Restreaming Graph Partitioning: Simple Versatile Algorithms for Advanced Balancing

Joel Nishimura[*]
Center for Applied Mathematics
Cornell University
Ithaca, NY
jdn48@cornell.edu

Johan Ugander[*]
Center for Applied Mathematics
Cornell University
Ithaca, NY
jhu5@cornell.edu

ABSTRACT

Partitioning large graphs is difficult, especially when performed in the limited models of computation afforded to modern large scale computing systems. In this work we introduce *restreaming graph partitioning* and develop algorithms that scale similarly to streaming partitioning algorithms yet empirically perform as well as fully offline algorithms. In streaming partitioning, graphs are partitioned serially in a single pass. Restreaming partitioning is motivated by scenarios where approximately the same dataset is routinely streamed, making it possible to transform streaming partitioning algorithms into an iterative procedure.

This combination of simplicity and powerful performance allows restreaming algorithms to be easily adapted to efficiently tackle more challenging partitioning objectives. In particular, we consider the problem of *stratified graph partitioning*, where each of many node attribute strata are balanced simultaneously. As such, stratified partitioning is well suited for the study of network effects on social networks, where it is desirable to isolate disjoint dense subgraphs with representative user demographics. To demonstrate, we partition a large social network such that each partition exhibits the same degree distribution in the original graph — a novel achievement for non-regular graphs.

As part of our results, we also observe a fundamental difference in the ease with which social graphs are partitioned when compared to web graphs. Namely, the modular structure of web graphs appears to motivate full offline optimization, whereas the locally dense structure of social graphs precludes significant gains from global manipulations.

Categories and Subject Descriptors: G.2.2 [**Mathematics of Computing**]: Discrete Mathematics—*Graph Theory, Graph Algorithms*

Keywords: Graph clustering, social networks, balanced partitioning, stratified partitioning, multi-constraint balance.

1. INTRODUCTION

The tremendous scale of modern graph datasets has rapidly increased the demand for efficient algorithms for graph analysis. With the World Wide Web featuring over a trillion URLs and online social networks such as Facebook featuring more than a billion active users, it is becoming increasingly difficult to perform even the simplest graph computations.

The tractability of large-scale graph computations often hinges upon the ability to efficiently partition a graph for distributed computation. The scale of this partitioning varies depending on the domain, but the lesson is the same: partitioning massive graphs for distributed computation can greatly decrease both network communication and runtime [24], while even in-memory computations can benefit from partitioned graph arrangements [14].

But partitioning large graphs is difficult, especially within modern limited models of large-scale computation. Responding to this, the goal of *streaming graph partitioning* is to partition the node set of a graph into k balanced disjoint subsets by serially examining only individual nodes and their local adjacency list. Importantly, a streaming graph partition algorithm is forced to make a permanent partition assignment the very first (and only) time it examines each node, as opposed to allowing the partitioning to come from post-processing, as in the semi-streaming model of computation [2]. The motivation for streaming graph partitioning is that often times the distributed systems performing graph computations are 'anyways' required to load a graph from a datastore, and one might as well execute this loading process – streaming the graph to the computation system – in an intelligent manner.

We introduce *restreaming graph partitioning*, which is motivated by situations where the same graph – or approximately the same graph – can be expected to be repeatedly streamed on a regular basis. After all, if a graph is going to be reloaded from a datastore with any regularity, streaming graph partitioning is making it unnecessarily difficult for itself by starting over from scratch with each stream. Instead, restreaming graph partitioning retains node assignments across streams, allowing subsequent streams to produce partitionings with fewer cut edges.

In fact restreaming can produce partitions of such quality and with such modest memory requirement that restreaming graph partitioning merits serious consideration as an efficient iterative streaming algorithm well outside the motivating 'data loading' context. Surprisingly, we find that after only a handful of restreams, our restreaming graph partition algorithms converge upon graph partitions competitive with or even superior to a fully offline partitioning algorithms, METIS [11], in a number of important instances.

In particular, restreaming graph partition algorithms cut fewer edges than METIS in social graphs, though they cut more edges in web graphs. Indeed, it is well understood that social graphs and web graphs are quite different in structure [26, 6]. We posit that there is also a fundamental difference in the partitioning of web and social graphs. While the local dense structure of social graphs precludes very high quality partitionings, it rewards the local greedy

[*]All authors contributed equally to this work.

moves of restreaming graph partitioning. Meanwhile, the fully offline optimization of METIS is able to discover the extremely high quality partitions of web graphs through multi-level coarsening and non-greedy Kernighan-Lin refinement [13]. Given the increasing size and importance of social graphs, it is important to develop new lightweight algorithms specially designed with these in mind.

Towards this goal, we construct restreaming versions of the streaming partitioning algorithms *Linear Deterministic Greedy* (LDG) and *FENNEL*, algorithms developed in [22] and [23] respectively, both greedy heuristic approaches to partitioning. Where Linear Deterministic Greedy uses multiplicative weights to guarantee balance, FENNEL mimics modularity maximization [4, 19] by using regularization to direct a greedy assignment strategy towards balance. This regularization approach does not itself guarantee balanced partitions, and as part of this work, we show how one can 'temper' such a regularization over the course of restreams to obtain a restreaming variation on FENNEL that can ultimately guarantee balance in a way that ordinary modularity maximization can not.

Given that restreaming these highly scalable algorithms can bring them into competition with fully offline methods, we show how their scalability makes it possible to adapt towards much more sophisticated objectives. Indeed, for certain types of distributed graph computation it can be desirable to obtain more sophisticated notions of balance than just the number of nodes [11]. It is straight forward to modify any of the streaming and restreaming algorithms we consider to balance any cumulative node attribute, for example the total degree of each partition, (or as in [23], the number of internal edges on each partition).

But beyond simply balancing one attribute, we show that significantly more sophisticated notions of balance, similar to *multi-constraint* balance from high-performance computing [11], are obtainable. First, we show how the multiplicative weights in LDG can be modified to balance both node count and edge count at once. Moreover, we show how restreaming LDG and FENNEL can be adapted to efficiently perform *stratified graph partitioning*, a constrained graph partitioning problem we introduce that aims not just to balance nodes across partitions, but also ensure that each partition of the graph exhibits a balanced proportion of nodes from an arbitrary number of strata. This offers an important contribution for the study of social networks, making it possible to create dense balanced clusters, where each cluster contains an equal proportion of users from several age strata, countries, activity levels, and friend counts. As an important demonstration, we study *degree-stratified graph partitioning*, where each balanced cluster is required to exhibit the same degree distribution.

The social network example above addresses an important problem in online social network experimentation [8]. In such experiments, one wishes to select treatment and control groups that are structurally isolated from each other in order to minimize spillover effects. Without stratified balance constraints, it is natural to partition a social graph either geographically or according to some other basis of assortativity. As a result, ordinary graph partitioning, without stratified balance, risks producing graph partitions that are highly heterogenous, none of the partitions being representative. In introducing stratified graph partitioning, we hope to contribute a highly scalable partitioning methodology useful as a stratification technique for variance reduction in network experimentation [25], and also cross validation on graphs [18].

Lastly, we discuss parallelization. A notable drawback of single-shot streaming partitioning is that it is fundamentally serial, making parallelization difficult without continuous communication between parallel workers [23]. These algorithms are specifically intended for partitioning extremely large graphs, and we show how restreamed graph partitioning can be easily parallelized – communicating only between stream iterations – at only a small cost in the final partition quality.

2. STREAMING PARTITIONING

Multi-way graph partitioning is a classical NP-hard problem. Even the two-way partitioning problem *minimum graph bisection* is NP-hard [9], with the best known polynomial time approximation algorithm achieving only a $O(\sqrt{n}\log(n))$-factor approximation [7] for general graphs. Similarly, semi-streaming algorithms can guarantee weak approximation bounds for graph cuts while utilizing only $O(n\text{polylog}(n))$ memory [2]. Meanwhile, a robust community of research has emerged to develop efficient algorithms that achieve good performance on real world graphs. Among existing offline algorithms, we focus on the METIS package [11] for graph partitioning, and use METIS as our running basis for comparison when comparing online to offline methods.

In this section, we review streaming graph partitioning and introduce restreaming graph partitioning. We show how FENNEL, a streaming algorithm previously without balance guarantees, can be 'tempered' to guarantee balance. We then discuss how our restreaming framework is capable of both managing dynamic graph partitioning and efficient parallelization.

2.1 The streaming model

We now review the basic details of the streaming partitioning model. Let $P^t = \{P_1^t, \ldots, P_k^t\}$ denote a k-way partitioning of the node set at time t, where P_i^t is the set of nodes in partition i at time t and $P^t(u)$ denotes the partition that contains node u. A streaming algorithm is sequentially presented a node u and its neighbors $N(u)$, and it must assign u to a partition i utilizing no more information than contained in the current partitioning P^t. Over the course of a stream, the time counter advances by one for each node it examines.

Since streaming graph partition algorithms make decisions based on an incomplete but increasing amount of information, the order in which data is streamed can affect performance, and worst-case orders can easily undermine the streaming approach [22, 23]. However, it has generally been observed that presenting the data in either a breadth first, depth first, or in a random order does not greatly alter performance [22, 23]. Of these orders, a random ordering is the simplest to guarantee in large-scale streaming data scenarios, and so we restrict our analysis to only consider random node orders for simplicity. When considering restreams later on, we focus on persistent random orders.

Finding partitions that are strictly balanced, where $|P_i| = |P_j|$ for all i and j, is rarely necessary. As a result, many partitioning algorithms [12, 23] include a 'slackness' parameter, explicitly or implicitly allowing deviations from exact balance, often in exchange for superior cuts. As part of this work, we present algorithms for exact balance and also modifications for 'slacked' balance.

Stanton and Kliot [22] considered a broad range of heuristics for performing streaming node assignment. Of these heuristics, the method with the best performance was 'Linear Deterministic Greedy' (LDG), where each node u is assigned to the partition

$$\underset{i\in\{1,\ldots,k\}}{\text{argmax}}\ |P_i^t \cap N(u)|\left(1 - \frac{|P_i^t|}{C_i}\right), \tag{1}$$

where C_i is the maximum capacity of partition i. Notice that since this examines each node but once, $|P_i^t \cap N(u)|$ will be exactly 0 for many nodes at the start of the stream, and $|P_i^t \cap N(u)|$ is only likely to reflect the actual number of neighbors a node shares with a partition near the end of the stream. Single shot LDG exhibits impressive performance despite this handicap. While the original investigation of LDG was merely heuristic, subsequent work has shown that an algorithm inspired by LDG is capable of recovering a planted partitioning from a basic infinite random graph model, and also that no streaming algorithm can obtain an $o(n)$ approximation with a worst-case or random stream ordering on an arbitrary graph [21].

Meanwhile, FENNEL [23], a streaming generalization of modularity maximization, attempts to maximize the following objective function:

$$H = \sum_{u \in V} |P^t(u) \cap N(u)| - \frac{\alpha}{2} \sum_{i=1}^{k} |P_i^t|^\gamma. \quad (2)$$

Notice that when $\gamma = 2$, the regularization becomes functionally equivalent to $\alpha \sum_i \binom{|P_i^t|}{2}$, which is equivalent to modularity maximization with an Erdős-Rényi baseline with probability α. As a streaming greedy maximization, maximizing this objective function corresponds to assigning u to the partition that maximizes the change $\Delta H_i^t(u) = |P_i^t \cap N(u)| - \frac{\alpha}{2}[(|P_i^t|+1)^\gamma - (|P_i^t|)^\gamma]$. To first order, this corresponds to maximizing $|P_i^t \cap N(u)| - \alpha\frac{\gamma}{2}(|P_i^t|)^{\gamma-1}$, where the first order approximation is exact for $\gamma = 2$. Thus the FENNEL assignment rule is:

$$\underset{i \in \{1,\ldots,k\}}{\operatorname{argmax}} \; |P_i^t \cap N(u)| - \alpha \frac{\gamma}{2}(|P_i^t|)^{\gamma-1}. \quad (3)$$

In this work we focus our analysis of FENNEL on the special case of $\gamma = 2$, namely streaming modularity maximization.

While the multiplicative weights of LDG enforce exact balance, the additive regularization used by FENNEL only ensures approximate balance. While it is straightforward to show that this assignment mechanism must produce exact balance for $\alpha > \lceil \frac{n}{k} \rceil$, such a large α focuses almost entirely on balancing and leads to a very poor partitioning. Nonetheless, when run at appropriately chosen values of α, FENNEL performs very well on a number of real world networks, producing very nearly balanced partitions [23].

The first phase of many common multiphase modularity maximization algorithms, including the Louvain method [4] and modularity-specialized label propagation [16], bear a clear similarity to FENNEL. The connection between regularizing label propagation and modularity maximization was also outlined by Barber and Clark [3]. By restreaming FENNEL, we show how modularity maximization also fits well within a restreaming framework.

3. RESTREAMING PARTITIONING

For the distribution of very large graphs, the utility of streaming graph partitioning derives from the routine need to stream graph datasets, and when performing this streaming it can be worthwhile to attempt to partition the dataset with some intelligent assignment mechanism. It is equally routine, however, that the streaming process is repeated periodically, and often frequently.

For example, a social networking service might be interested in a streaming partitioning algorithm because it loads a graph from memory to dedicated ranking servers on a daily basis [24]. However, if a streaming algorithm sees nearly the same data routinely, it is clearly worth considering what information can be retained between streams so as to improve performance.

We thus introduce the concept of restreaming graph partitioning, and in particular we present restreaming versions of LDG and FENNEL, the two single-shot streaming graph partitioning algorithms presented earlier. In our restreaming framework, subsequent streams of LDG and FENNEL have access to the result of previous streams. We consider a graph as being streamed in a random but persistent order each time it is restreamed, and we use persistent (de-randomized) tie breaking across restreams.

3.1 Restreaming LDG

In the case of restreaming LDG, P_i^t records the most recent partition assignment, either from the previous stream or, when present, from the current stream. Additionally, let x_i^t record the number of nodes assigned to i during the current stream. The assignment rule for restreaming LDG remains functionally similar to (1),

$$\underset{i \in \{1,\ldots,k\}}{\operatorname{argmax}} \; |P_i^t \cap N(u)| \left(1 - \frac{x_i^t}{C_i}\right). \quad (4)$$

Since each x_i^t increases over each stream from 0 to C_i, the partitioning achieves exact balance at the end of each stream. Conceptually, restreaming LDG resembles a repeated shooting method, where each time the partitions are built up anew, with the benefit of the probable assignments for nodes not yet seen in the current stream. Since LDG matches the constraints C_i after each restream, it is ideal for applications with hard constraints, otherwise these hard constraints can be loosened by setting $\Sigma_i C_i > n$.

3.2 Restreaming FENNEL

FENNEL can be restreamed without any change to its objective function. Whereas restreaming LDG rebuilds the partitioning each time, and thus involves implicit notions of the beginning and end of a stream, FENNEL's objective function can be computed without knowing its location in the stream. On the other hand, this property of FENNEL prevents it from reaching exact balance after a single stream. In the restreaming scenario, however, we show that it is possible to achieve exact balance using FENNEL by 'tempering' the solution towards increasingly balanced partitions over repeated restreams. Namely, with each restream we run FENNEL with a larger value of parameter α, denoting the value of α during stream s as α_s. In this way tempering increasingly emphasizes balance, while granting time in the earlier streams to finding high quality partitions. Alternatively, had FENNEL been run with too high an initial α, the algorithm would have resorted to placing nodes in partitions based solely on balance and without regard to the quality of the partitions. As noted earlier, once each node is reconsidered by a stream of FENNEL for which $\alpha_s > \lceil \frac{n}{k} \rceil$, the assignment mechanism will necessarily return a balanced partition. We formalize this observation through the following proposition.

PROPOSITION 1. *If $\alpha_s > \lceil \frac{n}{k} \rceil$ then at the completion of restream s, $|P_i^t| \in \{\lfloor \frac{n}{k} \rfloor, \lceil \frac{n}{k} \rceil\}$ for all i.*

PROOF. Suppose not: then at some time $\tau \leq t$ a node u was assigned to a partition i where $|P_i^\tau| - |P_j^\tau| \geq 1$ for j being the smallest partition. Since $|P_i^\tau \cap N(u)| \leq |P_i^\tau|$:

$$\begin{aligned} \Delta H_i^\tau(u) &= |P_i^\tau \cap N(u)| - \alpha_s |P_i^\tau| \\ &\leq |P_i^\tau| - \alpha_s |P_i^\tau| \\ &= -|P_i^\tau|(\alpha_s - 1), \\ \Delta H_j^\tau(u) &= |P_j^\tau \cap N(u)| - \alpha_s |P_j^\tau| \geq 0 - \alpha_s |P_j^\tau|. \end{aligned}$$

Then:

$$\begin{aligned} \Delta H_i^\tau(u) - \Delta H_j^\tau(u) &= \alpha_s |P_j^\tau| - |P_i^\tau|(\alpha_s - 1) \\ &\leq \alpha_s |P_j^\tau| - (|P_j^\tau| + 1)(\alpha_s - 1) \\ &= |P_j^\tau| + 1 - \alpha_s. \end{aligned}$$

As P_j^τ is the smallest partition, $|P_j^\tau| \leq \lceil \frac{n}{k} \rceil - 1$, meaning that $\Delta H_i^\tau(u) - \Delta H_j^\tau(u) < 0$ — a contradiction as u would have then been assigned to partition j. □

When the maximum degree $d < \lceil \frac{n}{k} \rceil$ it can be shown the requirement is relaxed to $\alpha_s > d$. It's also clear that if restreaming FENNEL finds a balanced partition for some $\alpha < \lceil \frac{n}{k} \rceil$ then further tempering will not change that partition.

PROPOSITION 2. *If at some time t_0, $P^{t_0+j} = P^{t_0}$ for all $j = 1, \ldots, n$ (one complete stream), $|P_i^{t_0}| \in \{\lfloor \frac{n}{k} \rfloor, \lceil \frac{n}{k} \rceil\}$ for all i, and $\alpha_{s+1} \geq \alpha_s$ for all s then $P^t = P^{t_0}$ for all $t > t_0$.*

PROOF. We prove this by induction. Suppose no node has moved from time t_0 to some $t > t_0 + n$, and node u in partition j is the next node in the stream at time $t+1$. Since no nodes have moved for time n, $|P_i^{t+1-n} \cap N(u)| = |P_i^{t+1} \cap N(u)|$ and thus:

$$\begin{aligned} \Delta H_j^{t+1} - \Delta H_j^{t+1-n} &= -(\alpha_{s+1} - \alpha_s)(|P_j^{t+1}| - 1), \\ \Delta H_i^{t+1} - \Delta H_i^{t+1-n} &= -(\alpha_{s+1} - \alpha_s)|P_i^{t+1}| \quad u \notin i. \end{aligned}$$

Since α is increasing and $|P_j^{t+1} - 1| \leq |P_i^{t+1}|$ then $\Delta H_j^{t+1} - \Delta H_j^{t+1-n} \geq \Delta H_i^{t+1} - \Delta H_i^{t+1-n}$. Thus, as u was assigned to j at time $t+1-n$, and ties are broken consistently, u will also be assigned to j at time $t+1$. □

As we discuss later, when using tempered FENNEL to partition real world graphs we observe that the quality of the final partitioning is relatively insensitive to the initial value of α_0. This is a somewhat surprising observation that has the added benefit of removing the α_0 selection problem present in single stream FENNEL. Meanwhile, there remains a trade off between computation and performance in choosing how fast to temper, though our empirical results suggest that moderate numbers of restreams are typically sufficient.

3.3 Convergence over restreams

Every node allocation/relocation in FENNEL increases its objective function. As there are only a finite number of different possible partitions, FENNEL will converge to a final partitioning at any fixed α given enough restreams, even if α is not tempered all the way to the bound established in Proposition 1. But since α is in theory a continuous parameter, we may be concerned that the solutions differ at exponentially different values of α. While it is not of practical importance — since it is always possible to make large changes in α when tempering — we establish a resolution limit of α, the granularity below which the partitioning solution can not change. We show that there are only polynomially many unique values of α for which any changes in partitioning can occur. We emphasize that this resolution limit is much finer than the amount that we choose to increase α by in practice, but this investigation illustrates important structures of the tempering framework.

PROPOSITION 3. *For some $\alpha_s > 0$, and a partitioning P^t, then for any increasing sequence of L values $\alpha_s < \alpha_{s+1} < \ldots < \alpha_{s+L} \leq \alpha_s + \frac{1}{n^2}$ on which FENNEL is repeatedly restreamed to convergence, there is at most one value of $\alpha_{s+\ell}$, $0 \leq \ell \leq L$, such that the converged partitioning at $\alpha_{s+\ell}$ is different from the converged partitioning at $\alpha_{s+\ell+1}$.*

PROOF. Suppose there are two distinct pivotal $\alpha_{s+\ell}$, denoted, α_a and $\alpha_b > \alpha_a$, such that at α_{a+1} and α_{b+1} FENNEL converges to a different partition than at α_a and α_b. Let $\delta_{ij}^a(u) = |P_i^{t_a} \cap N(u)| - |P_j^{t_a} \cap N(u)|$, and $x_{ij}^a = |P_i^{t_a}| - |P_j^{t_a}|$. It must then be that there are partitions i, j, p and q and nodes u and v such that for these different α values, nodes would switch between partitions implying a change in sign of $\Delta H_i^{t_a}(u) - \Delta H_j^{t_a}(u)$ and of $\Delta H_p^{t_b}(v) - \Delta H_q^{t_b}(v)$ giving:

$$\begin{aligned} \delta_{ij}^a(u) - \alpha_a x_{ij}^a \geq 0 \qquad \delta_{ij}^a(u) - \alpha_{a+1} x_{ij}^a < 0 \\ \delta_{pq}^b(v) - \alpha_b x_{pq}^b \geq 0 \qquad \delta_{pq}^b(v) - \alpha_{b+1} x_{pq}^b < 0. \end{aligned}$$

Notice that it must be that $x_{ij}^a \neq 0$ and $x_{pq}^b \neq 0$. Without loss of generality assume that each $x_{ij}^a > 0$ and $x_{pq}^b > 0$, then for u or v to be assigned to i and p respectively, it must be that $\delta_{ij}^a(u) \geq 0$ and $\delta_{pq}^b(v) \geq 0$ as well. Thus it can be shown that: $(\alpha_{b+1} - \alpha_a)x_{ij}^a x_{pq}^b > \delta_{pq}^b(v)x_{ij}^a - \delta_{ij}^a(u)x_{pq}^b > 0$. Notice then that $c = \delta_{pq}^b(v)x_{ij}^a - \delta_{ij}^a(u)x_{pq}^b$ is both positive and an integer, yielding that $\alpha_{b+1} - \alpha_a > \frac{c}{x_{ij}^a x_{pq}^b} > \frac{1}{n^2}$, a contradiction. □

Note that this convergence is in theory incredibly slow. In practice it is vastly more efficient to increase α in larger steps, and without waiting for convergence at each value of α. Indeed, the tempering results we present later correspond to increasing α at an exponential rate, from an initial α_0 to the critical α for which FENNEL is guaranteed to be balanced.

When restreaming FENNEL, tempered or untempered, it ultimately converges only to one of many local maxima of its modularity-like objective function. As discussed in [10], modularity typically has many high quality local maxima, which is of great practicality if one merely needs to find high quality partitions, but also of grave concern when using modularity to discern 'community structure', something we are not attempting.

LDG does not have any of the same convergence guarantees. Indeed, restreaming LDG does not necessarily converge. Furthermore, should it converge, the resulting partitioning would depend upon the specific node ordering: if the graph was restreamed in a different order then nodes would be very likely to move. By comparison, the convergence of FENNEL and tempered FENNEL outlined above do not depend on any persistence in the node order. Despite the lack of convergence guarantees, LDG performs well, returning a balanced set after each restream. This lack of convergence guarantee is also one of LDG's strengths, enabling it to handle dynamic graphs very well.

3.4 Dynamic graphs

In real world settings, large empirical graph datasets are typically not static graphs, but rather they are slowly varying in time, with their edges sets evolving gradually relative to their immense size. In such cases, the graph may be expected to change slightly between streams, and it is important to consider the ability of both restreaming algorithms to accommodate dynamic graphs. One advantage of restreaming LDG is that it doesn't require any modification: since LDG rebuilds the graph each stream there aren't any restrictions on how the graph changes each time.

On the other hand, restreaming FENNEL is able to accommodate dynamic graphs only when α is held fixed in a manner similar to ordinary single stream FENNEL. When FENNEL is tempered across restreams, the resulting partitioning becomes increasingly rigid, unable to adjust to dynamic changes in a graph. In this way, FENNEL is only appropriate for dynamic graphs in its untempered form, precluding situations where exact balance is important.

3.5 Parallelization

Despite the simple computations and manageable memory footprint involved in LDG and FENNEL, in some settings the sheer size of the dataset being streamed may make parallelization highly desirable. Indeed, parallelizing single stream LDG and FENNEL is possible, but requires that a list of size $O(n)$ on each parallel thread is kept concurrent. In contrast, restreaming LDG and FENNEL can be parallelized without any communication during a stream, instead relying purely on inter-stream communication; thus speeding the streaming process by a factor equal to the number of machines used. Namely, for W workers, each of which will see a unique random $\frac{1}{W}$ fraction of a stream, we parallelize restreaming LDG and FENNEL in the following way. Each worker partitions their own $\frac{n}{W}$ nodes by utilizing the previous streams partitioning for nodes not in their stream, and the most recent destination of those $\frac{n}{W}$ nodes in their stream. Between restreams, each worker reports on their share of the partitioning and this compiled list is distributed to all workers for the next restream. For the first stream, we can initialize the partitioning utilizing a hash function applied to the node indices. Notice that this puts the first stream of LDG and FENNEL at significant performance disadvantage, but interestingly, this is largely overcome by additional restream iterations.

Thus these algorithms can be parallelized without communication for only a small partition quality tradeoff. Note that the parallelized implementation of the offline partitioning package METIS [12] also requires a similarly small quality tradeoff.

4. GENERALIZED TYPES OF BALANCE

In many situations the true objective function may depend not on balancing nodes, but on balancing edges, a combination of nodes and edges or some other more complicated function. In this section, we show how our restreaming algorithms can be modified to guarantee more general types of balance. In particular, we present a new balancing objective we call *stratified graph partitioning*, where an arbitrary number of node strata are each required to be balanced.

4.1 Balancing other quantities

The simplicity and directness of LDG and modularity maximization allow for straightforward generalizations. Indeed, notice that we can modify restreaming LDG's objective function, Equation 4, to balance the sum of the degrees of each partition. The object function can be modified as:

$$\underset{i\in\{1,\ldots,k\}}{\text{argmax}}\ |P_i^t \cap N(u)| \left(1 - \frac{x_i^t}{C_i}\right), \tag{5}$$

where $x_i^t = \Sigma_{u\in P_i^t}|N(u)|$ is the sum of the degrees in P_i^t and C_i is set to be $\lceil \frac{m}{k} \rceil$. More generally, x_i^t can be the sum of any positive node weights c_u, and each $C_i = \frac{1}{k}\sum_u c_u$, is simply the total possible sum split k ways. In this framework $c_u = 1$ corresponds to node balance, $c_u = |N(u)|$ corresponds to balancing degrees, and $c_u = 1 + \frac{n}{2m}|N(u)|$ treats node balance and degree balance as equally important. Here c_u can in fact be any arbitrary positive attribute calculated for each u a priori, such as the number of friends of friends on a social network, or the number of log records each node produced in the past month.

Note that when running this more general version of LDG, exact balance of the node attribute is no longer precisely guaranteed due to granularity. For example, when balancing degree and assigning a very high degree node late in a stream, that node will invariably push the sum x_i^t in Equation 8 over the threshold C_i. Thus LDG will only balance partitions to within the maximum value of c_u.

Similarly to LDG, FENNEL can be reinterpreted as a function on the total weights of each partition, rather than just the number of nodes. The general assignment rule can then be stated simply as

$$\underset{i\in\{1,\ldots,k\}}{\text{argmax}}\ |P_i^t \cap N(u)| - \alpha x_i^t. \tag{6}$$

The above modifications make it possible to address alternative notions of balance, which raises the important question of whether multiple balancing objectives can be obtained simultaneously. For LDG, we note that it is possible to adjust the multiplicative weights to attempt to balance multiple objectives simultaneously. Consider adjusting LDG such that u is assigned to:

$$\underset{i\in\{1,\ldots,k\}}{\text{argmax}}\ |P_i^t \cap N(u)| \left(1 - \frac{x_i^t}{C_i}\right) \prod_{\ell} f_\ell((\bar{y}_{\ell,i}^t - \bar{y}_\ell)(\bar{y}_\ell - y_{\ell,u})) \tag{7}$$

where $\bar{y}_\ell$ is the average value of objective ℓ on G, $\bar{y}_{\ell,i}^t$ is the average value of ℓ in P_i^t at time t, $y_{\ell,u}$ is the value of ℓ at u and $f_\ell(x)$ are positive increasing functions. For example, if the argument of f_ℓ is $(d_i - \frac{2m}{n})(\frac{2m}{n} - d_u)$ where d_i is the average degree of nodes in P_i^t, and d_u is the degree of u, then notice that the quantity is positive if and only if adding u to P_i^t moves the average degree of P_i^t towards the average degree of the graph. Thus, the strict LDG multiplicative forcing term forces this assignment rule to exactly balance nodes, while the second multiplicative forcing term biases the algorithm towards edge balance. While this algorithm does not guarantee strict balance on both edges and nodes simultaneously, it does balance these well empirically.

4.2 Stratified balance

Optimizing against multiple constraints rapidly increases the difficulty of the partitioning problem. In contrast, we introduce a restricted problem whose goal is to balance the counts of nodes from several distinct strata. This problem arises when it is important that each individual partition resembles the demographics of a full graph. For example, in social network experiments it can be important that test groups have an equal number of men and women, or have similar levels of educational attainment. If the network is assortative under these demographic traits then a good partitioning algorithm risks producing slices that are very different demographically. A particularly interesting instance of this problem is creating graph partitions that share the same degree distributions (up to integer divisibility). A node's degree in the full graph G is always available to the partitioning algorithm, and it commonly relates to important demographic node attributes such as age or geography, making balancing degree distributions a good proxy for demographic balance.

While in many situations producing miniaturized, representative partitions is a natural goal, it is quite different from the goal, or output, of many graph partitioning algorithms. For example, spectral partitioning tends to produce bisections with very different degree distributions, usually with one dense connected partition containing nodes of high degree and the other with low degree nodes that were successively 'trimmed' away from the first partition [20]. Colloquially this tendency of spectral partitioning can be described as partitioning a graph into a 'hairball' and 'whiskers' [15]. For graphs that have a pronounced core-periphery structure [5], algorithms that minimize the edge cut of a partition frequently (and rightfully so) split the graph between the core and the periphery. This results in a high quality cut between partitions with very different degree distributions, and typically, very different types of nodes.

Similarly, algorithms that aim to perform community detection are frequently tested for their ability to take graphs and produce communities whose nodes are fundamentally different. Indeed, community detection algorithms frequently demonstrate their effectiveness by revealing hidden node information utilizing only network information. In this way, community detection algorithms are frequently calibrated to create the worst possible test groups, and the partitions that least resemble the graph as a whole.

In contrast, the goal of stratified graph partitioning is to produce partitions where a node's membership in a partition reveals no information of that node's strata. We formalize the problem as:

PROBLEM 1 (STRATIFIED GRAPH PARTITIONING). *For a graph $G = (V, E)$, where nodes belong to L disjoint strata V_i such that $\cup_{i=1}^{L} V_i = V$, partition the graph into k disjoint partitions P_j such that $\cup_{j=1}^{k} P_j = V$, maximizing the number of uncut edges $\frac{1}{2}\sum_u |P(u) \cap N(u)|$, subject to the constraints that $|P_i \cap V_j| \leq C_{ij}$ for all i, j, and constraints C_{ij}.*

Setting each $C_{ij} = \lceil \frac{|V_j|}{k} \rceil$ requires that each partition proportionally represent the distribution of the strata in the original graph.

Despite the daunting increase in constraints, it is easy to adjust both of our simple restreaming algorithms to address this problem. For LDG this simply requires keeping additional indexing, such that a node $u \in V_\ell$ is assigned to:

$$\underset{i \in \{1,\ldots,k\}}{\operatorname{argmax}} \; |P_i^t \cap N(u)| \left(1 - \frac{x_{i,\ell}^t}{C_{i,\ell}}\right). \tag{8}$$

Likewise, one can adjust FENNEL's additive regularization, Equation 6, so that when assigning node $u \in V_\ell$, each x_i is also dependent on ℓ, becoming $x_{i,\ell}$.

Stratified graph partitioning has an interesting intersection with METIS in the high performance computing literature. Namely, in *multi-constraint graph partitioning* each node has an associated weight vector w_u, and the partitioning aims to balance the sum of these weights for each partition [12]. The primary aim of multi-constraint graph partitioning in the context of high performance computing is to enable efficient parallelization of large computations by dividing meshes into partitions with similar number of nodes and other attributes that affect either memory or computational requirements. The multi-constraint graph partitioning approach can apply to the stratified graph partitioning problem as well: simply consider a vertex of strata j as having weight $w_u[j] = 1$ and $w_u[i] = 0$ otherwise.

However, the added generality of multi-constraint partitioning leads the METIS implementation to have a memory footprint of $\tilde{O}(m + Ln)$, as it stores each node's weight vector in memory. Meanwhile, the modified restreaming version of LDG requires only memory $\tilde{O}(n + Lk)$. While in the high performance computing literature, it may not be necessary to have large L, if the goal is to match degree distributions it is frequently desirable to have L on the order of $\Theta(\frac{m}{n})$, at which point L has a large impact on the runtime of METIS. For example, whereas METIS was able to partition the LiveJournal graph with 9GB of RAM, when doing 100 degree strata the memory footprint rises to 23GB. Meanwhile, the memory footprint of LDG barely changes as the number of strata are increased.

Finally, there is an important difference in emphasis between multi-constraint graph partitioning and stratified graph partitioning. While multi-constraint graph partitioning can perform stratified graph partitioning, it does so by balancing marginalized traits and not joint constraints. One must be careful of this distinction lest one may balance gender and degree by stacking one partition with high degree women and low degree men, and the other with high degree men and low degree women. This would not produce slices with comparable composition. Instead, one should make a Cartesian product of the features so that each combination of features belongs to a distinct strata.

When it is important to do streaming or restreaming multi-constraint partitioning instead of stratified partitioning, we note that the framework of Equation 7 in Section 4.1 can be adjusted to allow multiple constraints, though we do not examine this in our results.

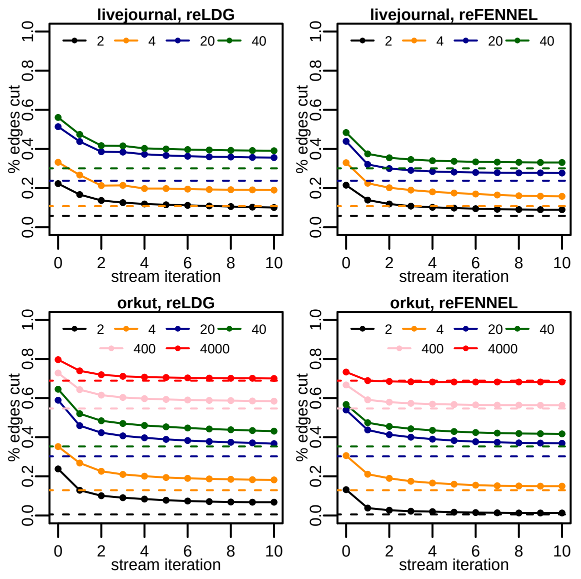

Figure 1: Iterating the restreaming partitioning process for static LiveJournal and Orkut graphs. The left column reports results for restreaming LDG, the right column for restreaming FENNEL. Dashed lines are METIS. Iteration zero corresponds to single-shot streaming implementations, though note that FENNEL does not guarantee balance until its final iteration due to ongoing tempering.

5. RESULTS

We now examine the performance of our restreaming partitioning algorithms on ten empirical graphs: six social graphs and four web graphs, listed in Table 1. All our graphs were obtained from the SNAP repository [1] except for the Orkut graph [17]. Graphs were made undirected by reciprocating all arcs. Self-loops and nodes with degree zero were removed in order to aid the interpretability of the fraction of edges cut by a partitioning algorithm.

The densest graph we analyze here was the Orkut graph, with 3.1 million nodes and 117 million edges. The algorithms we discuss scale effortlessly beyond this size, but we are not able to analyze graphs larger than this in comparison to METIS — performing ordinary node balanced graph partitioning on Orkut in METIS already requires 18 GB of RAM, making larger graphs intractable. For the Orkut graph our restreaming LDG algorithm utilizes just 200 MB of RAM for the same graph. Since the average degree of the Orkut graph is 76, this is very nearly the expected factor of 76 times smaller. In order to compare our results to METIS we focus our analysis on graphs up to this size.

5.1 Node balance results

We begin by discussing our performance for the standard node balanced partitioning problem. When evaluating single-shot streaming graph partitioning algorithms, it is unclear if the gap in quality between streaming and offline algorithms should be attributed to the limitations of the single-shot view of the graph or attributed to the limited local means of the algorithm. After examining the performance of our restreaming algorithms, it is clear that much of the gap can be attributed to the limits of the single-shot view, not to a fundamental limitation of local algorithms.

Both restreaming LDG and tempered FENNEL were effective on all the graphs. However, in examining our results, it is important to distinguish between 'web' graphs, whose structure derives from the structure of hyperlinks on the internet, and 'social'

Graph	|V|	|E|	avg deg	LDG	reLDG	reFENNEL	reFENNEL-llel	METIS(1.001)	METIS(1.03)
wikivote	7115	100762	28.32	0.867	0.775	**0.685**	0.775	0.822	0.764
astro-ph	18771	198050	21.10	0.623	0.439	**0.413**	0.438	0.535	0.372
enron	36692	183831	10.02	0.664	0.490	**0.471**	0.482	0.855	0.411
slashdot	77360	469180	12.12	0.821	0.730	**0.673**	0.686	0.711	0.693
livejournal	4846609	42851237	17.68	0.561	0.390	0.328	0.351	**0.309**	0.301
orkut	3072441	117185083	76.28	0.645	0.428	0.421	0.585	**0.376**	0.353
web-nd	325729	1090108	6.69	0.313	0.128	0.121	0.181	**0.036**	0.036
web-stanford	281903	1992636	14.13	0.378	0.207	0.176	0.237	**0.123**	0.114
web-berkstan	685230	6649470	19.41	0.341	0.203	0.188	0.283	**0.117**	0.111
web-google	875713	8644106	19.74	0.290	0.163	0.160	0.206	**0.009**	0.008

Table 1: The percentage of edges cut (lower is better) for the basic methods studied in this paper applied to a diverse collection of graphs partitioned into 40 different partitions. The restreamed methods were run for 10 restreaming iterations while the parallel versions were split across 30 workers and run for 30 restreaming iterations. METIS(1.03) is run with 3% slack, while METIS(1.001) is run with slack 0.1% slack. For each graph the best score excluding METIS(1.03) is bolded.

graphs, whose structures represent relationships people create between each other. Indeed, there are well known differences between the structure of web and social graphs, both in their degree distribution, effective diameter, their clustering coefficients [26] and in their compressibility [6]. Social graphs are known to have significantly higher average local clustering coefficient, indicating that the structure around individual nodes is far from divisible, while web graphs are known to compress much better than social graphs.

Consistent with these observations, web graphs have extremely high quality cuts, as seen in Table 1, with METIS dividing web-google into 40 partitions cutting fewer than 1% of the edges. Indeed, the multiple stages of METIS are very well suited to discovering the extremely high quality cuts of such modular graphs. Meanwhile, the highly local nature of the restreaming graphs prevents them from discovering the same high quality partitioning on web graphs that METIS is able to find. On the other hand, the dense local structure of social graphs coupled with the apparent lack of the same modular organization inherent in the web is better suited to the restreaming graphs. As such, all the restreaming algorithms are competitive with METIS on the social graphs, see Figure 1. In particular, restreaming FENNEL performs the best of the restreaming algorithms, out-competing METIS with 0.1% slack on four graphs and even METIS with 3% slack on two graphs. We emphasize that restreaming tempered FENNEL is finding exactly balanced partitions and using only $O(n)$ memory.

Furthermore, over the course of iterating restreams, we observe that both restreaming LDG and FENNEL converge rapidly, and at times exponentially, as seen in Figure 1. This provides an advantageous tradeoff between computational work and the quality of the cut. An exponential convergence rate towards the local optima is consistent with the view that during each streaming pass of the algorithm, nodes are placed permanently in their final position with independent probability p and in a transient position with probability $(1 - p)$. Thus, after r restreams only $(1 - p)^r$ edges remain in a transient assignment. The details of this view are not reflected in the actual microstructure of any of the resteaming results we observe, but we believe that this observation provides a helpful intuition for how restreaming algorithms attempt to correct mistakes from previous iterations.

Note that all the results for FENNEL in this section are for tempered FENNEL, and as such, node balance is only guaranteed at the end of the final iteration, so the flat performance of tempered FENNEL over the course of the many iterations in Figure 1 hides the fact that the algorithm is maintaining the quality of the partitioning while moving towards balance. Indeed, for some graphs, in

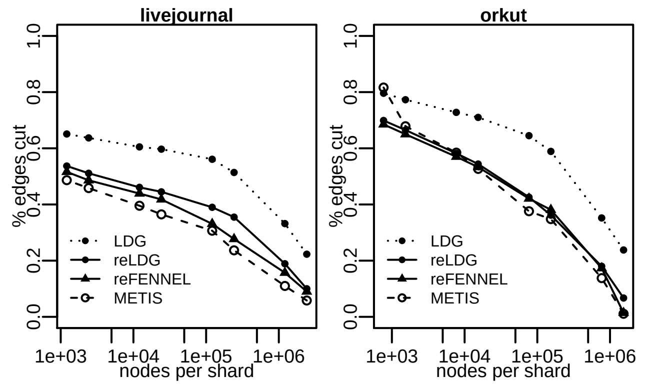

Figure 2: The quality of partitions as a function of the number of nodes per shard, for the LiveJournal and Orkut graphs. Notice how the restreamed algorithms essentially match METIS.

order to achieve balance, restreaming FENNEL must decrease the quality of the partitioning during the tempering process.

As a last look at node balance, we report the results of partitioning two large graphs, LiveJournal and Orkut, into many many partitions. In Figure 2 we observe that our restreaming algorithms match METIS in performance across the full range of partition counts. In fact, when METIS is run with 0.1% slack and $k \geq 200$, the quality of the partitioning deteriorates rapidly, making it significantly worse than the quality of LDG and FENNEL. Since tight slack was not the intended use case for METIS, we report our results for METIS using 1% slack. Still, we see in Figure 2 that when Orkut is divided into 4000 partitions (of roughly 1000 nodes each), the quality decreases markedly. This deterioration in quality does not occur at 4000 partitions if the slack setting for METIS is further increased.

5.2 Tempering

The FENNEL algorithm is only able to achieve high quality cuts with exact balance because restreaming allows for tempering. Figure 3 displays the effect that the initial choice of α_0 has when tempering FENNEL over 20 restreams to the critically stable α, α_c discussed in Proposition 1. Figure 3 shows that when tempering FENNEL, the final quality of the ultimate tempered partitioning is only sensitive to the choice of the initial α_0 when it is very large, but almost entirely insensitive to the choice as long as α_0 is small enough. Note that during a single stream, the choice of α can have a very large impact on both the quality of the partition and the departure from balance, but this importance disappears when

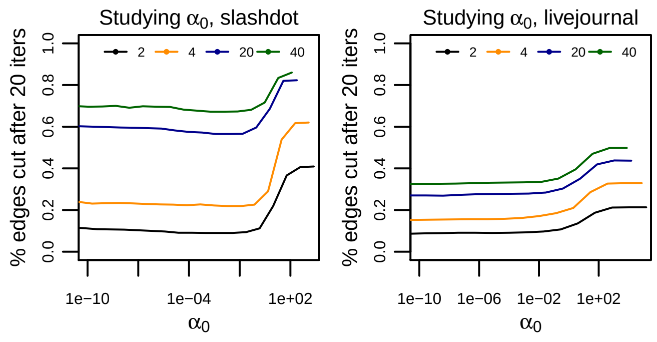

Figure 3: The effect of varying α_0 when tempering, where α_0 is the initial value of α and α is increased to the critical α_c over 20 restreams. For $k = 2, 4, 20, 40$ shards, we see that for sufficiently small α_0 the quality of the edge cut does not depend much on the initial α_0 from which the tempering begins.

tempering. Notice also that this observation appears to apply independently over the number of partitions being sought. This fortunately removes some of the parameter complexity inherent in a single stream of FENNEL while also allowing FENNEL to achieve exact balance.

5.3 Other types of balance

In Section 4 we developed a range of different balance constraints that restreaming partitioning could be adopted towards. Here we report on the quality of the graph cuts obtained when restreaming algorithms are applied towards balance constraints other than node count. A discussion of stratified graph partitioning follows.

In Figure 4, we observe the differences in edge balance and degree counts when running LDG under different constraints: balancing nodes, balancing edges, balancing a sum of the two, or balancing both via the multi-balance multiplicative weights developed in Section 4.1. When either the degree counts or the node counts are left unconstrained, the algorithms clearly utilize the unconstrained flexibility. Thus, in situations where balancing degrees is important, using a method designed to balance nodes would be a poor proxy for the original problem. Indeed, even balancing based on a linear function of nodes and degree fails to balance both. Alternatively, when LDG (and FENNEL, though the results are not shown) are altered to handle multiple constraints, they are able to balance both nodes and degrees for only a small cost in partition quality. The quality of the partitions is seen in Figure 4. It is clear from this figure that stronger notions of balance than just node balance are within reach using simple restreaming algorithms. We now turn our attention to stratified partitioning.

5.4 Stratified balance results

The primary goal in stratified balance is to produce partitions representative of the original degree distribution, such that the nodes of degree d_i in the original graph are split equally between all the partitions. As seen in Figure 5, stratified LDG is able to produce partitions of increasing similarity at a small cost in quality. Since the strata in Figure 5 correspond to separately balancing separate contiguous degree strata, stratified graph partitioning requires that the cumulative degree distributions (CDFs) for all partitions intersect at all strata boundaries. While exactly matching the degree distributions of a graph would require as many strata as there are unique degrees, using only 100 strata produces very similar degree distributions, and even only 10 strata corrects for the majority of the difference.

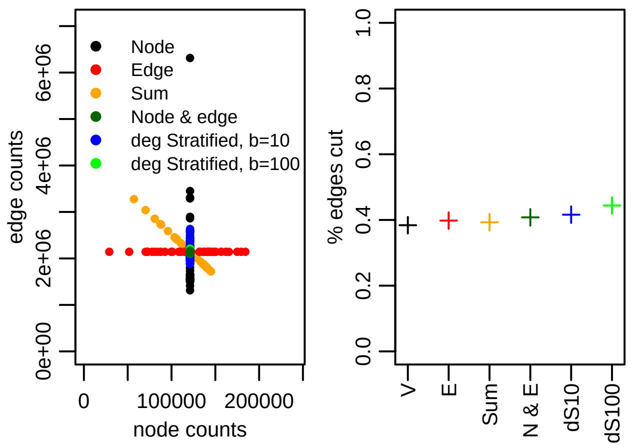

Figure 4: The tradeoffs between node balance and degree balance when LiveJournal is partitioned into 40 different partitions utilizing several different objectives. We see that stronger notions of balance cost very little in partition quality.

Increasing the number of constraints reduces the quality of the partitioning slightly, in a manner similar to multi-constraint METIS. For a large number of strata, multi-constraint METIS and LDG produce partitionings of increasingly similar quality, such that by 100 strata on LiveJournal, METIS and LDG produce edge cuts within 1.5% of each other. Meanwhile, multi-constraints trials executed in METIS required significantly more memory than the corresponding single constraint trials, while stratified LDG only required mildly more time and memory than unstratified LDG.

5.5 Parallel results

Finally we consider the results of parallelizing LDG and FENNEL as discussed in Section 3.5. Figure 6 shows the effect of parallelizing LDG and FENNEL on the LiveJournal graph, partitioning it into 40 different partitions. Note that the first stream of the parallelized version cuts a large percentage of the graph's edges, and it takes longer for these parallelized versions to approach their final quality. However, within less than 20 restreams both algorithms, whether run on 2, 10 and 100 workers, produce partitions of quality comparable to the single thread versions of LDG and FENNEL while only requiring that $\frac{1}{2}$, $\frac{1}{10}$ and $\frac{1}{100}$ of the graph be streamed to each worker respectively. Thus, for a small price in partition quality and an increase in the number of restreams, LDG and FENNEL can be effectively parallelized to many machines. The poor partition quality after the first stream shows that this parallelization strategy can not be applied to single shot streaming partitioning, and that restreaming plays an important role in enabling parallelization.

6. CONCLUSION

Given the enormous sizes of social and web graphs it is increasingly important to carefully navigate the fundamental tradeoff between the quality of a graph partition and the memory and computational requirements to compute it. To address this tradeoff, we introduce the problem of *restreaming graph partitioning* and develop two algorithms that iteratively partition graphs using only the same $O(n)$ memory required in single pass streaming graph partitioning. Surprisingly, our results demonstrate that these restreaming algorithms are able to close much of the distance between streaming graph partition algorithms and full offline graph partitioning optimization suites—at times even outperforming them. The competitiveness of these streaming graph partitions is particularly no-

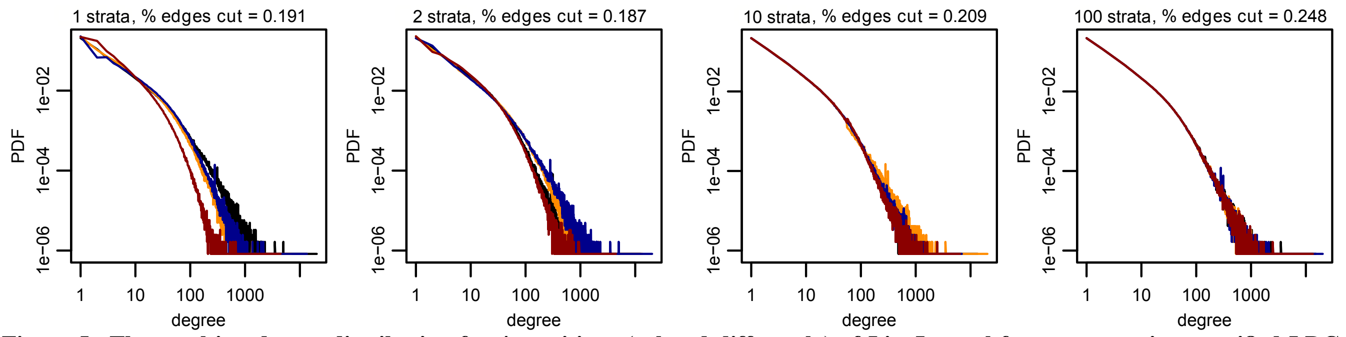

Figure 5: The resulting degree distribution for 4 partitions (colored differently) of LiveJournal from restreaming stratified LDG where portions of the degree distribution are explicitly balanced across 1, 2, 10 and 100 different stratified strata. As the number of strata increases the degree distributions become increasingly similar, though at a small cost in the quality of the edge cut.

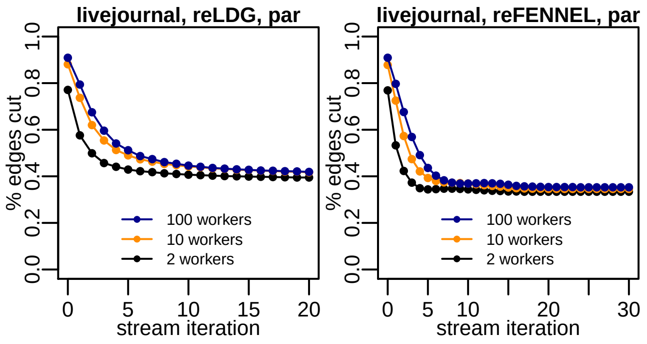

Figure 6: The percentage of edges cut for parallelized versions of FENNEL and LDG when partitioning LiveJournal into 40 partitions while parallelized across 2, 10 and 100 machines. Notice that while there is a small cost associated with increasing the number of *machines*, it is small compared to the gains of restreaming.

ticeable on social graphs. Furthermore, while restreaming graph partitioning preserves the same small memory footprint as single shot streaming algorithms, restreaming allows for true parallelization, with communication between workers only between streams.

The simplicity and effectiveness of these algorithms allows for their easy modification to a number of more complex objectives. In particular we introduce the problem of *stratified graph partitioning* as a way of creating partitionings where the composition of each partition resembles the composition of the graph as a whole. Despite the significant increase in constraints in stratified graph partitioning, simple modifications to a restreaming algorithm allows for the partitioning of a large social graph such that each partition has the same degree distribution. This particular application addresses a fundamental question in the design of test groups on social graphs.

Acknowledgements. We thank Jon Kleinberg, Isabel Kloumann, and Justin Vincent for helpful comments. This work was supported in part by NSF grants IIS-0910664 and IIS-1016099.

7. REFERENCES

[1] `http://snap.stanford.edu/data/`.

[2] K.J. Ahn, S. Guha, and A. McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *PODS*, p. 5–14, 2012.

[3] M. Barber and J. Clark. Detecting network communities by propagating labels under constraints. *Phys Rev E*, 80(2):026129, 2009.

[4] V. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *J of Statistical Mechanics: Theory and Experiment*, 2008(10):10008, 2008.

[5] S. P. Borgatti and M. G. Everett. Models of core/periphery structures. *Social Networks*, 21(4):375–395, 2000.

[6] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On compressing social networks. In *KDD*, p. 219–228, 2009.

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

[8] S. E. Fienberg. A brief history of statistical models for network analysis and open challenges. *Journal of Computational and Graphical Statistics*, 21(4):825–839, 2012.

[9] M. R. Garey and D. S. Johnson. *Computers and intractability*, 1979.

[10] B. H. Good, Y.-A. de Montjoye, and A. Clauset. Performance of modularity maximization in practical contexts. *Phys Rev E*, 81:046106, 2010.

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

[12] G. Karypis and V. Kumar. Parallel multilevel series k-way partitioning scheme for irregular graphs. *SIAM Review*, 41(2):278–300, 1999.

[13] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 1970.

[14] A. Kyrola, G. Blelloch, and C. Guestrin. Graphchi: Large-scale graph computation on just a PC. In *OSDI*, 2012.

[15] J. Leskovec, K.J. Lang, A. Dasgupta, and M.W. Mahoney. Statistical properties of community structure in large social and information networks. In *WWW*, p. 695–704, 2008.

[16] X. Liu and T. Murata. Advanced modularity-specialized label propagation algorithm for detecting communities in networks. *Physica A*, 389(7):1493–1500, 2010.

[17] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *IMC*, 2007.

[18] J. Neville, B. Gallagher, T. Eliassi-Rad, and T. Wang. Correcting evaluation bias of relational classifiers with network cross validation. *Knowledge and Info Sys*, p. 1–25, 2012.

[19] M.E.J. Newman. Modularity and community structure in networks. *PNAS*, 103(23): 8577–8582, 2006.

[20] D. A. Spielman and S.-H. Teng. Spectral partitioning works: planar graphs and finite element meshes. In *FOCS*, p. 96–105, 1996.

[21] I. Stanton. Streaming balanced graph partitioning for random graphs. *arXiv preprint arXiv:1212.1121*, 2012.

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

[23] C. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic. Fennel: Streaming graph partitioning for massive scale graphs. *Microsoft Technical Report MSR-TR-2012-113*, 2012.

[24] J. Ugander and L. Backstrom. Balanced label propagation for partitioning massive graphs. In *WSDM*, 2013.

[25] J. Ugander, B. Karrer, L. Backstrom, and J. Kleinberg. Graph cluster randomization: Network exposure to multiple universes. In *KDD*, 2013.

[26] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. The anatomy of the facebook social graph. *arXiv preprint arXiv:1111.4503*, 2011.