. Lin, W. Zhang, and Z. Tian. Hierarchical core maintenance on large dynamic graphs. *Proc. VLDB Endow.*, 14(5):757–770, 2021.

[134] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, and J. Zhou. Efficient (α, β)-core computation in bipartite graphs. *Proc. VLDB Endow.*, 29(5):1075–1099, 2020.

[135] Q. C. Liu, J. Shi, S. Yu, L. Dhulipala, and J. Shun. Parallel batch-dynamic k-core decomposition. *CoRR*, abs/2106.03824, 2021.

[136] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[137] T. M. Low, D. G. Spampinato, A. Kutuluru, U. Sridhar, D. T. Popovici, F. Franchetti, and S. McMillan. Linear algebraic formulation of edge-centric k-truss algorithms with adjacency matrices. In *IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, 2018.

[138] H. Lu, M. Halappanavar, and A. Kalyanaraman. Parallel heuristics for scalable community detection. *Parallel Computing*, 47:19 – 37, 2015. Special Issue: Graph Analysis for Scientific Discovery.

[139] M. Lucic, O. Bachem, and A. Krause. Strong coresets for hard and soft bregman clustering with applications to exponential family mixtures. In *AISTATS*, 2016.

[140] Q. Luo, D. Yu, X. Cheng, Z. Cai, J. Yu, and W. Lv. Batch processing for truss maintenance in large dynamic graphs. *IEEE Transactions on Computational Social Systems*, 7(6):1435–1446, 2020.

[141] Q. Luo, D. Yu, H. Sheng, J. Yu, and X. Cheng. Distributed algorithm for truss maintenance in dynamic graphs. In *Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pages 104–115, 2021.

[142] J. Macqueen. Some methods for classification and analysis of multivariate observations. In *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[143] D. Makkar, D. A. Bader, and O. Green. Exact and parallel triangle counting in dynamic graphs. In *IEEE International Conference on High Performance Computing (HiPC)*, pages 2–12, 2017.

[144] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval.* Cambridge University Press, USA, 2008.

[145] D. W. Matula and L. L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3), July 1983.

[146] D. Mawhirter and B. Wu. Automine: Harmonizing high-level abstraction and high performance for graph mining. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, page 509–523, New York, NY, USA, 2019. Association for Computing Machinery.

[147] R. Meusel, S. Vigna, O. Lehmberg, and C. Bizer. The graph structure in the web–analyzed on different aggregation levels. *The Journal of Web Science*, 1(1):33–47, 2015.

[148] A. Mhedhbi and S. Salihoglu. Optimizing subgraph queries by combining binary and worst-case optimal joins. *Proc. VLDB Endow.*, 12(11), July 2019.

[149] R. J. Mokken. Cliques, clubs and clans. *Quality & Quantity*, 13(2):161–173, 1979.

[150] A. Montresor, F. De Pellegrini, and D. Miorandi. Distributed k-core decomposition. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 24(2):288–300, 2012.

[151] C. Nash-Williams. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society*, s1-39(1):12–12, 1964.

[152] Neo4j. Neo4j. `https://neo4j.com/`, 2019.

[153] J. Nešetřil and S. Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 026(2), 1985.

[154] M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 69:026113, 03 2004.

[155] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.

[156] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Phys. Rev. E*, 69:066133, Jun 2004.

[157] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.

[158] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra. Worst-case optimal join algorithms. *J. ACM*, 65(3):16:1–16:40, Mar. 2018.

[159] M. Ortmann and U. Brandes. Efficient orbit-aware triad and quad census in directed and undirected graphs. *Applied Network Science*, 2, 06 2017.

[160] R. Ostrovsky, Y. Rabani, L. J. Schulman, and C. Swamy. The effectiveness of lloyd-type methods for the k-means problem. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 165–176, 2006.

[161] R. Pagh and C. E. Tsourakakis. Colorful triangle counting and a MapReduce implementation. *Inf. Process. Lett.*, 112(7):277–281, Mar. 2012.

[162] G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435:814–818, 07 2005.

[163] X. Pan, D. Papailiopoulos, S. Oymak, B. Recht, K. Ramchandran, and M. I. Jordan. Parallel correlation clustering on big graphs. In *International Conference on Neural Information Processing Systems*, volume 1 of *NIPS*, page 82–90, Cambridge, MA, USA, 2015. MIT Press.

[164] H.-M. Park and C.-W. Chung. An efficient MapReduce algorithm for counting triangles in a very large graph. In *ACM Conference on Information and Knowledge Management (CIKM)*, pages 539–548, 2013.

[165] H.-M. Park, F. Silvestri, U. Kang, and R. Pagh. MapReduce triangle enumeration with guarantees. In *ACM Conference on Information and Knowledge Management (CIKM)*, pages 1739–1748, 2014.

[166] H.-M. Park, F. Silvestri, R. Pagh, C.-W. Chung, S.-H. Myaeng, and U. Kang. Enumerating trillion subgraphs on distributed systems. *ACM Trans. Knowl. Discov. Data*, 12(6), Oct. 2018.

[167] A. Pinar, C. Seshadhri, and V. Vishal. ESCAPE: Efficiently counting all 5-vertex subgraphs. In *International Conference on World Wide Web (WWW)*, pages 1431–1440, 2017.

[168] A. Prat-Pérez, D. Dominguez-Sal, and J.-L. Larriba-Pey. High quality, scalable and parallel community detection for large real graphs. In *International Conference on World Wide Web*, number 23 in WWW, page 225–236, New York, NY, USA, 2014. Association for Computing Machinery.

[169] X. Qiu, W. Cen, Z. Qian, Y. Peng, Y. Zhang, X. Lin, and J. Zhou. Real-time constrained cycle detection in large dynamic graphs. *Proc. VLDB Endow.*, 11(12):1876–1888, Aug. 2018.

[170] X. Que, F. Checconi, F. Petrini, and J. A. Gunnels. Scalable community detection with the louvain algorithm. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 28–37, 2015.

[171] P. Ribeiro, P. Paredes, M. E. P. Silva, D. Aparicio, and F. Silva. A survey on subgraph counting: Concepts, algorithms, and applications to network motifs and graphlets. *ACM Comput. Surv.*, 54(2), mar 2021.

[172] J. Riedy, D. A. Bader, and H. Meyerhenke. Scalable multi-threaded community detection in social networks. In *IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPS)*, number 26, pages 1619–1628. IEEE, 2012.

[173] R. A. Rossi, N. K. Ahmed, and E. Koh. Higher-order network representation learning. In *Companion Proceedings of the The Web Conference 2018*, WWW '18, page 3–4, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee.

[174] R. A. Rossi, R. Zhou, and N. K. Ahmed. Estimation of graphlet counts in massive networks. *IEEE Trans. Neural Netw. Learning Syst.*, 30(1):44–57, 2019.

[175] S. Samsi, V. Gadepally, M. Hurley, M. Jones, E. Kao, S. Mohindra, P. Monticciolo, A. Reuther, S. Smith, W. Song, et al. Static graph challenge: Subgraph isomorphism. In *IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, 2017.

[176] S.-V. Sanei-Mehri, A. E. Sariyuce, and S. Tirthapura. Butterfly counting in bipartite networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2150–2159, 2018.

[177] A. E. Sariyüce. Motif-driven dense subgraph discovery in directed and labeled networks. In *The Web Conference (WWW)*, pages 379–390, 2021.

[178] A. E. Sarıyüce, B. Gedik, G. Jacques-Silva, K.-L. Wu, and Ü. V. Çatalyürek. Incremental k-core decomposition: Algorithms and evaluation. *Proc. VLDB Endow.*, 25(3):425–447, 2016.

[179] A. E. Sariyüce and A. Pinar. Fast hierarchy construction for dense subgraphs. *Proc. VLDB Endow.*, 10(3):97–108, Nov. 2016.

[180] A. E. Sariyüce and A. Pinar. Peeling bipartite networks for dense subgraph discovery. In *ACM International Conference on Web Search and Data Mining (WSDM)*, pages 504–512, 2018.

[181] A. E. Sariyüce, C. Seshadhri, and A. Pinar. Local algorithms for hierarchical dense subgraph discovery. *Proc. VLDB Endow.*, 12(1):43–56, 2018.

[182] A. E. Sariyüce, C. Seshadhri, A. Pinar, and U. V. Çatalyürek. Nucleus decompositions for identifying hierarchy of dense subgraphs. *ACM Trans. Web*, 11(3):16:1–16:27, July 2017.

[183] N. S. Sattar and S. Arifuzzaman. Parallelizing louvain algorithm: Distributed memory challenges. In *IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*, pages 695–701, 2018.

[184] S. E. Schaeffer. Survey: Graph clustering. *Comput. Sci. Rev.*, 1(1):27–64, Aug. 2007.

[185] S. B. Seidman. Network structure and minimum degree. *Soc. Networks*, 5(3):269 – 287, 1983.

[186] S. B. Seidman and B. L. Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical Sociology*, 6(1):139–154, 1978.

[187] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt. Efficient graphlet kernels for large graph comparison. In D. van Dyk and M. Welling, editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 488–495, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009. PMLR.

[188] J. Shi, L. Dhulipala, D. Eisenstat, J. Łăcki, and V. Mirrokni. Scalable community detection via parallel correlation clustering. *Proc. VLDB Endow.*, 14(11):2305–2313, jul 2021.

[189] J. Shi, L. Dhulipala, and J. Shun. Parallel clique counting and peeling algorithms. In *SIAM Conference on Applied and Computational Discrete Algorithms (ACDA)*, pages 135–146, 2021.

[190] J. Shi, L. Dhulipala, and J. Shun. Theoretically and practically efficient parallel nucleus decomposition. volume 15, page 583–596. VLDB Endowment, nov 2021.

[191] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[192] J. Shi and J. Shun. Parallel algorithms for butterfly computations. In *SIAM Symposium on Algorithmic Principles of Computer Systems (APoCS)*, pages 16–30, 2020.

[193] J. Shun, F. Roosta-Khorasani, K. Fountoulakis, and M. W. Mahoney. Parallel local graph clustering. *PVLDB*, 9(12):1041–1052, 2016.

[194] J. Shun and K. Tangwongsan. Multicore triangle computations without tuning. In *IEEE International Conference on Data Engineering (ICDE)*, pages 149–160, 2015.

[195] S. Smith, X. Liu, N. K. Ahmed, A. S. Tom, F. Petrini, and G. Karypis. Truss decomposition on shared-memory parallel systems. In *IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, 2017.

[196] P. H. Sneath and R. R. Sokal. *Numerical Taxonomy: The Principles and Practice of Numerical Classification.* W.H. Freeman, San Francisco, 1973.

[197] C. L. Staudt and H. Meyerhenke. Engineering parallel algorithms for community detection in massive networks. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 27(1):171–184, 2016.

[198] H. Steinhaus. Sur la division des corps matériels en parties. In *Bulletin L'Académie Polonaise des Science*, volume 4, pages 801–804, 1957.

[199] D. Steinley. Properties of the hubert-arabie adjusted rand index. *Psychological methods*, 9:386–96, 09 2004.

[200] K. Stoffel and A. Belkoniene. Parallel k/h-means clustering for large data sets. In *Proceedings of the 5th International Euro-Par Conference on Parallel Processing*, Euro-Par '99, page 1451–1454, Berlin, Heidelberg, 1999. Springer-Verlag.

[201] A. Strehl and J. Ghosh. Cluster ensembles - a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, 01 2002.

[202] B. Sumengen, A. Rajagopalan, G. Citovsky, D. Simcha, O. Bachem, P. Mitra, S. Blasiak, M. Liang, and S. Kumar. Scaling hierarchical agglomerative clustering to billion-sized datasets. *CoRR*, abs/2105.11653, 2021.

[203] B. Sun, T.-H. H. Chan, and M. Sozio. Fully dynamic approximate k-core decomposition in hypergraphs. *ACM Trans. Knowl. Discov. Data (TKDD)*, 14(4), May 2020.

[204] B. Sun, M. Danisch, T.-H. H. Chan, and M. Sozio. KClist++: A simple algorithm for finding k-clique densest subgraphs in large graphs. *Proc. VLDB Endow.*, 13(10), June 2020.

[205] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *International World Wide Web Conference (WWW)*, pages 607–614, 2011.

[206] K. Tangwongsan, A. Pavan, and S. Tirthapura. Parallel triangle counting in massive streaming graphs. In *ACM Conference on Information and Knowledge Management (CIKM)*, pages 781–786, 2013.

[207] V. A. Traag. Faster unfolding of communities: Speeding up the louvain algorithm. *Phys. Rev. E*, 92:032801, Sept. 2015.

[208] V. A. Traag, L. Waltman, and N. J. van Eck. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports*, 9(1):5233, Mar. 2019.

[209] C. Tsourakakis. The k-clique densest subgraph problem. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, page 1122–1132, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee.

[210] C. E. Tsourakakis. Counting triangles in real-world networks using projections. *Knowl. Inf. Syst.*, 26(3):501–520, 2011.

[211] C. E. Tsourakakis, P. Drineas, E. Michelakis, I. Koutis, and C. Faloutsos. Spectral counting of triangles via element-wise sparsification and triangle-based link recommendation. *Social Network Analysis and Mining*, 1(2):75–81, Apr 2011.

[212] C. E. Tsourakakis, J. Pachocki, and M. Mitzenmacher. Scalable motif-aware graph clustering. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 1451–1460, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.

[213] V. Vassilevska. Efficient algorithms for clique problems. *Inf. Process. Lett.*, 109(4), 2009.

[214] A. Vattani. K-means requires exponentially many iterations even in the plane. In *Proceedings of the Twenty-Fifth Annual Symposium on Computational Geometry*, SCG '09, page 324–332, New York, NY, USA, 2009. Association for Computing Machinery.

[215] N. Veldt, D. F. Gleich, and A. Wirth. A correlation clustering framework for community detection. In *World Wide Web Conference*, WWW, page 439–448, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee.

[216] J. Wang and J. Cheng. Truss decomposition in massive networks. *Proc. VLDB Endow.*, 5(9):812–823, May 2012.

[217] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang. Vertex priority based butterfly counting for large-scale bipartite networks. *PVLDB*, 12(10), June 2019.

[218] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang. Efficient bitruss decomposition for large-scale bipartite graphs. In *IEEE International Conference on Data Engineering (ICDE)*, 2020.

[219] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang. Towards efficient solutions of bitruss decomposition for large-scale bipartite graphs. *The VLDB Journal*, 31(2):203–226, mar 2022.

[220] P. Wang, J. Zhao, X. Zhang, Z. Li, J. Cheng, J. C. S. Lui, D. Towsley, J. Tao, and X. Guan. MOSS-5: A fast method of approximating counts of 5-node graphlets in large graphs. *IEEE Transactions on Knowledge and Data Engineering*, 30(1):73–86, Jan 2018.

[221] Y. Wang, Y. Gu, and J. Shun. Theoretically-efficient and practical parallel dbscan. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 2555–2571, New York, NY, USA, 2020. Association for Computing Machinery.

[222] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

[223] D. Wen, L. Qin, Y. Zhang, X. Lin, and J. Yu. I/O efficient core graph decomposition: Application to degeneracy ordering. *IEEE Transactions on Knowledge & Data Engineering (TKDE)*, 31(01):75–90, jan 2019.

[224] S. White and P. Smyth. *A Spectral Clustering Approach To Finding Communities in Graphs*, pages 274–285.

[225] C. Yang, M. Lyu, Y. Li, Q. Zhao, and Y. Xu. Ssrw: A scalable algorithm for estimating graphlet statistics based on random walk. In *Database Systems for Advanced Applications: 23rd International Conference, DASFAA 2018, Gold Coast, QLD, Australia, May 21-24, 2018, Proceedings, Part I*, page 272–288, Berlin, Heidelberg, 2018. Springer-Verlag.

[226] Y. Yang, Y. Fang, M. E. Orlowska, W. Zhang, and X. Lin. Efficient bi-triangle counting for large bipartite networks. *Proc. VLDB Endow.*, 14(6):984–996, feb 2021.

[227] H. Yin, A. R. Benson, and J. Leskovec. Higher-order clustering in networks. *Physical Review E*, 97(5), 2018.

[228] S. Yu, Y. Wang, Y. Gu, L. Dhulipala, and J. Shun. Parchain: A framework for parallel hierarchical agglomerative clustering using nearest-neighbor chain. *Proc. VLDB Endow.*, 15(2):285–298, oct 2021.

[229] J. Zeng and H. Yu. Parallel modularity-based community detection on large-scale graphs. In *IEEE International Conference on Cluster Computing*, pages 1–10, 2015.

[230] B. Zhang and S. Horvath. A general framework for weighted gene co-expression network analysis. *Statistical Applications in Genetics and Molecular Biology*, 4(1), 2005.

[231] J. Zhang, G. Wu, X. Hu, S. Li, and S. Hao. A parallel k-means clustering algorithm with mpi. In *2011 Fourth International Symposium on Parallel Architectures, Algorithms and Programming*, pages 60–64, 2011.

[232] Y. Zhang, H. Jiang, F. Wang, Y. Hua, D. Feng, and X. Xu. LiteTE: Lightweight, communication-efficient distributed-memory triangle enumerating. *IEEE Access*, 7:26294–26306, 2019.

[233] Y. Zhang, V. Kiriansky, C. Mendis, M. Zaharia, and S. P. Amarasinghe. Making caches work for graph analytics. In *IEEE International Conference on Big Data (BigData)*, pages 293–302, 2017.

[234] Y. Zhang and S. Parthasarathy. Extracting analyzing and visualizing triangle k-core motifs within networks. In *IEEE International Conference on Data Engineering (ICDE)*, pages 1049–1060, 2012.

[235] Y. Zhang and J. X. Yu. Unboundedness and efficiency of truss maintenance in evolving graphs. In *ACM SIGMOD International Conference on Management of Data*, page 1024–1041, 2019.

[236] F. Zhao and A. K. Tung. Large scale cohesive subgraphs discovery for social network visual analysis. *Proc. VLDB Endow.*, 6(2):85–96, 2012.

[237] W. Zhao, H. Ma, and Q. He. Parallel k-means clustering based on mapreduce. In M. G. Jaatun, G. Zhao, and C. Rong, editors, *Cloud Computing*, pages 674–679, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[238] Z. Zou. Bitruss decomposition of bipartite graphs. In *Database Systems for Advanced Applications*, pages 218–233, 2016.