FENNEL: Streaming Graph Partitioning for Massive Scale Graphs

Charalampos E. Tsourakakis[1] Christos Gkantsidis [2] Bozidar Radunovic[2] Milan Vojnovic[2]
[1] Aalto University, Espoo, Finland
[2]Microsoft Research, Cambridge, UK

ABSTRACT

Balanced graph partitioning in the streaming setting is a key problem to enable scalable and efficient computations on massive graph data such as web graphs, knowledge graphs, and graphs arising in the context of online social networks. Two families of heuristics for graph partitioning in the streaming setting are in wide use: place the newly arrived vertex in the cluster with the largest number of neighbors or in the cluster with the least number of non-neighbors.

In this work, we introduce a framework which unifies the two seemingly orthogonal heuristics and allows us to quantify the interpolation between them. More generally, the framework enables a well principled design of scalable, streaming graph partitioning algorithms that are amenable to distributed implementations. We derive a novel one-pass, streaming graph partitioning algorithm and show that it yields significant performance improvements over previous approaches using an extensive 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 in many cases comparable to the *de-facto* standard offline software METIS and in some cases even superiror. 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. We also demonstrate the performance gains by using our graph partitioner while solving standard PageRank computation in a graph processing platform with respect to the communication cost and runtime.

Categories and Subject Descriptors

H.2.4 [**Database Systems**]: [Distributed databases]; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*; G.2.2 [**Mathematics of Computing**]: [Graph Algorithms]

WSDM'14, February 24–28, 2014, New York, NY, USA.

http://dx.doi.org/10.1145/2556195.2556213 .

Keywords

Streaming; Balanced Graph partitioning; Distributed Computing

1. INTRODUCTION

The scale of graph data that needs to be processed in the context of online services is massive. For example, the Web graph amounts to at least one trillion of links, Facebook recently reported more than 1 billion of users and 140 billion of friend connections, and, in 2009, Twitter reported more than 40 million of users and about 1.5 billion of social relations [21]. A standard approach for scalable computation on massive scale input graph data is to partition an input graph into smaller partitions and then use a large distributed system to process them. The partition sizes 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 between different partitions. Many popular graph processing platforms such as Pregel [23] that builds on MapReduce [7], and its open source cousin Apache Giraph, PEGASUS [13] and GraphLab [22] 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 is efficient for balancing the number of vertices over different clusters, but ignores entirely the graph structure. 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 [4], 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 [19] 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., [23, 27]. It is worth emphasizing that the balanced graph partitioning problem appears in various guises in numerous other domains, e.g., see [16].

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

	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 normalized maximum load ρ for FENNEL, the previously best-known heuristic (linear weighted degrees [29]) 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.

created upon post-processing datasets such as Twitter posts are also dynamic, see for instance [2]. 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* [29]. 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., [20], and heuristics that are used in practice [16, 26, 29]. 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 general framework for graph partitioning that relaxes the hard cardinality constraints on the number of vertices in a cluster [3, 20]. Our formulation provides a unifying framework that subsumes two of the most popular heuristics used for streaming balanced graph partitioning: the folklore heuristic of [26] which places a vertex to the cluster with the fewest non-neighbors, and the degree-based heuristic of [29], which serves as the current state-of-the-art method with respect to performance.

• Our framework allows us to define formally the notion of interpolation between between the non-neighbors heuristic [26] and the neighbors heuristic [29]. This provides improved performance for the balanced partitioning problem in the streaming setting. Moreover, for a special case of our framework, we provide a $O(\frac{\log(k)}{k})$ approximation algorithm, where k is the number of clusters.

• We evaluate our proposed streaming graph partitioning method, FENNEL, on a wide range of graph datasets, both real-world and synthetic graphs, and show that it produces high quality graph partitions. For example, Table 1 shows the performance of FENNEL versus the best previously-known heuristic, which is the linear weighted degrees [29], and the baseline Hash Partition of vertices. We observe that FENNEL achieves, simultaneously, significantly smaller fraction of edges cut and balanced cluster sizes.

• We compare FENNEL to Hash Partition, the default partitioner of various major large-scale graph processing platforms. Specifically, we demonstrate significant performance gains with respect to the communication cost and the runtime while running Pagerank over a distributed graph in the graph processing platform Apache Giraph.

Structure of the Paper. The remainder of the paper is organized as follows. Section 2 discusses the related work. Section 3 introduces our graph partitioning framework and presents our main theoretical result. Section 4 presents a scalable, streaming algorithm. Section 5 evaluates our method versus the state-of-the-art work on an extensive set of real-world and synthetic graphs. Section 6 illustrates that FENNEL is a choice for achieving fast and high-quality partitioning in a real system, by performing a baseline comparison to Hash Partition in Apache Giraph. Finally, Section 7 concludes the paper.

2. RELATED WORK

Balanced graph partitioning is an NP-hard problem [10]. It is a fundamental problem in every parallel and distributed application since data placement typically affects significantly the execution efficiency of jobs [17]. 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 $\nu\frac{n}{k}$, for some given positive integer k and $\nu \geq 1$. The case $k = 2, \nu = 1$ is equivalent to the *minimum bisection problem*, an NP-hard problem [10]. Several approximation algorithms and heuristics exist for this problem, see, e.g., [9] and [18] respectively. When $\nu = 1 + \epsilon$ for any desired but fixed $\epsilon > 0$ there exists a $O(\epsilon^{-2}\log^{1.5} n)$ approximation algorithm [19]. When $\nu = 2$ there exists an $O(\sqrt{\log k \log n})$ approximation algorithm based on semidefinite programming (SDP) [20]. Due to the practical importance of k-paritioning there exist several heuristics, among which METIS [15] and its parallel version [28] stands out for its good performance. For this reason, METIS is used in many existing systems, e.g. [14]. Recently, it was shown that label propagation [31] is efficient and effective. An extensive summary of existing heuristics can be found in [1].

Online graph partitioning was introduced by Stanton and Kliot [29]. The online 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 expensive in a distributed system. Currently the most advanced online partitioning algorithm is by Stanton and Kliot [29], against which we extensively compare our approach.

3. PROPOSED FRAMEWORK

Notation. Let $G(V, E)$ be a simple undirected graph, where $|V| = n, |E| = m$. For a subset of vertices $S \subseteq V$, let $e(S, S)$ be the set of edges with both end-vertices in S, and let $e(S, V \setminus S)$ be the set of edges across the cut $(S, V \setminus S)$. 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 *vertex partition* $\mathcal{P} = (S_1, \ldots, S_k)$ to be a family of pairwise disjoint vertex sets whose union is V. We refer to each $S_i, i = 1, \ldots, k$, as a *cluster* of vertices. Let $\partial e(\mathcal{P})$ to be the set of edges that cross partition boundaries, i.e. $\partial e(\mathcal{P}) = \cup_{i=1}^{k} e(S_i, V \setminus S_i)$. Finally, we refer to $|\partial e(\mathcal{P})|$ as the *edge-cut size*.

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

The size of individual clusters. We denote with $\sigma(S_i)$ the size of the cluster of vertices S_i, where σ is a mapping to the set of positive real numbers. Special instances of interest are (a) *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. $|e(S_i, S_i)| + |e(S_i, V \setminus S_i)|$, (b) *interior-edge cardinality* where the size of cluster i is proportional to the number of internal edges $|e(S_i, S_i)|$, and (c) *vertex cardinality* where the size of partition i is proportional to the total number of vertices $|S_i|$. The edge cardinality is a measure of cluster size that is of interest in some applications, e.g. iterative computations on input graph data where the computational complexity within a cluster is linear in the number of edges with at least one vertex in the given cluster. The vertex cardinality is a standard measure of the size of a cluster and for some graphs, e.g., of bounded degree, may serve as a proxy for the edge cardinality.

The global objective function. We define a global objective function that consists of two elements: (1) the inter-partition cost $c_{\text{OUT}} : \mathbb{N}^k \to \mathbb{R}_+$ and (2) the intra-partition cost $c_{\text{IN}} : \mathbb{N}^k \to \mathbb{R}_+$. 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 $\mathcal{P} = (S_1, S_2, \ldots, S_k)$, we define the global cost function as

$$f(\mathcal{P}) = c_{\text{OUT}}(|e(S_1, V \setminus S_1)|, \ldots, |e(S_k, V \setminus S_k)|) + c_{\text{IN}}(\sigma(S_1), \ldots, \sigma(S_k)).$$

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 instance 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 $c_{\text{IN}}(\sigma(S_1), \ldots, \sigma(S_k)) = \sum_{i=1}^{k} c(\sigma(S_i))$, 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 $\mathcal{P}^* = \{S_1^*, \ldots, S_k^*\}$ 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 [29].

- 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. Finally, it is worth mentioning that even if we focus on vertex balanced partitions in Section 5, FENNEL also works for edge balanced parititions as well, see [29].

Application to Classical Balanced Partitioning. Classical balanced graph partitioning problem is the most common special case of our framework, in which the inter-partition cost is equal to the total number of edges cut, and the intra-partition cost is defined in terms of the vertex cardinalities. We will next explain the intuition behind our framework in this important case, and how to derive appropriate cost functions.

The starting point in the literature, e.g., [19, 20], is to impose hard cardinality constraints, namely $|S_i^*| \leq \nu \frac{n}{k}$ for some small constant ν, $i = 1, \ldots, k$. This set of constaints makes the problem significantly hard. Currently, the state-of-the-art work depends on the Arora-Rao-Vazirani barrier [3] which results in a $O(\sqrt{\log n})$ approximation factor. The typical formulation is the following:

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

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 $i = 1, 2, \ldots, k$. 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. Our graph partitioning method is based on solving the following optimization problem:

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

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 choosen 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 focus on 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 sub-optimal 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 which makes a connection with the concept of graph modularity [24]. For a graph $G = (V, E)$ and $S \subseteq V$, we define the function $h : 2^V \to \mathbb{R}$ as:

$$h(S) = |e(S, S)| - c(|S|)$$

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, 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

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

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 special 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 $p = 2\alpha$. In this case, each summation element corresponds to the difference between the realized and the expected number of edges within each cluster under the null-hypothesis that the graph is an Erdös-Rényi random graph with probability p. This is intimately related to the concepts of graph modularity [11] and quasi-cliques [30].

Approximation guarantees. For the special case of $\gamma = 2$ we can derive an approximation algorithm with provable guarantees. We design a semi-definite programming algorithm for a shifted version of our objective. Specifically, the objective is shifted by $\alpha\binom{n}{2}$ to ensure that the optimal objective value is non-negative, a necessary condition for designing multiplicative approximation algorithms. Given that a typical real-world graph is sparse, e.g., $m = O(n)$ or $m = O(n\text{polylog}(n))$, the objective is dominated by the main $O(n^2)$ term. Therefore, the shifted objective guarantees performance at least as good as Hash Partitioning due to the main term and then optimizes a second order term which is $O(m) = o(n^2)$. Our approximation guarantee provides an imrovement of $O(\log k)$ over hash partitioning, see Theorem 1.

THEOREM 1. *There exists a polynomial time algorithm which provides an $\Omega(\frac{\log k}{k})$-approximation guarantee for the shifted by $\alpha\binom{n}{2}$ Optimal k-Graph Partitioning.*

The proof and algorithm along with the discussion about when the incurred additive error due to shifting does not render the algorithm useless will appear in an extended version of this work. An interesting research direction is to pursue additive rather than multiplicative guarantees for maximizing $g(\mathcal{P})$.

4. ONE-PASS STREAMING ALGORITHM

General design. We derive a streaming algorithm by using a greedy assignment of vertices to clusters 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 $\mathcal{P} = (S_1, S_2, \ldots, S_k)$, a vertex v is assigned to partition i such that

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

Defining $\delta g(v, S_i) = g(S_1, \ldots, S_i \cup \{v\}, \ldots, S_j, \ldots, S_k) - g(S_1, \ldots, S_i, \ldots, S_j, \ldots, S_k)$, the above greedy vertex assignment naturally suggests the following streaming 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 $j \in [k]$

Application to Classical Balanced Partitioning. We treat the important special case of balancing edge-cuts and vertex cardinality in more detail, since we evaluate it experimentally in Section 5. 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 terms 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 [29], 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* [26]. 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 [29] and [26]. The overall complexity of our algorithm is $O(n + m)$.

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 our experimental findings. Specifically, Section 5.1 describes the experimental setup. Sections 5.2 and 5.3 present our findings for synthetic and real-world graphs 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 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 [6]. 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 [5]. 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

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

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

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.

	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.

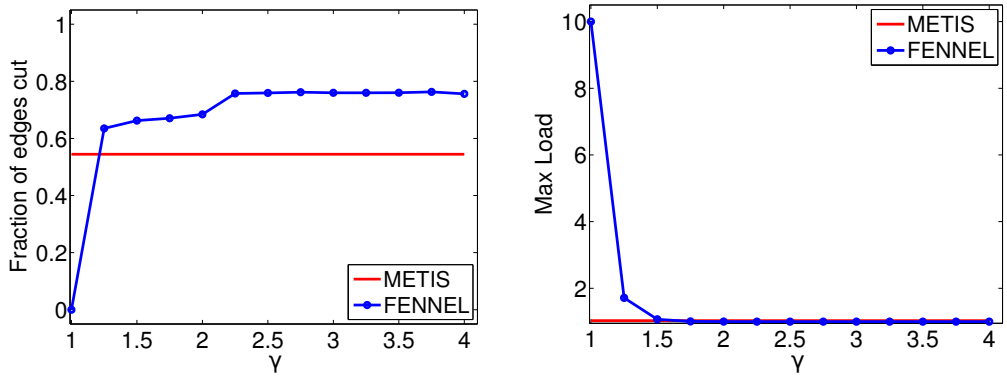

Figure 1: Fraction of edges cut λ and maximum load normalized ρ as a function of γ, ranging from 1 to 4 with a step of 0.25, over five randomly generated power law graphs with slope 2.5. The straight lines show the performance of METIS.

	BFS		Random	
Method	λ	ρ	λ	ρ
H	96.9%	1.01	96.9%	1.01
B [29]	97.3%	1.00	96.8%	1.00
DG [29]	0%	32	43%	1.48
LDG [29]	34%	1.01	40%	1.00
EDG [29]	39%	1.04	48%	1.01
T [29]	61%	2.11	78%	1.01
LT [29]	63%	1.23	78%	1.10
ET [29]	64%	1.05	79%	1.01
NN [26]	69%	1.00	55%	1.03
Fennel	14%	1.10	14%	1.02
METIS [16]	8%	1.00	8%	1.02

Table 3: Performance of various existing methods on amazon0312 ($k = 32$).

		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 [16] averaged over 5 random graphs generated according to the HP(5 000, k, 0.8, 0.5) model.

In Section 5.2, we use the greedy vertex assignment without any threshold. Given that we are able to control the 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 $\nu = 1.1$. Despite the fact that one can choose γ based on the how skewed the degree sequence is in order to obtain superior performance, see Section 5.2, we use the same value γ for all graphs. This choice of γ is reasonable in the light of our findings in Section 5.2. The choice of α is based on the related discussion in Section 3. Finally, $\nu = 1.1$ is a small enough load balancing factor for any practical purposes.

For competitors, we consider the state-of-the-art heuristics. Specifically, in our evaluation we consider the following heuristics from [29], which we briefly describe here for completeness. Let v be the newly arrived vertex, then place v to a cluster S_i with

- Balanced (B): minimum cardinality $|S_i|$.
- Hash partitioning (H): S_i choosen uniformly at random.
- Deterministic Greedy (DG): maximum $|N(v) \cap S_i|$.
- Linear Weighted Deterministic Greedy (LDG): maximum $|N(v) \cap S_i|(1 - |S_i|/(n/k))$.
- Exponentially Weighted Deterministic Greedy (EDG): maximum $|N(v) \cap S_i|(1 - \exp(|S_i| - n/k))$.
- Triangles (T): maximum $t_{S_i}(v)$.
- Linear Weighted Triangles (LT): maximum $t_{S_i}(v)(1 - |S_i|/(n/k))$.
- Exponentially Weighted Triangles (ET): maximum $t_{S_i}(v)(1 - \exp|S_i| - n/k))$.
- Non-Neighbors (NN): minumum $|S_i \setminus N(v)|$.

In accordance with [29], 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 the following, the best competitor is LDG unless otherwise mentioned.

5.2 Synthetic Datasets

In this section we present result that support the following main findings: (a) The value $\gamma = \frac{3}{2}$ achieves good performance for all graphs considered. (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 $\gamma < \frac{3}{2}$

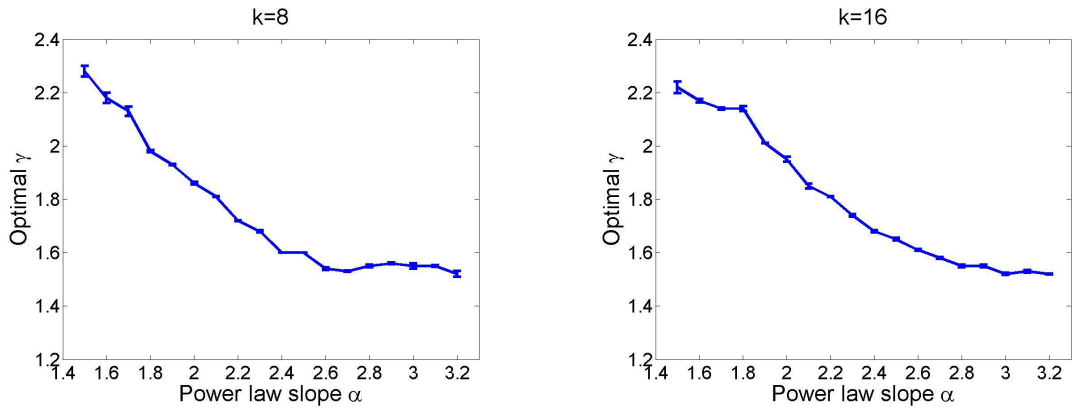

Figure 2: Figure plots the average optimal value γ^* by ranging the power-law exponent in the interval $[1.5, 3.2]$ using a step of 0.1 over twenty generated graphs (for each exponent) that results in the smallest possible fraction of edges cut λ conditioning on a maximum normalized load $\rho = 1.2$ versus the power-law exponent of the degree sequence. Error bars indicate the variance around the average optimal value γ^*. Strong concentration around the average is observed. (a) $k = 8$ (b) $k = 16$.

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. (d) It is possible to compute easy graph statistics such as the power-law exponent of the degree sequence of a power-law graph to fine-tune the choice of γ.

Hidden Partition: We report averages over five randomly generated graphs according to the model $\mathsf{HP}(5000, k, 0.8, 0.5)$ 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 observed consistently the following behavior: for $\gamma = 1$, $\lambda = 0$ and $\rho = k$, which means that one cluster receives all vertices. For any γ greater than 1, we obtain good 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. It is noteworthy that this behavior was not expected a priori, since in general we expect the edge-cut λ to increase and ρ to decrease as γ grows. Given the insensitivity of FENNEL to γ in this setting, we present our findings for γ fixed to $\frac{3}{2}$, in Table 4. 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 [12], but power-law graphs are commonly observed in practice so we consider them. 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 typical slope of real-world networks [24]. 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 assumes a value that may be considered unacceptably large for 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 $\lambda_{\text{Random}} = 63.51\%$.

Finally, we explore the graph-specific parameter selection. Specifically, we consider the degree sequence as an easy-to-compute graph statistic and we generate for a realistic range of power-law exponents [8] twenty random power-law graphs on 3000 vertices. The number of vertices is chosen to be large enough in order to ensure that for large power-law exponents the graphs have sufficiently many edges to guarantee concentration of our findings. For each exponent, we compute the average optimal value γ^* in the range $[1.5, 3.2]$ using a step of 0.1 over twenty generated graphs that results in the smallest possible fraction of edges cut λ conditioning on a maximum normalized load $\nu = 1.2$. Figure 2 shows the resulting error bars for 8 and 16 clusters respectively. The variance for each power-law exponent is very small compared to the square of the average, indicating a strong concentration around the mean as the error bars show. We observe that γ^* tends to be non-increasing function with the power-law slope α. Also, by comparing Figures 2(a),(b) we observe that γ^* is less sensitive to k, with a general trend of a decreasing γ^* as k increases. Understanding these functions and exploiting them for increasing systems' performance is an interesting problem for future work.

5.3 Real-World Datasets

We summarize our main findings as follows: (a) FENNEL *is superior to existing streaming partitioning algorithms.*

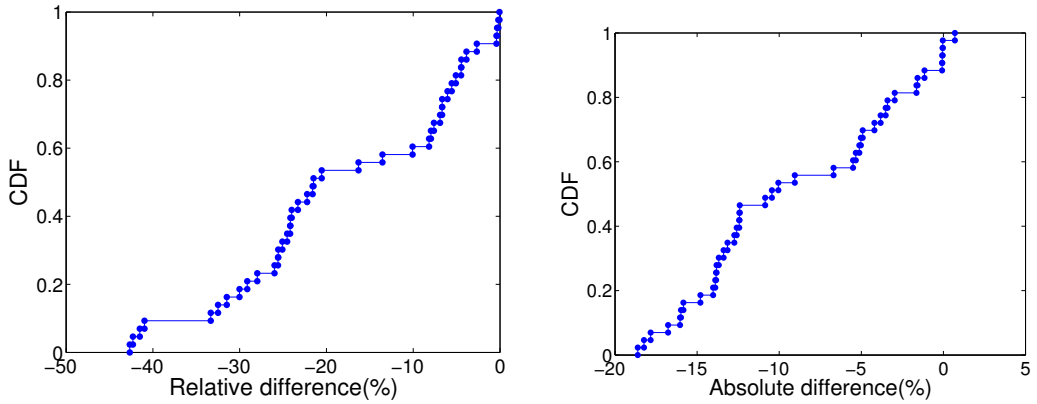

Figure 3: (Left) CDF of the relative difference $(\frac{\lambda_{\mathbf{Fennel}}-\lambda_{\mathbf{c}}}{\lambda_{\mathbf{c}}}) \times 100\%$ of percentages of edges cut of our method and the best competitor for all graphs in our dataset. (Right) Same but for the absolute difference $(\lambda_{\mathbf{Fennel}} - \lambda_{\mathbf{c}}) \times 100\%$.

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.

Specifically, over a wide range of k values and over all datasets FENNEL consistently outperforms the current state-of-the-art. FENNEL achieves excellent load balancing with significantly smaller edge cuts. (b) For smaller values of k (less or equal than 64) the observed gain is more pronounced. (c) FENNEL *is fast.* Our implementation scales 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, it is a graph that can be loaded into the main memory of a state-the-art personal computer. The performance of FENNEL on this graph is remarkably good. 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.

In Figure 3, we summarize the results of comparison results of FENNEL and all other heuristics that we consider on all graphs from our dataset in Table 2. In that figure, 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 absolute difference (relative difference) is in the excess of 20% (15%), thus providing significant performance gain.

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. Figures 4(a), (b) provide a closer look at both criteria and their relation. Specifically, we consider the difference of the fraction of edges cut versus 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 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 comment on the efficiency of our method with respect to the run 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

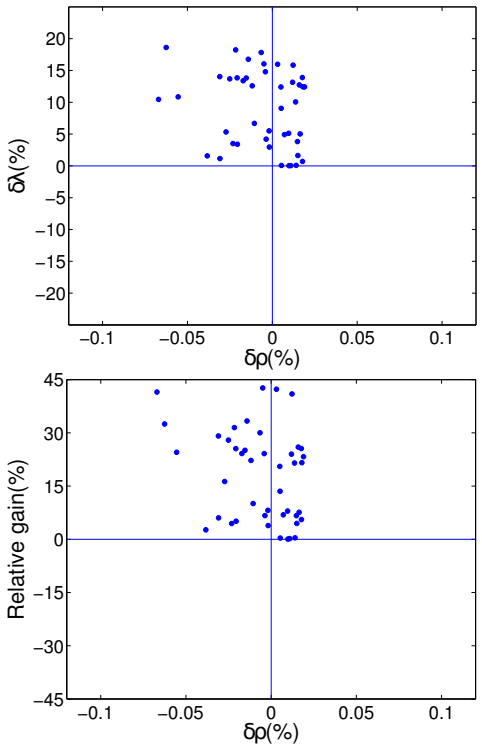

Figure 4: Absolute difference $\delta\lambda$ and relative gain versus the maximum load imbalance $\delta\rho$.

it took to partition each graph from our dataset versus the graph size m. We observe that few minutes suffice 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

Evaluating a partitioning algorithm is a non-trivial task from a systems perspective, since it depends on system characteristics. For instance, in a large-scale production data center, it may be more important to balance the traffic across clusters than the traffic or the amount of computation executed within a cluster. However, in a small-scale cluster consisting of few tens of nodes, rented by a customer from a large cloud provider such as Amazon's Elastic MapReduce, for example, it may be more important to well balance the computation load on each node. Given this diversity of scenaria, a detailed evaluation is out of the scope of this work. Here, we perform a basic experiment to verify the superiority of our proposed method versus the standard approach load balancing approach of hash partitioning with respect to speeding up a large-scale computation.

We consider Pagerank as the computational task. Notice, that an advantage of Fennel is that it gives a flexibility in choosing a suitable objective that accomodates the needs of the specific application. We demonstrate the efficiency and flexibility of Fennel with the typical Elastic MapReduce scenario in mind. 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 Apache Giraph, a graph processing

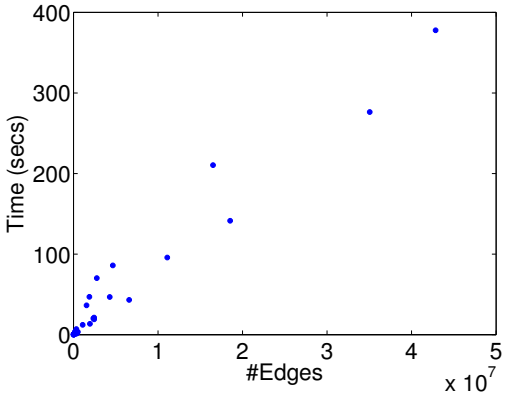

Figure 5: Elapsed running time for Fennel versus number of edges.

platform that runs on top of Hadoop. We implemented a PageRank algorithm on Giraph and run it on Live Journal graph data [1].

Since the complexity of PageRank depends on the number of edges and not vertices, we use a version of Fennel objective (eq. 1) that balances the number of edges per cluster with $\gamma = 1.5$ We compare with hash partitioning, the default partitioning scheme used by Giraph, with respect to the following two metrics. The first metric is the average duration of an iteration of the PageRank algorithm. This metric is directly proportional to the actual run time and incorporates both the processing and the communication time. The second metric is the average number of Megabytes transmitted by a cluster node in each iteration. This metric directly reflects the quality of the cut and is proportional to the incurred network load.

The results are shown in Table 6. We see that Fennel has the best run time in all cases. This is because it achieves the best balance between the computation and communication load. Hash partitioning takes 25% more time than Fennel and it also has a much higher traffic load.

7. CONCLUSION

In this work we provide a novel perspective on a recent line of research [25, 29] for the balanced graph partitioning problem, which results in state-of-the-art performance in terms of *speed* and *quality*. Despite the fact that Fennel performs a single pass over the graph, it achieves performance comparable to Metis, see also [25]. Furthermore, our general framework is particularly suitable for dynamic graph data. Also, it allows us to quantify the notion of interpolation between the two state-of-the-art heuristics [26, 29] for streaming partitioning. Specifically, we evaluate our proposed framework extensively both on synthetic and real-world graphs. We verify consistently over a wide range of number of clusters, the superiority of our method compared to existing ones. Finally, we demonstrate the benefits of the method in the graph processing platform Apache Giraph.

An interesting research problem is to derive hard approximation guarantees for the class of algorithms defined by Fennel. We provide initial results in this direction for

[1]At the time of the evaluation, the Twitter graph was too large for a 16-node Giraph cluster.

	Run time [s]		Communication [MB]	
# Clusters (k)	Hash	FENNEL	Hash	FENNEL
4	32.27	**25.49**	321.41	**196.9**
8	17.26	**15.14**	285.35	**180.02**
16	10.64	**9.05**	222.28	**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 on LiveJournal data set.

the special case related to the concept of graph modularity. However, additive error guarantees appear to be more suitable for our objective. Second, it would be of interest to understand on a firm mathematical ground, the function shown in Figure 2. Potentially, this understanding can lead to choosing optimally the parameter values based on simple graph characteristics, such as the exponent of the degree distribution for a power-law graph.

8. REFERENCES

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

[2] A. Angel, N. Sarkas, N. Koudas, and D. Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *VLDB*, 2012.

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

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

[5] 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.

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

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

[8] Sergey N Dorogovtsev and Jose FF Mendes. Evolution of networks. *Advances in physics*, 51(4):1079–1187, 2002.

[9] U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM J. Comput.*, 31(4), 2002.

[10] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. In *STOC*, 1974.

[11] 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.

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

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

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

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

[16] 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.

[17] 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.

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

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

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

[21] 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.

[22] 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.

[23] G. Malewicz and et al. Pregel: a system for large-scale graph processing. In *SIGMOD '10*, pages 135–146, 2010.

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

[25] J. Nishimura and J. Ugander. Restreaming graph partitioning: Simple versatile algorithms for advanced balancing. In *ACM KDD*, 2013.

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

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

[28] 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.

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

[30] C.E. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. Tsiarli. Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees. In *KDD'13*, 2013.

[31] Johan Ugander and Lars Backstrom. Balanced label propagation for partitioning massive graphs. In *WSDM '13*, pages 507–516, 2013.