FENNEL: Streaming Graph Partitioning for Massive Scale Graphs

Charalampos E. Tsourakakis, Christos Gkantsidis, Božidar Radunović, Milan Vojnović[1]

November 2012

Technical Report
MSR-TR-2012-113

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
`http://www.research.microsoft.com`

[1]C. E. Tsourakakis is with Carnegie Mellon University, Pittsburgh, PA, USA (ctsourak@math.cmu.edu) - work performed while an intern with Microsoft Research. C. Gkantsidis (chrisgk@microsoft.com), B. Radunović (bozidar@microsoft.com), and M. Vojnović (milanv@microsoft.com) are with Microsoft Research, Cambridge, United Kingdom.

Abstract – Graph partitioning is a key problem to enable efficient solving of a wide range of computational tasks and querying over large-scale graph data, such as computing node centralities using iterative computations, and personalized recommendations. In this work, we introduce a unifying framework for graph partitioning which enables a well principled design of scalable, streaming graph partitioning algorithms that are amenable to distributed implementation. We show that many previously proposed methods are special instances of this framework, we derive a novel one-pass, streaming graph partitioning algorithm and show that it yields significant benefits over previous approaches, using a large set of real-world and synthetic graphs.

Surprisingly, despite the fact that our algorithm is a one-pass streaming algorithm, we found its performance to be overall comparable to the de-facto standard offline software METIS, and it even outperforms it on numerous real-world graphs. For instance, for the Twitter graph with more than 1.4 billion of edges, our method partitions the graph in about 40 minutes achieving a balanced partition that cuts as few as 6.8% of edges, whereas it took more than $8\frac{1}{2}$ hours by METIS to produce a balanced partition that cuts 11.98% of edges. Furthermore, modularity–a popular measure for community detection [21, 47, 46]–is also a special instance of our framework. We establish the first rigorous approximation algorithm, achieving a guarantee of $O(\log(k)/k)$ for partitioning into k clusters.

Finally, we evaluate the performance gains by using our graph partitioner while solving standard PageRank computation in a graph processing platform, and observe significant gains in terms of the communication cost and runtime.

Keywords

Graph partitioning, Clustering, Streaming, Distributed Computing, Community Detection

1. INTRODUCTION

Nowadays, the scale of graph data that needs to be processed is massive. For example, in the context of online services, the Web graph amounts to at least one trillion of links [1], Facebook recently reported more than 1 billion of users and 140 billion of friend connections [2], and, in 2009, Twitter reported more than 40 million of users and about 1.5 billion of social relations [39]. The unprecedented proliferation of data provides us with new opportunities and benefits but also poses hard computational challenges. Frequent graph computations such as community detection [18], finding connected components [25], counting triangles [43], iterative computations using graph input data such as computing PageRank and its variations [48], shortest path and radius computations [11, 26] become challenging computational tasks in the realm of big graph data. An appealing solution to improve upon scalability is to partition massive graphs into smaller partitions and then use a large distributed system to process them. The sizes of the partitions have to be balanced to exploit the speedup of parallel computing over different partitions. Furthermore, it is critical that the number of edges between distinct partitions is small in order to minimize the communication cost incurred due to messages that are exchanged among different partitions. Many popular graph processing platforms such as Pregel [42] that builds on MapReduce [15], and its open source cousin Apache Giraph, PEGASUS [25] and GraphLab [41] use as a default partitioner Hash Partition of vertices, which corresponds to assigning each vertex to one of the k partitions uniformly at random. This heuristic would balance the number of vertices over different clusters, but being entirely oblivious to the graph structure, it may well result in suboptimal edge cuts. In fact, the expected fraction of edges cut by a random partition of vertices into $k \geq 1$ clusters is equal to $1 - 1/k$. Given the fact that real-world graphs tend to have sparser cuts [9], it is important to discover methods that are computationally efficient, practical, and yet yield high quality graph partitioning.

The problem of finding a balanced graph partition that minimizes the number of edges cut is known as the *balanced graph partitioning* problem, which has a rich history in the context of theoretical computer science. This problem is known to be NP-hard [37] and several approximation algorithms have been derived in previous work, which we review in Section 2. In practice, systems aim at providing good partitions in order to enhance their performance, e.g., [42, 50]. It is worth emphasizing that the balanced graph partitioning problem appears in various guises in numerous domains. For instance, it is critical for efficiently solving large-scale computational fluid dynamics and computational mechanics problems [59] and sparse linear systems [31].

Another major challenge in the area of big graph data is efficient processing of dynamic graphs. For example, new accounts are created and deleted every day in online services such as Facebook, Skype and Twitter. Furthermore, graphs created upon post-processing datasets such as Twitter posts are also dynamic, see for instance [7]. It is crucial to have efficient graph partitioners of dynamic graphs. For example, in the Skype service, each time a user logs in, his/her online contacts get notified. It is expensive when messages have to be sent across different graph partitions since this would typically involve using network infrastructure. The balanced graph partitioning problem in the dynamic setting is known as *streaming graph partitioning* [54]. Vertices (or edges) arrive and the decision of the placement of each vertex (edge) has to be done "on-the-fly" in order to incur as little computational overhead as possible.

It is worth noting that the state-of-the-art work on graph partitioning seems to roughly divide in two main lines of research. Rigorous mathematically work and algorithms that do not scale to massive graphs, e.g., [38], and heuristics that are used in practice [29, 30, 49, 54]. Our work contributes towards bridging the gap between theory and practice.

Summary of our Contributions. Our contributions can be summarized in the following points:

• We introduce a framework for graph partitioning that overcomes the computational complexity of the traditional balanced graph partitioning problem. Specifically, in the traditional balanced graph partitioning problem, the goal is to minimize the number of edges cut subject to hard constraints on the number of vertices in a cluster [8, 38]. We relax the hard cardinality constraints by formulating the graph partitioning objective function to consist of two elements: an element that accounts for the cost of edges cut and an element that that accounts for the cost related to the sizes of individual clusters.

• Our formulation provides a unifying framework that accommodates many of previously proposed heuristics as special cases. For example, the folklore heuristic of [49] which

	Fennel		Best competitor		Hash Partition		METIS	
# Clusters (k)	λ	ρ	λ	ρ	λ	ρ	λ	ρ
2	6.8%	1.1	34.3%	1.04	50%	1	11.98%	1.02
4	29%	1.1	55.0%	1.07	75%	1	24.39%	1.03
8	48%	1.1	66.4%	1.10	87.5%	1	35.96%	1.03

Table 1: Fraction of edges cut λ and the maximum load ρ for Fennel, the previously best-known heuristic (linear weighted degrees [54]) and hash partitioning of vertices for the Twitter graph with approximately 1.5 billion edges. Fennel and best competitor require around 40 minutes, METIS more than $8\frac{1}{2}$ hours.

places a vertex to the cluster with the fewest non-neighbors, and the degree-based heuristic of [54], which serves as the current state-of-the-art method with respect to performance.

• We show that interpolating between the non-neighbors heuristic [49] and the neighbors heuristic [54] provides improved performance for the balanced partitioning problem.

• A special case of interest is that of a modularity objective [21, 46, 5] for which we provide the first rigorous approximation guarantee. Specifically, we provide a $O(\frac{\log(k)}{k})$ approximation algorithm, for given number k of clusters. It is noteworthy that this result stands in a stark contrast to the best known approximation ratio for the traditional balanced graph partition, which becomes worse with the total number of vertices [38].

• We evaluate our proposed streaming graph partitioning method, Fennel, on a wide range of graph datasets, both real-world and synthetic graphs, showing that it produces high quality graph partitioning. Table 1 shows the performance of Fennel versus the best previously-known heuristic, which is the linear weighted degrees [54], and the baseline Hash Partition of vertices. We observe that Fennel achieves, simultaneously, significantly smaller fraction of edges cut and balanced cluster sizes.

• We also demonstrate the performance gains with respect to communication cost and run time while running iterative computations over partitioned input graph data in a distributed cluster of machines. Specifically, we evaluated Fennel and other partitioning methods by computing PageRank in the graph processing platform Apache Giraph. We observe significant gains with respect to the byte count among different clusters and run time in comparison with the baseline Hash Partition of vertices, and the best-performing competitor.

Structure of the Paper. The remainder of the paper is organized as follows. Section 2 discusses the related work. In Section 3, we introduce our graph partitioning framework and present our main theoretical result. In Section 4, we present our scalable, streaming algorithm. In Section 5, we evaluate our method versus the state-of-the-art work on a broad set of real-world and synthetic graphs, while in Section 6 we provide our experimental results in the Apache Giraph. In Section 7 we discuss certain aspects of Fennel. In Section 8, we conclude.

2. RELATED WORK

Graph partitioning is an NP-hard problem [20] with numerous applications in various domains, with a long history and a very active present. Here, we discuss related work to graph partitioning in two major different settings. The first setting which serves as our initial motivation is the balanced graph partitioning problem [37]. The second setting is related to community detection [18].

Balanced graph partitioning problem: A fundamental problem in every parallel and distributed application is data placement since it typically affects significantly the execution efficiency of jobs, e.g., [60, 32]. This is particularly true for graphs, as interaction locality can easily be inferred from the graph edges. One class of examples are social graphs [27, 50], where operations are user interactions, defined through social engagements represented with graph edges. However, many other graph algorithms benefit from careful graph partitioning and placement, such as machine learning and data mining [23, 25, 54]. The goal of balanced graph partitioning is to minimize an application's overall runtime. This is achieved by assigning to each processor/machine an equal amount of data and concurrently minimizing the parallel/distributed overhead by minimizing the number of edges cut by the corresponding partition. Formally, the (k, ν)-balanced graph partitioning asks to divide the vertices of a graph in components each of size less than $\nu\frac{n}{k}$. The case $k = 2, \nu = 1$ is equivalent to the *minimum bisection problem*, an NP-hard problem [20]. Several approximation algorithms, e.g., [16], and heuristics, e.g., [17, 33] exist for this problem. When $\nu = 1+\epsilon$ for any desired but fixed ϵ there exists a $O(\epsilon^{-2}\log^{1.5} n)$ approximation algorithm [37]. When $\nu = 2$ there exists an $O(\sqrt{\log k \log n})$ approximation algorithm based on semidefinite programming (SDP) [38]. Due to the practical importance of k-partitioning there exist several heuristics, among which METIS [52] and its parallel version [53] stand out for their fine performance. METIS is widely used in many existing systems [27]. There are also heuristics that improve efficiency and partition quality of METIS in a distributed system [51]. An extensive summary of existing heuristics can be found in [3].

The methods above are offline. Recently, Stanton and Kliot worked on the online partitioning problem [54]. This setting is also well adapted to dynamic graphs, where offline methods incur an expensive computational cost, requiring to repartition the entire graph. Moreover, the newly obtained partitioning can significantly differ from the old one. This in turn implies a large reshuffle of the data, which is very costly in a distributed system. Online algorithms assign vertices to components as they arrive and never reassign them later. One of the earliest online algorithm was proposed by Kernighan and Lin [33], which is also used as a subroutine in METIS. Currently the most advanced online partitioning algorithm is by Stanton and Kliot [54], which we extensively compare our approach against.

It is also worth noting a separate line of work whose goal is to optimize small dynamic graph queries [34, 50, 55]. In this setting queries are small and are ideally executed on a single machine, and dynamic replication is used to make sure that most of the data is available locally. Our case is different as queries are large and each query is executed on many or all of the sites. We note that our partitioning algorithm can be deployed in parallel with the replication methods.

Community Detection: Finding communities, i.e., groups of vertices within which connections are dense, but between which connections are sparser, is a central problem of network science [44]. The difference between the balanced graph partitioning problem is that there exists no no restriction on the number of vertices per subset. Another difference that also arises frequently in practice is that we do not know the true number of clusters a priori, or even their existence. The notion of a community has been formalized in various ways, see [40]. Of interest to our work is the popular modularity measure [21, 47, 46]. Modularity measures the number of within-community edges relative to a null random graph model that is usually considered to be a random graph with the same degree distribution. Despite the popularity of modularity, a few rigorous results exist. Specifically, Brandes et al. [10] proved that maximizing modularity is NP-hard. Approximation algorithms without theoretical guarantees whose performance is evaluated in practice also exist [5]. We advance this line of work.

3. PROPOSED METHOD

In this section we introduce our framework for graph partitioning and our main theoretical result. We first present our general graph partitioning framework in Section 3.1, and then discuss the classic balanced graph partitioning in Section 3.2 as a special instance of our framework. We discuss the intuition behind our approach and state our main theoretical result which provides the first rigorous approximation algorithm for computing the modularity of a graph under an Erdös-Rényi null model.

Notation. Throughout the paper we use the following notation. Let $G(V, E)$ be a simple, undirected graph. Let the number of vertices and edges be denoted as $|V| = n$ and $|E| = m$. For a subset of vertices $S \subseteq V$, let $e(S, S)$ be the set of edges with both end vertices in the set S, and let $e(S, V \setminus S)$ be the set of edges whose one end-vertex is in the set S and the other is not. For a given vertex v let $t_S(v)$ be the number of triangles (v, w, z) such that $w, z \in S$. We define a *partition* of vertices $\mathcal{P} = (S_1, \dots, S_k)$ to be a family of pairwise disjoint sets vertices, i.e., $S_i \subseteq V$, $S_i \cap S_j = \emptyset$ for every $i \neq j$. We call S_i to be a cluster of vertices. Finally, for a graph $G = (V, E)$ and a partition $\mathcal{P} = (S_1, S_2, \dots, S_k)$ of the vertex set V, let $\partial e(\mathcal{P})$ be the set of edges that cross partition boundaries, i.e. $\partial e(\mathcal{P}) = \cup_{i=1}^{k} e(S_i, V \setminus S_i)$.

3.1 Our Graph Partitioning Framework

We formulate a graph partitioning framework that is based on accounting for the cost of internal edges and the cost of edges cut by a partition of vertices in a single global objective function.

The size of individual partitions. We denote with $\sigma(S_i)$ the size of the cluster of vertices S_i, where σ is a mapping to the set of real numbers. Special instances of interest are (1) *edge cardinality* where the size of the cluster i is proportional to the total number of edges with at least one end-vertex in the set S_i, i.e. $|e(S_i, S_i)| + |e(S_i, V \setminus S_i)|$, (2) *interior-edge cardinality* where the size of cluster i is proportional to the number of internal edges $|e(S_i, S_i)|$, and (3) *vertex cardinality* where the size of partition i is proportional to the total number of vertices $|S_i|$. The edge cardinality of a cluster is an intuitive measure of cluster size. This is of interest for computational tasks over input graph data where the computational complexity within a cluster of vertices is linear in the number of edges with at least one vertex in the given cluster. For example, this is the case for iterative computations such as solving the power iteration method. The vertex cardinality is a standard measure of the size of a cluster and for some graphs may serve as a proxy for the edge cardinality, e.g. for the graphs with bounded degrees.

The global objective function. We define a global objective function that consists of two elements: (1) the inter-partition cost $c_{\mathrm{OUT}} : N^k \to \mathbb{R}_+$ and (2) the intra-partition cost $c_{\mathrm{IN}} : N^k \to \mathbb{R}_+$. These functions are assumed to be increasing and super-modular (or convex, if extended to the set of real numbers). For every given partition of vertices $\mathcal{P} = (S_1, S_2, \dots, S_k)$, we define the global cost function as

$$\begin{aligned} f(\mathcal{P}) &= c_{\mathrm{OUT}}(|e(S_1, V \setminus S_1)|, \dots, |e(S_k, V \setminus S_k)|) \\ &\quad + c_{\mathrm{INT}}(\sigma(S_1), \dots, \sigma(S_k)). \end{aligned}$$

It is worth mentioning some particular cases of interest. Special instance of interest for the inter-partition cost is the linear function in the total number of cut edges $|\partial e(\mathcal{P})|$. This case is of interest in cases where an identical cost is incurred per each edge cut, e.g. in cases where messages are exchanged along cut edges and these messages are transmitted through some common network bottleneck. For the intra-partition cost, a typical goal is to balance the cost across different partitions and this case is accommodated by defining $c_{\mathrm{INT}}(\sigma(S_1), \dots, \sigma(S_k)) = \sum_{i=1}^{k} c(\sigma(S_i))$, where $c(x)$ is a convex increasing function such that $c(0) = 0$. In this case, the intra-partition cost function, being defined as a sum of convex functions of individual cluster sizes, would tend to balance the cluster sizes, since the minimum is attained when sizes are equal.

We formulate the graph partitioning problem as follows.

Optimal k-Graph Partitioning

Given a graph $G = (V, E)$, find a partition $\mathcal{P}^* = \{S_1^*, \dots, S_k^*\}$ of the vertex set V, such that $f(\mathcal{P}^*) \geq f(\mathcal{P})$, for all partitions $\mathcal{P}$ such that $|\mathcal{P}| = k$.

We refer to the partition $\mathcal{P}^*$ as the optimal k graph partition of the graph G.

Streaming setting. The streaming graph partitioning problem can be defined as follows. Let $G = (V, E)$ be an input graph and let us assume that we want to partition the graph into k disjoint subsets of vertices. The vertices arrive in some order, each one with the set of its neighbors.We consider three different stream orders, as in [54].

- Random: Vertices arrive according to a random permutation.
- BFS: This ordering is generated by selecting a vertex uniformly at random and performing breadth first search starting from that vertex.
- DFS: This ordering is identical to the BFS ordering, except that we perform depth first search.

A k-partitioning streaming algorithm has to decide whenever a new vertex arrives to which cluster it is going to be placed. A vertex is never moved after it has been assigned to a cluster. The formal statement of the problem follows.

3.2 Classic Balanced Graph Partitioning

In this section we consider the traditional instance of a graph partitioning problem that is a special case of our framework by defining the inter-partition cost to be equal to the total number of edges cut and the intra-partition cost defined in terms of the vertex cardinalities.

The starting point in the existing literature, e.g., [37, 38], is to admit hard cardinality constraints, so that $|S_i^*| \leq \nu \frac{n}{k}$ for $i = 1, \ldots, k$, where $\nu \geq 1$ is a fixed constant. This set of constraints makes the problem significantly hard. Currently, state-of-the-art work depends on the impressive ARV barrier [8] which results in a $O(\sqrt{\log n})$ approximation factor. The typical formulation is the following:

$$\boxed{\begin{array}{rl} \text{minimize}_{\mathcal{P}=(S_1,\ldots,S_k)} & |\partial e(\mathcal{P})| \\ \text{subject to} & |S_i| \leq \nu \frac{n}{k}, \ \forall i \in \{1,\ldots,k\} \end{array}}$$

Our approach: Just Relax ! The idea behind our approach is to *relax* the hard cardinality constraints by introducing a term in the objective $c_{\text{IN}}(\mathcal{P})$ whose minimum is achieved when $|S_i| = \frac{n}{k}$ for all $i \in \{1, \ldots, k\}$. Therefore, our framework is based on a well-defined global graph partitioning objective function, which allows for a principled design of approximation algorithms and heuristics as shall be demonstrated in Section 4. Our graph partitioning method is based on solving the following optimization problem:

$$\boxed{\text{minimize}_{\mathcal{P}=(S_1,\ldots,S_k)} \quad |\partial e(\mathcal{P})| + c_{\text{IN}}(\mathcal{P})} \qquad (1)$$

Intra-partition cost: With the goal in mind to favor balanced partitions, we may define the intra-partition cost function by $c_{\text{IN}}(\mathcal{P}) = \sum_{i=1}^{k} c(|S_i|)$ where $c(x)$ is an increasing function chosen to be *super-modular*, so that the following increasing returns property holds $c(x+1) - c(x) \geq c(y+1) - c(y)$, for every $0 \leq y \leq x$.

We shall focus our attention to the following family of functions $c(x) = \alpha x^{\gamma}$, for $\alpha > 0$ and $\gamma \geq 1$. By the choice of the parameter γ, this family of cost functions allows us to control how much the imbalance of cluster sizes is accounted for in the objective function. In one extreme case where $\gamma = 1$, we observe that the objective corresponds to minimizing the number of cut-edges, thus entirely ignoring any possible imbalance of the cluster sizes. On the other hand, by taking larger values for the parameter γ, the more weight is put on the cost of partition imbalance, and this cost may be seen to approximate hard constraints on the imbalance in the limit of large γ. Parameter α is also important. We advocate a principled choice of α independently of whether it is suboptimal compared to other choices. Specifically, we choose $\alpha = m\frac{k^{\gamma-1}}{n^{\gamma}}$. This provides us a proper scaling, since for this specific choice of α, our optimization problem is equivalent to minimizing a natural normalization of the objective function $\frac{\sum_{i=1}^{k} e(S_i, V \setminus S_i)}{m} + \frac{1}{k}\sum_{i=1}^{k}\left(\frac{|S_i|}{\frac{n}{k}}\right)^{\gamma}$.

An equivalent maximization problem. We note that the optimal k graph partitioning problem admits an equivalent formulation as a maximization problem. It is of interest to consider this alternative formulation as it allows us to make a connection with the concept of graph modularity, which we do later in this section. For a graph $G = (V, E)$ and $S \subseteq V$, we define the function $h : 2^V \to \mathbb{R}$ as:

$$h(S) = |e(S, V \setminus S)| - c(|S|)$$

where $h(\emptyset) = h(\{v\}) = 0$ for every $v \in V$. Given $k \geq 1$ and a partition $\mathcal{P} = \{S_1, \ldots, S_k\}$ of the vertex set V, we define the function g as

$$g(\mathcal{P}) = \sum_{i=1}^{k} h(S_i).$$

Now, we observe that maximizing the function $g(\mathcal{P})$ over all possible partitions $\mathcal{P}$ of the vertex set V such that $|\mathcal{P}| = k$ corresponds to the k graph partitioning problem. Indeed, this follows by noting that

$$\begin{aligned} g(\mathcal{P}) &= \sum_{i=1}^{k} |e(S_i, S_i)| - c(|S_i|) \\ &= (m - \sum_{i=1}^{k} |e(S_i, V \setminus S_i)|) - c(|S_i|) \\ &= m - f(\mathcal{P}). \end{aligned}$$

Thus, maximizing function $g(\mathcal{P})$ corresponds to minimizing function $f(\mathcal{P})$, which is precisely the objective of our k graph partitioning problem.

Modularity: We note that when the function $c(x)$ is taken from the family $c(x) = \alpha x^{\gamma}$, for $\alpha > 0$ and $\gamma = 2$, our objective has a combinatorial interpretation. Specifically, our problem is equivalent to maximizing the function

$$\sum_{i=1}^{k} [|e(S_i, S_i)| - p\binom{|S_i|}{2}]$$

where $p = \alpha/2$. In this case, each summation element admits the following interpretation: it corresponds to the difference between the realized number of edges within a cluster and the expected number of edges within the cluster under the null-hypothesis that the graph is an Erdös-Rényi random graph with parameter p. This is intimately related to the concepts of graph modularity [21, 47, 46] and quasi-cliques [4]. For this special case, because of the combinatorial meaning of the objective function, we design a non-trivial, semidefinite programming (SDP) algorithm and show that it provides provides the following guarantee.

THEOREM 1. *There exists a polynomial time algorithm which provides an $\Omega(\frac{\log k}{k})$-approximation guarantee for the Optimal k Graph Partitioning.*

The proof is based on a partitioning algorithm that is derived by using a randomized rounding of a solution to our SDP relaxation of the original combinatorial optimization problem.

Notice that compared to the existing literature we avoid any dependence on the number of vertices n, exactly because we avoid the hard cardinality constraints. Compared to the random assignment baseline, where a vertex is assigned randomly to a cluster, which we analyze [56], we improve by a logarithmic factor. This suggests that the offline SDP solver will yield significantly better performance for small values of k. It is worth noting that we conjecture that our approximation factor is the best possible, see Appendixfor the grounds of this conjecture. Finally, we outline that despite the fact that semidefinite programming has been used to approximate "in-practice" modularity [5], to the best of our knowledge, Theorem 2 would provide the first rigorous approximation guarantee for the given problem.

4. ONE-PASS STREAMING ALGORITHM

We derive a streaming algorithm by using a greedy assignment of vertices to partitions as follows: assign each arriving vertex to a partition such that the objective function of the k graph partitioning problem, defined as a maximization problem, is increased the most. Formally, given that current vertex partition is $\mathcal{P} = (S_1, S_2, \dots, S_k)$, a vertex v is assigned to partition i such that

$$\begin{aligned} & g(S_1, \dots, S_i \cup \{v\}, \dots, S_j, \dots, S_k) \\ \geq \ & g(S_1, \dots, S_i, \dots, S_j \cup \{v\}, \dots, S_k), \text{ for all } j \in [k]. \end{aligned}$$

Defining $\delta g(v, S_i) = g(S_1, \dots, S_i \cup \{v\}, \dots, S_j, \dots, S_k) - g(S_1, \dots, S_i, \dots, S_j, \dots, S_k)$, the above greedy assignment of vertices corresponds to that in the following algorithm.

Greedy vertex assignment

- Assign vertex v to partition i such that $\delta g(v, S_i) \geq \delta g(v, S_j)$, for all $j \in [k]$

Special case: edge-cut and balanced vertex cardinality. This is a special case of introduced that we discussed in Section 3.1. In this case, $\delta g(v, S_l) = |N(v) \cap S_l| - \delta c(|S_l|)$, where $\delta c(x) = c(x+1) - c(x)$, for $x \in \mathbb{R}_+$, and $N(v)$ denotes the set of neighbors of vertex v. The two summation elements in the greedy index $\delta g(v, S_l)$ account for the two underlying objectives of minimizing the number of cut edges and balancing of the partition sizes. Notice that the component $|N(v) \cap S_i|$ corresponds to the number of neighbours of vertex v that are assigned to partition S_i. In other words, this corresponds to the degree of vertex v in the subgraph induced by S_i. On the other hand, the component $\delta c(|S_i|)$ can be interpreted as the marginal cost of increasing the partition i by one additional vertex.

For our special family of cost functions $c(x) = \alpha x^\gamma$, we have $\delta c(x) = \alpha \gamma x^{\gamma - 1}$. For $\gamma = 1$, the greedy index rule corresponds to assigning a new vertex v to partition i with the largest number of neighbours in S_i, i.e $|N(v) \cap S_i|$. This is one of the greedy rules considered by Stanton and Kliot [54], and is a greedy rule that may result in highly imbalanced partition sizes.

On the other hand, in case of quadratic cost $c(x) = \frac{1}{2}x^2$, the greedy index is $|N(v) \cap S_i| - |S_i|$, and the greedy assignment corresponds to assigning a new vertex v to partition i that minimizes the number of *non-neighbors* of v inside S_i, i.e. $|S_i \setminus N(v)|$. Hence, this yields the following heuristic: *place a vertex to the partition with the least number of non-neighbors* [49]. This assignment accounts for both the cost of cut edges and the balance of partition sizes.

Finally, we outline that in many applications there exist very strict constraints on the load balance. Despite the fact that we investigate the effect of the parameter γ on the load balance, one may apply the following algorithm, which enforces to consider only machines whose load is at most $\nu \times \frac{n}{k}$. This algorithm for $1 \leq \gamma \leq 2$ amounts to interpolating between the basic heuristics of [54] and [49]. The overall complexity of our algorithm is $O(n+m)$.

	Nodes	Edges	Description
amazon0312	400 727	2 349 869	Co-purchasing
amazon0505	410 236	2 439 437	Co-purchasing
amazon0601	403 364	2 443 311	Co-purchasing
as-735	6 474	12 572	Auton. Sys.
as-Skitter	1 694 616	11 094 209	Auton. Sys.
as-caida	26 475	53 381	Auton. Sys.
ca-AstroPh	17 903	196 972	Collab.
ca-CondMat	21 363	91 286	Collab.
ca-GrQc	4 158	13 422	Collab.
ca-HepPh	11 204	117 619	Collab.
ca-HepTh	8 638	24 806	Collab.
cit-HepPh	34 401	420 784	Citation
cit-HepTh	27 400	352 021	Citation
cit-Patents	3 764 117	16 511 740	Citation
email-Enron	33 696	180 811	Email
email-EuAll	224 832	339 925	Email
epinions	119 070	701 569	Trust
Epinions1	75 877	405 739	Trust
LiveJournal1	4 843 953	42 845 684	Social
p2p-Gnutella04	10 876	39 994	P2P
p2p-Gnutella05	8 842	31 837	P2P
p2p-Gnutella06	8 717	31 525	P2P
p2p-Gnutella08	6 299	20 776	P2P
p2p-Gnutella09	8 104	26 008	P2P
p2p-Gnutella25	22 663	54 693	P2P
p2p-Gnutella31	62 561	147 878	P2P
roadNet-CA	1 957 027	2 760 388	Road
roadNet-PA	1 087 562	1 541 514	Road
roadNet-TX	1 351 137	1 879 201	Road
Slashdot0811	77 360	469 180	Social
Slashdot0902	82 168	504 230	Social
Slashdot081106	77 258	466 661	Social
Slashdot090216	81 776	495 661	Social
Slashdot090221	82 052	498 527	Social
usroads	126 146	161 950	Road
wb-cs-stanford	8 929	2 6320	Web
web-BerkStan	654 782	6 581 871	Web
web-Google	855 802	4 291 352	Web
web-NotreDame	325 729	1 090 108	Web
web-Stanford	255 265	1 941 926	Web
wiki-Talk	2 388 953	4 656 682	Web
Wikipedia-20051105	1 596 970	18 539 720	Web
Wikipedia-20060925	2 935 762	35 046 792	Web
Twitter	41 652 230	1 468 365 182	Social

Table 2: Datasets used in our experiments.

Greedy vertex assignment with threshold ν

- Let $I_\nu = \{i : \mu_i \leq \nu \frac{n}{k}\}$. Assign vertex v to partition $i \in I_\nu$ such that $\delta g(v, S_i) \geq \delta g(v, S_j)$, for all $j \in I_\nu$

5. EXPERIMENTAL EVALUATION

In this section we present results of our experimental evaluations of the quality of graph partitions created by our method and compare with alternative methods. We first describe our experimental setup in Sections 5.1, and then present our findings using synthetic and real-world graphs, in Section 5.2 and 5.3, respectively.

5.1 Experimental Setup

The real-world graphs used in our experiments are shown in Table 2. Multiple edges, self loops, signs and weights were removed, if any. Furthermore, we considered the largest connected component from each graph in order to ensure that there is a non-zero number of edges cut. All graphs are publicly available on the Web. All algorithms have been

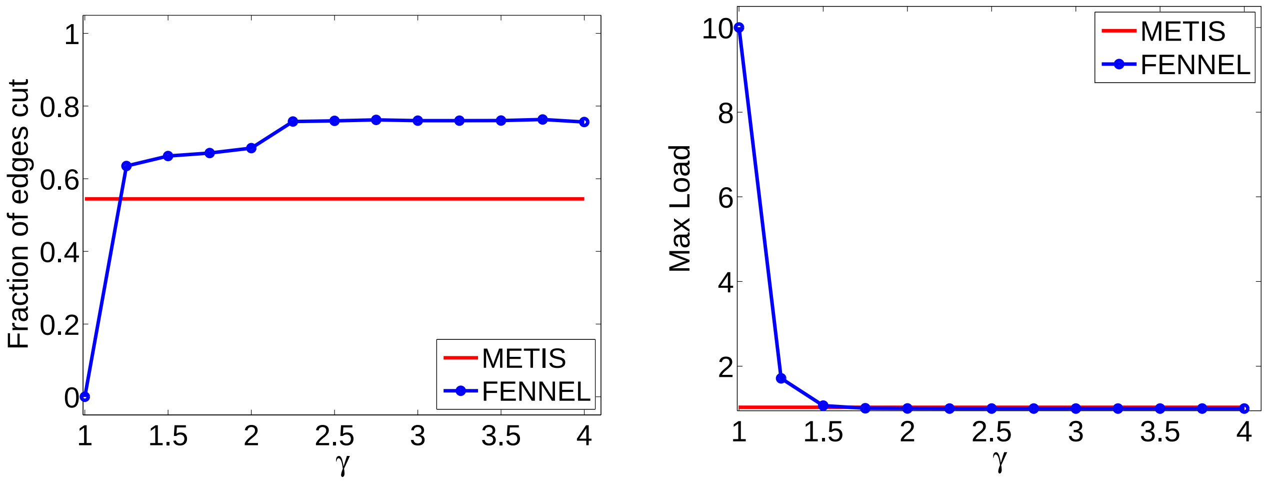


Figure 1: Fraction of edges cut λ and maximum load normalized ρ as a function of γ, ranging from 1 to 4 with a step of 0.25, over five randomly generated power law graphs with slope 2.5. The straight lines show the performance of METIS.

	BFS		Random	
Method	λ	ρ	λ	ρ
H	96.9%	1.01	96.9%	1.01
B [54]	97.3%	1.00	96.8%	1.00
DG [54]	0%	32	43%	1.48
LDG [54]	34%	1.01	40%	1.00
EDG [54]	39%	1.04	48%	1.01
T [54]	61%	2.11	78%	1.01
LT [54]	63%	1.23	78%	1.10
ET [54]	64%	1.05	79%	1.01
NN [49]	69%	1.00	55%	1.03
Fennel	14%	1.10	14%	1.02
METIS [29]	8%	1.00	8%	1.02

Table 3: Performance of various existing methods on amazon0312, k is set to 32, illustrating the typical performance of various methods: Fennel outperforms one-pass, streaming competitors and has performance inferior but comparable to METIS.

implemented in JAVA, and all experiments were performed on a single machine, with Intel Xeon CPU at 3.6GHz, and 16GB of main memory. Wall-clock times include only the algorithm execution time, excluding the required time to load the graph into memory.

In our synthetic experiments, we use two random graph models. The first model is the hidden partition model [13]. It is specified by four parameters parameters: the number of vertices n, the number of clusters k, the intercluster and intracluster edge probabilities p and q, respectively. First, each vertex is assigned to one of k clusters uniformly at random. We add an edge between two vertices of the same (different) cluster(s) with probability p (q) independently of the other edges. We denote this model as $\mathsf{HP}(n,k,p,q)$. The second model we use is a standard model for generating random power law graphs. Specifically, we first generate a power-law degree sequence with a given slope δ and use the Chung-Lu random graph model to create an instance of a power law graph [12]. The model $\mathsf{CL}(n,\delta)$ has two parameters: the number of vertices n and the slope δ of the expected power law degree sequence.

We evaluate our algorithms by measuring two quantities from the resulting partitions. In particular, for a fixed partition $\mathcal{P}$ we use the measures of the *fraction of edges cut* λ and the *normalized maximum load* ρ, defined as

$$\lambda = \frac{\#\text{ edges cut by }\mathcal{P}}{\#\text{ total edges}} = \frac{|\partial e(\mathcal{P})|}{m}, \text{ and}$$

$$\rho = \frac{\text{maximum load}}{\frac{n}{k}}.$$

Throughout this section, we also use the notation $\lambda_{\mathcal{M}}$ and $\mu_{\mathcal{M}}$ to indicate the partitioning method $\mathcal{M}$ used in a particular context. In general, we omit indices whenever it is clear to which partition method we refer to. Notice that $k \geq \rho \geq 1$ since the maximum load of a cluster is at most n and there always exists at least one cluster with at least $\frac{n}{k}$ vertices.

In Section 5.2, we use the greedy vertex assignment without any threshold. Given that we are able to control ground truth, we are mainly interested in understanding the effect of the parameter γ on the tradeoff between the fraction of edges cut and the normalized maximum load. In Section 5.3, the setting of the parameters we use throughout our experiments is $\gamma = \frac{3}{2}$, $\alpha = \sqrt{k}\frac{m}{n^{3/2}}$, and $\nu = 1.1$. The choice of γ is based on our findings from Section 5.2 and of α based on Section 3. Finally, $\nu = 1.1$ is a reasonable load balancing factor for real-world settings.

As our competitors we use state-of-the-art heuristics. Specifically, in our evaluation we consider the following heuristics from [54], which we briefly describe here for completeness. Let v be the newly arrived vertex.

- Balanced (B): place v to the cluster S_i with minimal size.
- Hash partitioning (H): place v to a cluster chosen uniformly at random.
- Deterministic Greedy (DG): place v to S_i that maximizes $|N(v) \cap S_i|$.

		Fennel		METIS	
m	k	λ	ρ	λ	ρ
7 185 314	4	62.5 %	1.04	65.2%	1.02
6 714 510	8	82.2 %	1.04	81.5%	1.02
6 483 201	16	92.9 %	1.01	92.2%	1.02
6 364 819	32	96.3%	1.00	96.2%	1.02
6 308 013	64	98.2%	1.01	97.9%	1.02
6 279 566	128	98.4 %	1.02	98.8%	1.02

Table 4: Fraction of edges cut λ and normalized maximum load ρ for Fennel and METIS [29] averaged over 5 random graphs generated according to the HP(5000,0.8,0.5) model. As we see, Fennel despite its small computational overhead and deciding on-the-fly where each vertex should go, achieves comparable performance to METIS.

- Linear Weighted Deterministic Greedy (LDG): place v to S_i that maximizes $|N(v) \cap S_i| \times (1 - \frac{|S_i|}{\frac{n}{k}})$.
- Exponentially Weighted Deterministic Greedy (EDG): place v to S_i that maximizes $|N(v) \cap S_i| \times \Big(1 - \exp\big(|S_i| - \frac{n}{k}\big)\Big)$
- Triangles (T): place v to S_i that maximizes $t_{S_i}(v)$.
- Linear Weighted Triangles (LT): place v to S_i that maximizes $t_{S_i}(v) \times \Big(1 - \frac{|S_i|}{\frac{n}{k}}\Big)$.
- Exponentially Weighted Triangles (ET): place v to S_i that maximizes $t_{S_i}(v) \times \Big(1 - \exp\big(|S_i| - \frac{n}{k}\big)\Big)$.
- Non-Neighbors (NN): place v to S_i that minimizes $|S_i \setminus N(v)|$.

In accordance with [54], we observed that LDG is the best performing heuristic. Even if Stanton and Kliot do not compare with NN, LDG outperforms it also. Non-neighbors typically have very good load balancing properties, as LDG as well, but cut significantly more edges. Table 3 shows the typical performance we observe across all datasets. Specifically, it shows λ and ρ for both BFS and random order for amazon0312. DFS order is omitted since qualitatively it does not differ from BFS. We observe that LDG is the best competitor, Fennel outperforms all existing competitors and is inferior to METIS, but of comparable performance. In whatever follows, whenever we refer to the best competitor, unless otherwise mentioned we refer to LDG. Time-wise METIS is the fastest, taking 11.4 seconds to run. Hashing follows with 12 seconds and the rest of the methods except for T, LT, ET take the same time up the integer part, i.e., 13 seconds. Triangle based methods take about 10 times more time. Existing approximate counting methods can mitigate this [57, 58]. It is also worth emphasizing that for larger graphs Fennel is faster than METIS.

5.2 Synthetic Datasets

Before we delve into our findings, it is worth summarizing the main findings of this section. (a) For all synthetic graphs we generated, the value $\gamma = \frac{3}{2}$ achieves the best performance *pointwise*, not in average. (b) The effect of the stream order is minimal on the results. Specifically, when $\gamma \geq \frac{3}{2}$ all orders result in the same qualitative results. When $\gamma < \frac{3}{2}$ BFS and DFS orders result in the same results which are worse with respect to load balancing –and hence better for the edge cuts– compared to the random order. (c) Fennel's performance is comparable to METIS.

Hidden Partition: We report averages over five randomly generated graphs according to the model HP$(5000, k, 0.8, 0.5)$ for each value of k we use. We study (a) the effect of the parameter γ, which parameterizes the function $c(x) = \alpha x^\gamma$, and (b) the effect of the number of clusters k.

We range γ from 1 to 4 with a step of 1/4, for six different values of k shown in the second column of Table 4. For all k, we observe, consistently, the following behavior: for $\gamma = 1$ we observe that $\lambda = 0$ and $\rho = k$. This means that one cluster receives all vertices. For any γ greater than 1, we obtain excellent load balancing with ρ ranging from 1 to 1.05, and the same fraction of edges cut with METIS up the the first decimal digit. This behavior was not expected a priori, since in general we expect λ shifting from small to large values and see ρ shifting from large to small values as γ grows. Given the insensitivity of Fennel to γ in this setting, we fix $\gamma = \frac{3}{2}$ and present in Table 4 our findings. For each k shown in the second column we generate five random graphs. The first column shows the average number of edges. Notice that despite the fact that we have only 5,000 vertices, we obtain graphs with several millions of edges. The four last columns show the performance of Fennel and METIS. As we see, their performance is comparable and in one case (k=128) Fennel clearly outperforms METIS.

Power Law: It is well known that power law graphs have no good cuts [23], but they are commonly observed in practice. We examine the effect of parameter γ for k fixed to 10. In contrast to the hidden partition experiment, we observe the expected tradeoff between λ and ρ as γ changes. We generate five random power law graphs CL(20 000,2.5), since this value matches the slope of numerous real-world networks [45]. Figure 1 shows the tradeoff when γ ranges from 1 to 4 with a step of 0.25 for the random stream order. The straight line shows the performance of METIS. As we see, when $\gamma < 1.5$, ρ is unacceptably large for demanding real-world applications. When $\gamma = 1.5$ we obtain essentially the same load balancing performance with METIS. Specifically, $\rho_{\text{Fennel}} = 1.02$, $\rho_{\text{METIS}} = 1.03$. The corresponding cut behavior for $\gamma = 1.5$ is $\lambda_{\text{Fennel}} = 62.58\%$, $\lambda_{\text{METIS}} = 54.46\%$. Furthermore, we experimented with the random, BFS and DFS stream orders. We observe that the only major difference between the stream orders is obtained for $\gamma = 1.25$. For all other γ values the behavior is identical. For $\gamma = 1.25$ we observe that BFS and DFS stream orders result in significantly worse load balancing properties. Specifically, $\rho_{\text{BFS}} = 3.81$, $\rho_{\text{DFS}} = 3.73$, $\rho_{\text{Random}} = 1.7130$. The corresponding fractions of edges cut are $\lambda_{\text{BFS}} = 37.83\%$, $\lambda_{\text{DFS}} = 38.85\%$, and $\lambda_{\text{Random}} = 63.51\%$.

5.3 Real-World Datasets

Again, before we delve into the details of the experimental results, we summarize the main points of this Section: (1) *Fennel is superior to existing streaming partitioning algorithms.* Specifically, it consistently, over a wide range of k values and over all datasets, performs better than the current state-of-the-art. Fennel achieves excellent load balancing with significantly smaller edge cuts. (2) For smaller values of k (less or equal than 64) the observed gain is more pronounced. (c) *Fennel is fast.* Our implementation scales

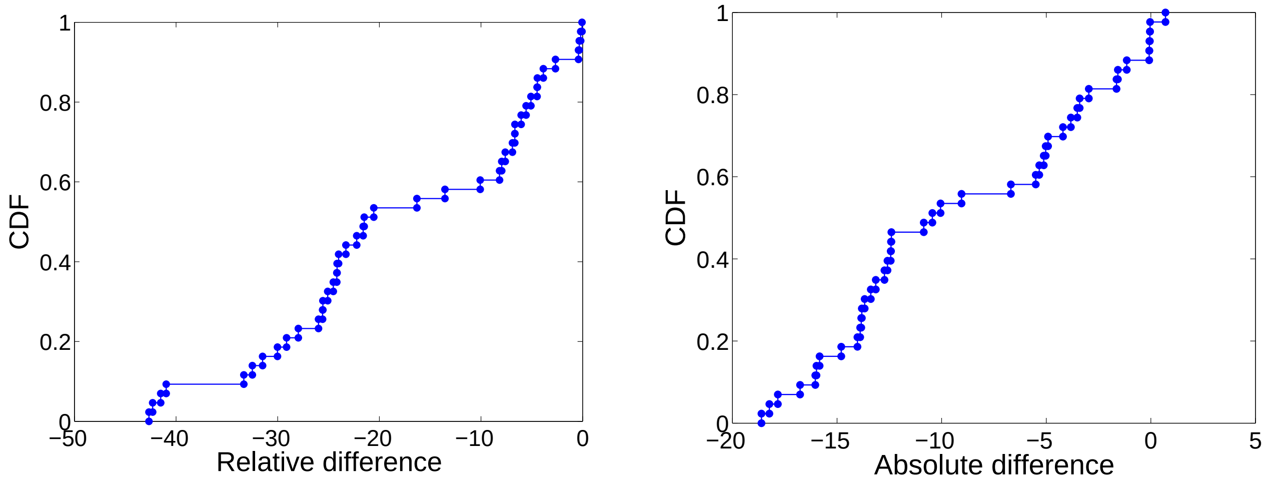


Figure 2: (Left) CDF of the relative difference $(\frac{\lambda_{\mathbf{Fennel}}-\lambda_{\mathbf{c}}}{\lambda_{\mathbf{c}}}) \times 100\%$ of percentages of edges cut of our method and the best competitor. (Right) Same but for the absolute difference $(\lambda_{\mathbf{Fennel}} - \lambda_{\mathbf{c}}) \times 100\%$.

k	Relative Gain	$\rho_{\text{Fennel}} - \rho_{\text{c}}$
2	25.37%	0.47%
4	25.07%	0.36%
8	26.21%	0.18%
16	22.07%	-0.43%
32	16.59%	-0.34%
64	14.33%	-0.67%
128	13.18%	-0.17%
256	13.76%	-0.20%
512	12.88%	-0.17%
1024	11.24%	-0.44%

Table 5: The relative gain $(1 - \frac{\lambda_{\mathbf{Fennel}}}{\lambda_{\mathbf{c}}}) \times 100\%$ and load imbalance, where subindex c stands for the best competitor, averaged over all datasets in Table 1 as a function of k. Our method consistently yields sparser edge cut the best competitor. The benefit is more pronounced for the smaller values of k.

well with the size of the graph. It takes about 40 minutes to partition the Twitter graph which has more than 1 billion of edges.

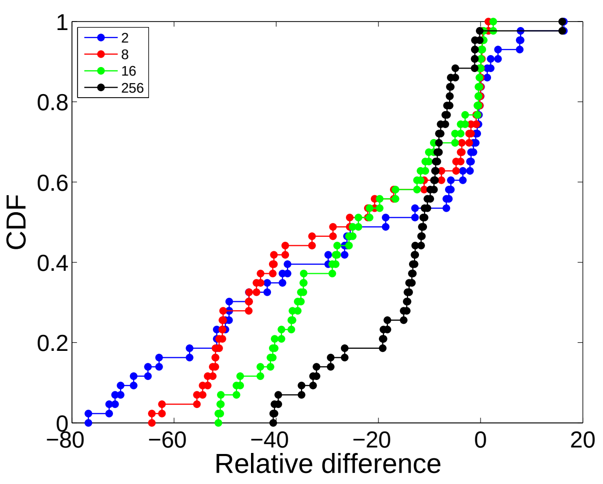
Twitter Graph. Twitter graph is the largest graph in our collection of graphs, with more than 1.4 billion edges. This feature makes it the most interesting graph from our collection, even if, admittedly, is a graph that can be loaded into the main memory. The results of Fennel on this graph are excellent. Specifically, Table 1 shows the performance of Fennel, the best competitor LDG, the baseline Hash Partition and METIS for $k = 2, 4$ and 8. All methods achieve balanced partitions, with $\rho \leq 1.1$. Fennel, is the only method that always attains this upper bound. However, this reasonable performance comes with a high gain for λ. Specifically, we see that Fennel achieves better performance of $k = 2$ than METIS. Furthermore, Fennel requires 42 minutes whereas METIS requires $8\frac{1}{2}$ hours. Most importantly, Fennel outperforms LDG consistently. Specifically, for $k = 16, 32$ and 64, Fennel achieves the following results $(\lambda, \rho) = (59\%, 1.1)$, $(67\%, 1.1)$, and $(73\%, 1.1)$, respectively. Linear weighted degrees (LDG) achieves $(76\%, 1.13)$, $(80\%, 1.15)$, and $(84\%, 1.14)$, respectively. Now we turn our attention to smaller bur reasonably-sized datasets.

Aggregate View. In Figure 2, we show the distribution of the difference of the fraction of edges cut of our method and that of the best competitor, conditional on that the maximum observed load is at most 1.1. This distribution is derived from the values observed by partitioning each input graph from our set averaged over a range of values of parameter k that consists of values $2, 4, \ldots, 1024$. These results demonstrate that the fraction of edges cut by our method is smaller than that of the best competitor in all cases. Moreover, we observe that the median difference (relative difference) is in the excess of 20% (15%), thus providing appreciable performance gains.

Detailed View. We provide a more detailed view in Figure 3 where we show the empirical distributions of the difference of the percentage of the edges cut of our method and that of the best competitor, conditional on the number of partitions k. These results provide further support that our method yields appreciable benefits for a wide range of parameter k. Several values of k are omitted for visual purposes. Interestingly, we consistently observe that the datapoints where Fennel and LDG have comparable performance, with LDG sometimes performing better by a little, are roadNet-CA, roadNet-PA, and roadNet-TX.

Furthermore, in Table 5, we present the average performance gains conditional on the number of partitions k. These numerical results amount to an average relative reduction of the fraction of edges cut in the excess of 18%. Moreover, the performance gains observed are consistent across different values of parameter k, and are more pronounced for smaller values of k.

Bicriteria. In our presentation of experimental results so far, we focused on the fraction of edges cut by conditioning on the cases where the normalized maximum load was smaller than a fixed threshold. We now provide a closer look at both criteria and their relation. In Figure 4, we consider the difference of the fraction of edges cut vs. the difference of normalized maximum loads of the best competitor and our method. We observe that in all the cases, the differences of normalized maximum loads are well within 10% while the



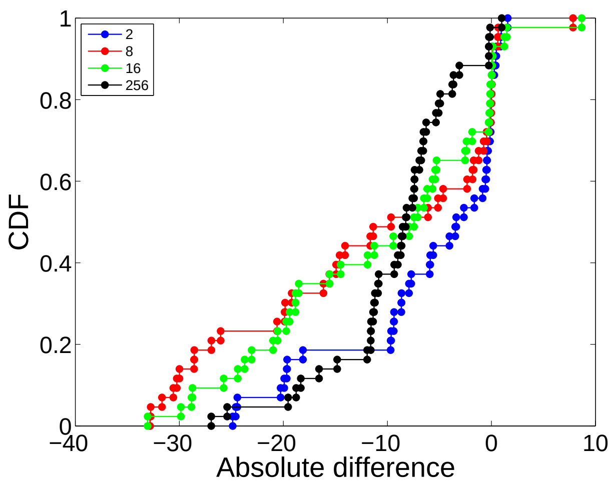


Figure 3: Same as in Figure 2 but conditional on the number of partitions k. Notice that the benefit is more pronounced for the smaller values of k. See Figure 6for a detailed plot, containing CDFs for all values of k we used.

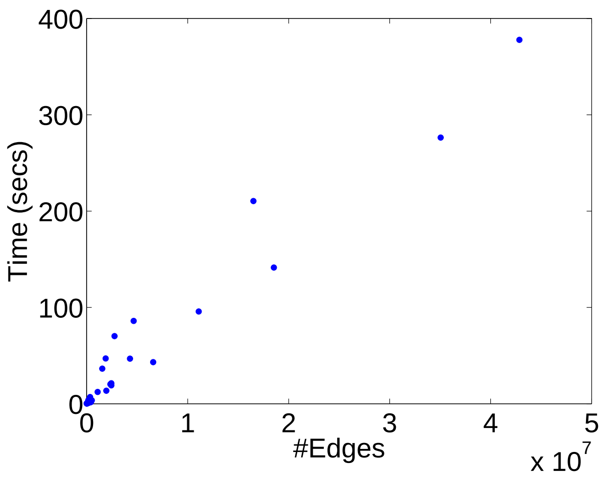


Figure 5: Fennel: time vs. number of edges.

fraction of edges cut by our method is significantly smaller. These results confirm that the observed reduction of the fraction of edges cut by our method is not at the expense of an increased maximum load.

Speed of partitioning. We now turn our attention to the efficiency of our method with respect to the running time to partition a graph. Our graph partitioning algorithm is a one-pass streaming algorithm, which allows for fast graph partitioning. In order to support this claim, in Figure 5, we show the run time it took to partition each graph from our dataset vs. the graph size in terms of the number of edges. We observe that it takes in the order of minutes to partition large graphs of tens of millions of edges. As we also mentioned before, partitioning the largest graph from our dataset collection took about 40 minutes.

6. SYSTEM EVALUATION

As stated in the introduction, one of the key goals of graph partitioning is to speed up large-scale computations. However, the partitioning objective may significantly vary depending on system characteristics. One example are large-scale production data centers. A typical production data center consists of large oversubscribed clusters [24]. In such an environment it is more important to balance the traffic across clusters than the traffic or the amount of computation executed within a cluster. A different example is a typical small-scale cluster rented by a customer from a large cloud provider. There are anecdotal evidences that most customers of Amazon's Elastic MapReduce service rent clusters with up to a few tens of nodes. In such a setting it is important to minimize the network traffic across the nodes but also to balance well the computational load on each node.

Each of these settings may require a different tunning of the partitioning objective. The distinct advantage of Fennel is that it gives a flexibility in choosing the suitable objective. We demonstrate the efficiency and flexibility of Fennel with the typical Elastic MapReduce scenario in mind[1]. We set up a cluster and we vary the number of nodes to 4, 8 and 16 nodes. Each node is equipped with Intel Xeon CPU at 2.27 GHz and 12 GB of main memory. On the cluster ,we run Giraph, a graph processing platform running on the top of Hadoop. We implemented a PageRank algorithm on Giraph and we run it on LiveJournal1 dataset[2].

Since the complexity of PageRank depends on the number of edges and not vertices, we use a version of the Fennel objective (Eq. 1) that balances the number of edges per cluster. In particular, we choose $c_{\mathrm{IN}}(\mathcal{P}) = \sum_{i=1}^{k} e(S_i, S_i)^\gamma$ with $\gamma = 1.5$.

We compare with Hash Partition of vertices, the default partitioner used by Giraph, and LDG. We look at two metrics. The first is the average duration of an iteration of the PageRank algorithm. This metric is directly proportional to the actual running time and incorporates both the processing and the communication time. The second metric is the

[1]Exhaustive evaluation on diverse distributed graph processing platforms is out of scope of this paper.
[2]Twitter data set was too large to fit on a 16-node Giraph cluster.

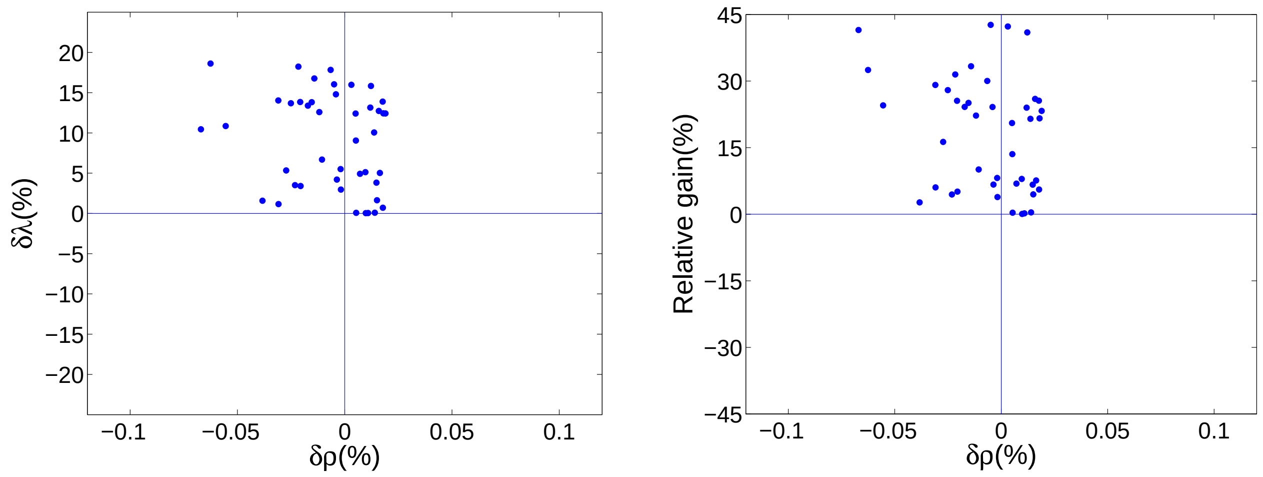


Figure 4: Absolute difference $\delta\lambda$ and Relative gain versus the maximum load imbalance $\delta\rho$.

average number of Megabytes transmitted by a cluster node in each iteration. This metric directly reflects the quality of a cut.

The results are depicted in Table 6. We see that Fennel has the best runtime in all cases. This is because it achieves the best balance between the computation and communication load. Hash Partition takes 25% more time than Fennel and it also has a much higher traffic load. The best competitor (LDG) has much worse runtime. This is because its objective is to balance vertices across nodes, and the PageRank iteration complexity is proportional to the maximal number of edges per node. However, it also achieves the smallest traffic load.

It is worth noting that in the evaluation presented in [54], LDG partitioning outperforms Hash Partition. This is because in they used a much larger number of partitions (100 and 400), which yield more balanced partitions but also a smaller difference in the cross-cut sizes (see Section 5). Also, the evaluation in [54] is performed on a different system (Spark).

7. DISCUSSION

In this section we discuss some of the extensions that can be accomodated by our framework and discuss some details about distributed implementation.

Assymetric edge costs. As discussed in Section 6, in some application scenarios some edges that cross partition boundaries may be more costly than other. For example, this is the case if individual graph partitions are assigned to machines in a data center and these machines are connected with an asymmetric network topology, so that the available network bandwidth varies across different pairs of machines, e.g. intra-rack vs. inter-rack machines in standard data center architectures [24]. Another example are data center network topologies where the number of hops between different pairs of machines vary substantially, e.g. torus topologies [14]. In such scenarios, it may be beneficial to partition a graph by accounting for the aforementioned asymmetries of edge-cut costs. This can be accomodated by appropriately defining the inter-partition cost function in our framework.

Distributed implementation. Our streaming algorithm requires computing marginal value indices that can be computed in a distributed fashion by maintaining local views on a global state. For concretness, let us consider the traditional objective where the inter-partition cost is a linear function of the total number of cut edges and the intra-partition cost is a sum of convex functions of vertex cardinalities of individual partitions. In this case, computing the marginal value indices requires to compute per each vertex arrival: (1) the number of neighbors of given vertex that were already assigned to given cluster of vertices, and (2) the number of vertices that were already assigned per cluster. The former corresponds to a set-intersection query and can be efficiently implemented by standard methods, e.g. using data structures such as minwise hashing [51]. The latter is a simple count tracking problem. Further optimizations could be made by trading accuracy for reduction of communication overhead by updating of the local views at a smaller rate than the rate of vertex arrival. An efficient implementation of the streaming graph partitioning methods in a distributed environment is out of scope of this paper and is left as an interesting direction for future work.

8. CONCLUSION

In this work we provide a new perspective on the well-studied problem of balanced graph partitioning. Specifically, we introduce a unifying framework that subsumes state-of-the-art heuristics [49, 54] for streaming graph partitioning. Furthermore, we show that interpolating between the two state-of-the-art heuristics [49, 54] provides significantly enhanced performance. We evaluate our proposed framework on a large graph collection where we verify consistently over a wide range of cluster (k) values the superiority of our method. On the theory side, we show that a special instance of our objective is the popular modularity measure [21, 46, 5] with an Erdös-Rényi null model. We prove the first rigorous approximation guarantee for this special case of our objective. Finally, we evaluate Fennel in Apache Giraph where we verify its efficiency. In future work, we plan (1) to test other choices of cost functions and, (2) to create an offline community detection algorithm using ideas from our framework.

		Runtime [s]			Communication [MB]		
Data set	# Clusters (k)	Hash	Best competitor	Fennel	Hash	Best competitor	Fennel
LiveJournal1	4	32.27	60.57	**25.49**	321.41	98.3	**196.9**
LiveJournal1	8	17.26	29.57	**15.14**	285.35	74.25	**180.02**
LiveJournal1	16	10.64	16.56	**9.05**	222.28	68.79	**148.67**

Table 6: The average duration of a step and the average amount of MB exchanged per node and per step during the execution of PageRank. Our method has the shortest runtime. All variances are small, except for the variance of the communication cost of the best competitor (where one heavily loaded node has much higher communication cost and runtime than the others).

Acknowledgements

We would like to thank to those who kindly helped us in getting access to some of the graph data that we used for our evaluation, including Nick Craswell, Bodo Billerbeck, Gabriel Kliot, Dushyanth Narayanan, and Isabelle Stanton. We would also like to thank Jingren Zhou for some helpful feedback.

9. REFERENCES

[1] `http://googleblog.blogspot.co.uk/2008/07/we-knew-web-was-big.html`.

[2] `http://techcrunch.com/2012/10/04/facebook-tops\--1-billion-monthly-users-ceo-mark\--zuckerberg-shares-a-personal-note/`.

[3] `http://staffweb.cms.gre.ac.uk/~wc06/partition/`.

[4] J. Abello, M. Resende, and S. Sudarsky. Massive quasi-clique detection. *LATIN 2002: Theoretical Informatics*, pages 598–612, 2002.

[5] G. Agarwal and D. Kempe. Modularity-maximizing graph communities via mathematical programming. *The European Physical Journal B-Condensed Matter and Complex Systems*, 66(3):409–418, 2008.

[6] Farid Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5, 1993.

[7] A. Angel, N. Sarkas, N. Koudas, and D. Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *Proc. VLDB Endow.*, 5(6):574–585, February 2012.

[8] S. Arora, R. Satish, and U. Vazirani. Expander flows, geometric embeddings and graph partitioning. In *STOC*, pages 222–231, 2004.

[9] D. Blandford, G. Blelloch, and I. Kash. An experimental analysis of a compact graph representation. In *ALENEX*, 2004.

[10] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner. On finding graph clusterings with maximum modularity. In *Graph-Theoretic Concepts in Computer Science*, pages 121–132. Springer, 2007.

[11] B. V. Cherkassky, A. V. Goldberg, and T. Radzik. Shortest paths algorithms: theory and experimental evaluation. In *SODA '94*, pages 516–525, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics.

[12] F. R. K. Chung and L. Lu. The average distance in a random graph with given expected degrees. *Internet Mathematics*, 1(1):91–113, 2003.

[13] A. Condon and R. M. Karp. Algorithms for graph partitioning on the planted partition model. In *RANDOM-APPROX*, pages 221–232, 1999.

[14] P. Costa, A. Donnelly, A. Rowstron, and G. O'Shea. Camdoop: exploiting in-network aggregation for big data applications. In *NSDI'12*, pages 3–3, 2012.

[15] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[16] U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM J. Comput.*, 31(4):1090–1118, April 2002.

[17] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *DAC '82*, pages 175–181, 1982.

[18] Santo Fortunato. Community detection in graphs. *CoRR*, abs/0906.0612, 2009.

[19] Alan M. Frieze and Mark Jerrum. Improved approximation algorithms for max k-cut and max bisection. *Algorithmica*, 18(1):67–81, 1997.

[20] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. In *STOC '74*, pages 47–63, 1974.

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

[22] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6), 1995.

[23] J. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *OSDI*, 2012.

[24] A. Greenberg and et al. Vl2: a scalable and flexible data center network. In *SIGCOMM '09*, 2009.

[25] U. Kang, C. E. T., and C. Faloutsos. Pegasus: A peta-scale graph mining system. In *ICDM*, pages 229–238, 2009.

[26] U. Kang, C. E. Tsourakakis, A. P. Appel, C. Faloutsos, and J. Leskovec. Hadi: Mining radii of large graphs. *ACM Trans. Knowl. Discov. Data*, 5(2):8:1–8:24, February 2011.

[27] T. Karagiannis, C. Gkantsidis, D. Narayanan, and A. Rowstron. Hermes: Clustering users in large-scale e-mail services. In *ACM Symposium on Cloud Computing 2010*, 2010.

[28] D. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. *J. ACM*, 45(2):246–265, 1998.

[29] G. Karypis and V. Kumar. Metis-unstructured graph partitioning and sparse matrix ordering system, version 2.0. 1995.

[30] G. Karypis and V. Kumar. Parallel multilevel graph partitioning. In *IPPS*, pages 314–319, 1996.

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

[32] Q. Ke, V. Prabhakaran, Y. Xie, Y. Yu, J. Wu, and J. Yang. Optimizing data partitioning for data-parallel computing. In *HotOS XIII*, May 2011.

[33] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. J.*, 49(2):291–307, February 1970.

[34] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao. Neighborhood based fast graph search in large networks. In *SIGMOD '11*, pages 901–912, 2011.

[35] Subhash Khot. On the unique games conjecture (invited survey). In *IEEE Conference on Computational Complexity*, pages 99–121, 2010.

[36] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? *SIAM J. Comput.*, 37(1):319–357, April 2007.

[37] A. Konstantin and H. Räcke. Balanced graph partitioning. In *SPAA '04*, pages 120–124, 2004.

[38] R. Krauthgamer, J. (S.) Naor, and R. Schwartz. Partitioning graphs into balanced components. In *SODA '09*, pages 942–949, 2009.

[39] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 591–600, New York, NY, USA, 2010. ACM.

[40] J. Leskovec, K.J. Lang, A. Dasgupta, and M. W. Mahoney. Statistical properties of community structure in large social and information networks. In *Proceeding of the 17th international conference on World Wide Web*, pages 695–704. ACM, 2008.

[41] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Graphlab: A new framework for parallel machine learning. In *UAI*, pages 340–349, 2010.

[42] G. Malewicz, M. H. Austern, A.J.C Bik, J.C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD '10*, pages 135–146, 2010.

[43] Kolountzakis M. N., G. L. Miller, R. Peng, and Tsourakakis C. E. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Internet Mathematics*, 8(1-2):161–185, 2012.

[44] M. Newman. *Networks: An Introduction.* Oxford University Press, 2010.

[45] M. E. J. Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

[46] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[47] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[48] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: bringing order to the web. Technical report, Stanford InfoLab, 1999.

[49] V. Prabhakaran and et al. Managing large graphs on multi-cores with graph awareness. In *USENIX ATC'12*, 2012.

[50] J. Pujol and et al. The little engine(s) that could: Scaling online social networks. In *ACM SIGCOMM 2010*, 2010.

[51] V. Satuluri, S. Parthasarathy, and Y. Ruan. Local graph sparsification for scalable clustering. In *SIGMOD '11*, pages 721–732, 2011.

[52] K. Schloegel, G. Karypis, and V. Kumar. Parallel multilevel algorithms for multi-constraint graph partitioning (distinguished paper). In *Proceedings from the 6th International Euro-Par Conference on Parallel Processing*, Euro-Par '00, pages 296–310, 2000.

[53] K. Schloegel, G. Karypis, and V. Kumar. Parallel static and dynamic multi-constraint graph partitioning. *Concurrency and Computation: Practice and Experience*, 14(3):219 – 240, 2002.

[54] I. Stanton and G. Kliot. Streaming graph partitioning for large distributed graphs. In *ACM KDD*, pages 1222–1230, 2012.

[55] Z. Sun and et al. Efficient subgraph matching on billion node graphs. *Proc. VLDB Endow.*, 5(9):788–799, May 2012.

[56] C. E. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic. Fennel: Streaming graph partitioning for massive distributed graphs. Microsoft Technical Report MSR-TR-2012-213, November 2012.

[57] Charalampos E. Tsourakakis. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *ICDM*, pages 608–617, 2008.

[58] Charalampos E. Tsourakakis, Mihail N. Kolountzakis, and Gary L. Miller. Triangle sparsifiers. *J. Graph Algorithms Appl.*, 15(6):703–726, 2011.

[59] C. Walshaw, M. G. Everett, and M. Cross. Parallel dynamic graph partitioning for adaptive unstructured meshes. *J. Parallel Distrib. Comput.*, 47(2):102–108, December 1997.

[60] J. Zhou, N. Bruno, and W. Lin. Advanced partitioning techniques for massively distributed computation. In *SIGMOD '12*, pages 13–24, 2012.

APPENDIX

A. APPROXIMATION GUARANTEES

We derive hard approximation guarantees for the optimal k graph partition problem. We shall focus on the case of quadratic cost function $c(x) = \alpha\binom{x}{2}$. This is a special case of interest in view of its underlying graph combinatorial meaning and relation the concept of modularity. We will derive approximation algorithms using the standard concept of an approximation guarantee.

Definition 1. *Let $\mathcal{P}^*$ be an optimal partition for the k graph partitioning problem. A partition $\mathcal{P}$ is said to be a ρ-approximation, if the following holds:*

$$g(\mathcal{P}) \geq \rho g(\mathcal{P}^*)$$

An algorithm that guarantees to output a ρ-approximation *partitioning is said to be a ρ-approximation algorithm.*

An approximation algorithm for a maximization problem is meaningful as a notion if the optimum solution is positive. However, in our setting our function g does not results as it can easily be seen in a non-negative optimum. For instance if $G = E_n$ the empty graph on n vertices, any partition of G in k parts results in a negative objective (except when $k = n$ where the objective becomes 0). Therefore, we need to shift our objective in order to come up with approximation algorithms.

In the remainder of this section, we shall first define a shifted objective function, which ensures that the optimal value is positive. Then, we will go on with showing that uniform random partition of vertices is an $\Omega(1/k)$ approximation algorithm. Recall that uniform random partitioning essentially corresponds to Hash Partition of vertices, a method which is commonly used in current practice. We will then show that one can do better by using an SDP-based rounding algorithm, which guarantees an $\Omega(\log(k)/k)$ approximation.

A.1 Shifted Objective

We define the following shifted objective.

DEFINITION 2. $k \geq 1$. *Also let $\mathcal{P}^* = \{S_1^*, \ldots, S_k^*\}$ be a partition of the vertex set V. We define the function $\tilde{g}$ as*

$$\tilde{g}(\mathcal{P}) = \sum_{i=1}^{k} |e(S_i, S_i)| + \alpha \left(\binom{n}{2} - \sum_{i=1}^{k} \binom{|S_i|}{2} \right).$$

We can interpret the shifted objective as the expected number of edges for given partition $\mathcal{P}$ and given that there are $e[S_i]$ edges within partition S_i, under a null hypothesis that the underlying graph is random Erdös-Rényi graph with parameter α.

LEMMA 1. *For any partition $\mathcal{P}$, $\tilde{g}(\mathcal{P}) \geq 0$.*

PROOF. The proof follows directly from the fact that for any positive-valued $s_1, s_2, \ldots, s_k$ such that $\sum_{i=1}^k s_i = n$, the following holds $n^2 \geq s_1^2 + \cdots + s_k^2$. $\square$

The approximation guarantees that hold for the shifted objective function have the following meaning for the original objective function. Suppose that $\mathcal{P}$ is a ρ-approximation with respect to the shifted-objective k graph partitioning problem, i.e. $\tilde{g}(\mathcal{P}) \geq \rho \tilde{g}(\mathcal{P}^*)$. Then, it holds

$$g(\mathcal{P}) \geq \rho g(\mathcal{P}^*) - (1-\rho)\alpha \binom{n}{2}.$$

Notice that this condition is equivalent to

$$g(\mathcal{P}) - g(\mathcal{P}^*) \geq -(1-\rho)[g(\mathcal{P}^*) + \alpha \binom{n}{2}].$$

This approximation guarantee is near to a ρ-approximation if the parameter α is small enough so that the term $g(\mathcal{P}^*)$ dominates the term $\alpha\binom{n}{2}$. For example, for an input graph that consists of k cliques, we have that $g(\mathcal{P}^*) \geq \alpha\binom{n}{2}$ corresponds to $\frac{\alpha}{1-\alpha} \leq \frac{1}{k}\frac{n+k}{n+1}$, from which we conclude that it suffices that $\alpha \leq \frac{1}{k+1}$.

A.2 Random Partition

Suppose each vertex is assigned to one of k partitions uniformly at random. This simple graph partition is a faithful approximation of hash partitioning of vertices that is commonly used in practice. In expectation, each of the k clusters will have $\frac{n}{k}$ vertices. Let $S_1, \ldots, S_k$ be the resulting k clusters. How well does this simple algorithm perform with respect to our objective? Let $\mathcal{P}^*$ be an optimal partition for the optimal k graph partition problem, i.e. $g(\mathcal{P}^*) \geq g(\mathcal{P})$, for every partition $\mathcal{P}$ of the vertex set V into k partitions. Notice that $\mathcal{P}^*$ is also an optimal partition for the optimal k graph partition problem with shifted objective function. Now, note that an edge $e = (u, v)$ has probability $\frac{1}{k}$ that both its endpoints belong to the same cluster. By the linearity of expectation, we obtain by simple calculations:

$$\begin{aligned} \mathbb{E}\left[\tilde{g}(S_1, \ldots, S_k)\right] &= \frac{m}{k} + \alpha\frac{k-1}{k}\binom{n}{2} \\ &\geq \frac{1}{k}\left(m + \alpha\binom{n}{2}\right) \\ &\geq \frac{1}{k}\tilde{g}(\mathcal{P}^*) \end{aligned}$$

where last inequality comes from the simple upper bound $\tilde{g}(\mathcal{P}) \leq m + \alpha\binom{n}{2}$ for any partition $\mathcal{P}$.

A.3 An SDP Rounding Algorithm

We define a vector variable x_i for each vertex $i \in V$ and we allow x_i to be one of the unit vectors $e_1, \ldots, e_k$, where e_j is a vector of dimension n with all elements equal to zero except for the j-th element equal to 1.

$$\boxed{\begin{array}{rl} \text{maximize} & \sum_{e=(i,j)} x_i x_j + \alpha \sum_{i<j} \left(1 - x_i x_j\right) \\ \text{subject to} & x_i \in \{e_1, \ldots, e_k\},\ \forall i \in \{1, \ldots, n\} \end{array}} \qquad (2)$$

We obtain the following semidefinite programming relaxation:

$$\boxed{\begin{array}{rl} \text{maximize} & \sum_{e=(i,j)} y_{ij} + \alpha \sum_{i<j} \left(1 - y_{ij}\right) \\ \text{subject to} & y_{ii} = 1,\ \forall i \in \{1, \ldots, n\} \\ & y_{ij} \geq 0,\ \forall i \neq j \\ & Y \succeq 0,\ Y \text{ symmetric} \end{array}} \qquad (3)$$

The above SDP can be solved within an additive error of δ of the optimum in time polynomial in the size of the input and $\log\left(\frac{1}{\delta}\right)$ by interior point algorithms or the ellipsoid method [6]. In what follows, we refer to the optimal value of the integer program as $\mathrm{OPT}_{\mathrm{IP}}$ and of the semidefinite program as $\mathrm{OPT}_{\mathrm{SDP}}$. Our algorithm is the following:

SDP-Relax

- *Relaxation:* Solve the semidefinite program [6] and compute a Cholesky decomposition of Y. Let $v_0, v_1, \ldots, v_n$ be the resulting vectors.
- *Randomized Rounding:* Randomly choose $t = \lceil \log k \rceil$ unit length vectors $r_i \in \mathbb{R}^k, i = 1, \ldots, t$. These t random vectors define $2^t = k$ possible regions in which the vectors v_i can fall: one region for each distinct possibility of whether $r_j v_i \geq 0$ or $r_j v_i < 0$. Define a cluster by adding all vertices whose vector v_i fall in a given region.

We now state our main theoretical result, which is for the the k graph partition problem with the objective function given by Definition 2.

THEOREM 2. *Algorithm SDP-Relax is a* $\Omega(\frac{\log k}{k})$ *approximation algorithm for the optimal* k *graph partition problem.*

PROOF. Let C_k be the score of the partition produced by our randomized rounding. Define $A_{i,j}$ to be the event that vertices i and j are assigned to the same partition. Then,

$$\mathbb{E}\left[C_k\right] = \sum_{e=(i,j)} \mathbf{Pr}\left[A_{i,j}\right] + \alpha \sum_{i<j} \Big(1 - \mathbf{Pr}\left[A_{i,j}\right]\Big)$$

As in Goemans-Williamson [22], given a random hyperplane with normal vector r that goes through the origin, the probability of $\text{sgn}(v_i^T r) = \text{sgn}(v_j^T r)$, i.e., i and j fall on the same side of the hyperplane, is $1 - \frac{\arccos(v_i^T v_j)}{\pi}$. Since we have t independent hyperplanes

$$\mathbf{Pr}\left[A_{i,j}\right] = \left(1 - \frac{\arccos(v_i^T v_j)}{\pi}\right)^t.$$

Let us define, for $t \geq 1$,

$$f_1(\theta) = \frac{\left(1 - \frac{\theta}{\pi}\right)^t}{\cos(\theta)},\ \theta \in [0, \frac{\pi}{2})$$

and

$$\rho_1 = \min_{0 \leq \theta < \frac{\pi}{2}} f_1(\theta).$$

Similarly, define for $t \geq 1$,

$$f_2(\theta) = \frac{1 - \left(1 - \frac{\theta}{\pi}\right)^t}{1 - \cos(\theta)},\ \theta \in [0, \frac{\pi}{2})$$

and

$$\rho_2 = \min_{0 \leq \theta < \frac{\pi}{2}} f_2(\theta).$$

We wish to find a ρ such that $\mathbb{E}\left[C_k\right] \geq \rho \text{OPT}_{\text{SDP}}$. Since, $\text{OPT}_{\text{SDP}} \geq \text{OPT}_{\text{IP}}$, this would then imply $\mathbb{E}\left[C_k\right] \geq \rho \text{OPT}_{\text{IP}}$. We note that

$$f_1'(\theta) = \frac{-\frac{t}{\pi}\left(1 - \frac{\theta}{\pi}\right)^{t-1} \cos(\theta) + \left(1 - \frac{\theta}{\pi}\right)^t \sin(\theta)}{\cos^2(\theta)}.$$

It follows that $f_1'(\theta(t)) = 0$ is equivalent to

$$t = (\pi - \theta(t)) \tan(\theta(t)).$$

Notice that the following two hold

$$\lim_{t \to \infty} \theta(t) = \frac{\pi}{2}$$

and

$$\frac{\pi}{2} \tan(\theta(t)) \leq t \leq \pi \tan(\theta(t)). \tag{4}$$

We next show that $f_1(\theta(t)) \geq \frac{1}{\pi} t 2^{-t}$, by the the following series of relations

$$\begin{aligned} f_1(\theta(t)) &= \frac{\left(1 - \frac{\theta(t)}{\pi}\right)^t}{\cos(\theta(t))} = \sqrt{1 + \tan^2(\theta(t))} \left(1 - \frac{\theta(t)}{\pi}\right)^t \\ &\geq \sqrt{1 + \tan^2(\theta(t))} 2^{-t} \geq \sqrt{1 + \left(\frac{t}{\pi}\right)^2} 2^{-t} \\ &\geq \frac{1}{\pi} t 2^{-t} \end{aligned}$$

where the second equality is by the fact $\cos(\theta) = \frac{1}{\sqrt{1 + \tan^2(\theta)}}$, the first inequality is by the fact $\theta(t) \leq \frac{\pi}{2}$, and the second inequality is by (4).

Thus, for $t = \log_2(k)$, we conclude

$$\rho_1 \geq \frac{1}{\pi \log(2)} \frac{\log(k)}{k}.$$

Now, we show that $\rho_2 = \frac{1}{2}$. First we show that for any $t \geq 1$, $f_2(\theta) \geq 1/2$. To this end, we note

$$\frac{1 - \left(1 - \frac{\theta}{\pi}\right)^t}{1 - \cos(\theta)} \geq \frac{1}{2} \Leftrightarrow \frac{1}{2}\Big(1 + \cos(\theta)\Big) \geq \left(1 - \frac{\theta}{\pi}\right)^t.$$

Notice that for all $\theta \in [0, \pi/2)$, if $t_1 \geq t_2 \geq 1$, then $\Big(1 - \frac{\theta}{\pi}\Big)^{t_1} \leq \Big(1 - \frac{\theta}{\pi}\Big)^{t_2}$. Hence, it suffices $\frac{1}{2}\Big(1 + \cos(\theta)\Big) \geq \big(1 - \frac{\theta}{\pi}\big)$. With the use of simple calculus, the latter is true and is also tight for $\theta = 0$ and $\theta = \pi/2$. It is worth observing the opposite trend of the values of ρ_1 and ρ_2. The reason is that $\mathbf{Pr}\left[A_{i,j}\right]$ drops as we use more hyperplanes and, of course, $1 - \mathbf{Pr}\left[A_{i,j}\right]$ grows.

We are now in a position to establish the following lower bound on the expected score of our randomized rounding procedure. Let $\theta_{i,j} = \arccos(v_i^T v_j)$.

$$\begin{aligned} \mathbb{E}\left[C_k\right] &= \sum_{e=(i,j)} \mathbf{Pr}\left[A_{i,j}\right] + \alpha \sum_{i<j} \Big(1 - \mathbf{Pr}\left[A_{i,j}\right]\Big) \\ &= \sum_{e=(i,j)} \left(1 - \frac{\theta_{i,j}}{\pi}\right)^t + \alpha \sum_{i<j} \left(1 - \left(1 - \frac{\theta_{i,j}}{\pi}\right)^t\right) \\ &\geq \sum_{e=(i,j)} \rho_1 \cos(\theta_{i,j}) + \alpha \sum_{i<j} \rho_2 \Big(1 - \cos(\theta_{i,j})\Big) \\ &\geq \min\{\rho_1, \rho_2\} \text{OPT}_{\text{SDP}} \\ &\geq \min\{\rho_1, \rho_2\} \text{OPT}_{\text{IP}} \end{aligned}$$

It suffices to set $\rho = \min\{\rho_1, \rho_2\}$. The above analysis shows that our algorithm is a ρ-approximation algorithm, where $\rho = \Omega(\frac{\log(k)}{k})$. $\square$

A.4 An Alternative Approach

One may ask whether the relaxation of Frieze and Jerrum [19], Karger, Motwani and Sudan [28] can improve significantly the approximation factor. Before we go into further details, we notice that the main "bottleneck" in our approximation is the probability of two vertices being in the same cluster, as k grows. The probability that i, j are in the same cluster in our rounding is $p(\theta)$ where

$$p(\theta) = \left(1 - \tfrac{\theta}{\pi}\right)^{\log k}.$$

Suppose $\theta = \frac{\pi}{2}(1-\epsilon)$. Then,

$$\begin{aligned} p(\theta) &= \left(1 - \tfrac{\theta}{\pi}\right)^{\log k} = \left(\frac{1+\epsilon}{2}\right)^{\log k} \\ &= \frac{1}{k} + \epsilon\frac{\log k}{k} + O(\epsilon^2). \end{aligned}$$

As we see from Lemma 5 in [19], for this θ, the asymptotic expression of $N_k(\cos(\theta))$, which is equal to ours $p(\theta)$ but with their rounding scheme, matches ours:

$$\begin{aligned} N_k(\cos(\theta)) &\approx \frac{1}{k} + \frac{2\log k}{k}\cos\left(\frac{\pi}{2}(1-\epsilon)\right) \\ &= \frac{1}{k} + \pi\epsilon\frac{\log k}{k} + O(\epsilon^2). \end{aligned}$$

This suggests that our approximation factor may be the best optimal assuming the unique games conjecture [36, 35].

B. DETAILED DISTRIBUTION FUNCTIONS

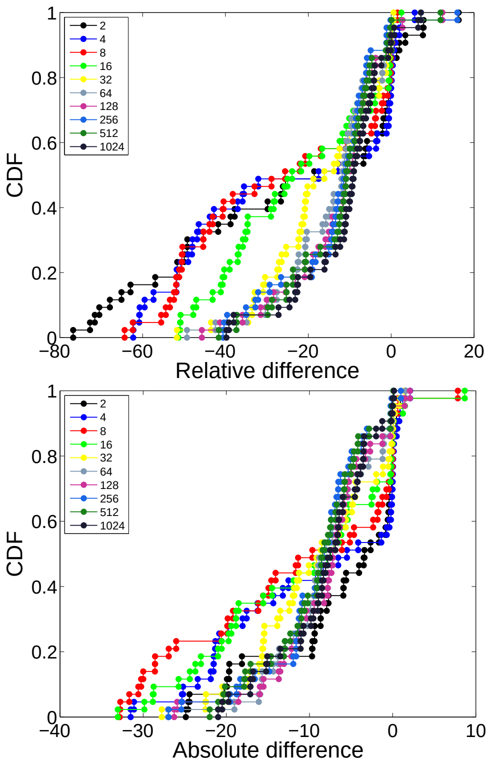


Figure 6: Same as in Figure 3 but for all values of k used.