Streaming Balanced Graph Partitioning Algorithms for Random Graphs

Isabelle Stanton
Google Inc
stanton@google.com

Abstract

With recent advances in storage technology, it is now possible to store the vast amounts of data generated by cloud computing applications. The sheer size of '*big data*' motivates the need for streaming algorithms that can compute approximate solutions without full random access to all of the data.

In this paper, we consider the problem of loading a graph onto a distributed cluster with the goal of optimizing later computation. We model this as computing an approximately balanced k-partitioning of a graph in a streaming fashion with only one pass over the data. We give lower bounds on this problem, showing that no algorithm can obtain an $o(n)$ approximation with a random or adversarial stream ordering. We analyze two variants of a randomized greedy algorithm (looking at the distribution of edges from the vertex to be assigned, one prefers the partition that is the $\arg\max$ and one that assigns the vertex proportional to the edge distribution) on random graphs with embedded balanced k-cuts and are able to theoretically bound the performance of each algorithms - the $\arg\max$ algorithm is able to recover the embedded k-cut while the proportional variant can not. This matches the experimental results in [30].

1 Introduction

Recent advances in storage technology and distributed computing have created the phenomenon known as *big data*. There are several paradigm shifts involved in the big data movement. From a theoretical perspective, one is that traditional assumptions like full uniform random access to the data are no longer reasonable. This shift motivates the study of streaming algorithms.

One type of big data is graphs. There has been a lot of interest in distributed graph computation systems from many different communities - systems builders, database experts, and machine learning. This has led to the creation of a huge number of such systems including GraphLab [20], Pregel [22], Horton [28], Spark [33], Trinity, and the filtering technique for MapReduce to name but a few. In terms of distributing data, some of these systems support custom partitionings but the vast majority use a hashing method to produce a random cut as the default partitioning. From a systems perspective, this approach makes sense - lookup is fast and easy to maintain. However, network is far slower than local communication between processor cores. A random cut on a graph is a good approximation to the MAXCUT problem and is the exact opposite of what one should do to minimize communication volume. Even marginal improvements in the partitioning can lead to large improvements in run time for distributed algorithms [30].

The communication problem is a major motivation for studying graph partitioning. The constraints of distributed computing and the fact that, when loading, the graph data arrives as a stream means that traditional graph partitioning algorithms that assume full access to the data are no longer scalable. Instead, our goal is to generate an approximately balanced k-partitioning of a graph, using a streaming algorithm with only one pass over the data as this models partitioning a graph while loading it onto a cluster.

Previous work addressed this problem from an experimental perspective. [30] evaluates 16 different partitioning heuristics on 21 different graphs to find how well each performs when compared with an offline partitioning heuristic (METIS [16]). A greedy algorithm assigns a vertex to the partition where it currently has the most edges. Surprisingly, a simple variant of greedy performed the best, even beating an adaptation of a local partitioning algorithm, EvoCut [3]. Also surprising was that adding randomization to the same algorithm caused it to perform significantly worse. Often, the addition of randomness allows us to design more effective algorithms, not less. In this paper, we seek to provide a theoretical foundation for understanding these results and motivate further study into more sophisticated algorithms.

Since the publication of [30], this model of streaming graph partitioning and the simple algorithms suggested have become quite interesting to the systems community. As of this writing, it has been implemented in several systems, including PowerGraph [14], GPS [27], xDGP [32], as well as extended to a realistic multi-pass setting [25].

Contributions This paper focuses on developing a rigorous understanding of two greedy streaming balanced graph partitioning algorithms. We give lower bounds on the approximation ratio that any streaming algorithm for balanced graph partitioning can obtain on both a random and adversarial ordering of the graph. In response to this lower bound, we focus our attention on a class of random graphs with embedded balanced k cuts. We analyze our greedy algorithms by using a novel coupling to finite Polya Urn processes. This very elucidating connection gives clear intuition as to why one algorithm performs well while the other does not. We include an experimental evaluation of the bounds attained by the theorems.

2 Notation and Definitions

We now introduce the notation and definitions. The *balanced graph partitioning problem* takes as input a graph G, an integer k and an allowed imbalance parameter of ϵ. The goal is to partition the vertices of G into k sets, each no larger than $(1+\epsilon)\frac{n}{k}$ vertices, while minimizing the number of edges cut.

Graph Models A graph $G = (V, E)$ consists of $n = |V|$ vertices and $m = |E|$ edges. $\Gamma(v)$ is the set of vertices that a vertex v neighbors. We consider graphs generated by two random models. The first, $G(n, p)$ is the traditional Erdös-Renyi model with n vertices. The traditional definition is that each of the possible $\binom{n}{2}$ edges is included independently with probability p. At certain points in the proofs in Section 5, we modify this definition to make it better match our streaming model. In particular, we allow multiple edges in order to maintain independence in our analysis.

$G(\Psi, P)$ is a generalization of $G(n, p)$, due to McSherry [23], that allows the graph to have l different Erdös-Renyi components, each with different parameters. Again, we have n vertices. $\Psi : \{1, 2, \ldots n\} \rightarrow$

$\{1, 2, \dots l\}$ is a function mapping the vertices into l disjoint clusters. Let C_i refer to the set of vertices mapped to i, i.e. $\Psi^{-1}(i) = C_i$. P is a $l \times l$ matrix where edges between vertices in C_i are included independently with probability $P_{i,i}$ and edges between vertices in C_i and C_j are included with probability $P_{i,j}$. There are many ways for $G(\Psi, P)$ to generate graphs in $G(n, p)$ - Ψ could map all vertices into the same cluster or we could have $P_{i,i} = P_{i,j} = p$ for all i, j. We make the same modification to the generative process as in $G(n, p)$ and allow multiple edges for clarity of the analysis.
Probability Distributions We only use variables drawn from a binomial distribution, where $X \sim B(n, p)$ is a random variable representing n independent trials, each with probability p of success.

2.1 Polya Urn Processes

The classical Polya Urn problem is: Given finitely many initial bins, each containing one ball, let additional balls arrive one at a time. For each new ball with probability p create a new bin and put the ball in it. With probability $1 - p$, place the ball in an existing bin with probability proportional to m^γ where m is the number of balls currently in that bin.

Many variants of the above process have been analyzed. In particular, Chung, Handjani, and Jungreis [8] analyze the *finite Polya urn process* where $p = 0$. The exponent γ plays an important role in the behavior of this process. With k bins, when $\gamma < 1$, in the limit, the load of each bin is uniformly distributed and each contains a $\frac{1}{k}$ fraction of the balls. When $\gamma > 1$, in the limit, the fractional load of one bin is 1. When $\gamma = 1$, the limit of the fractional loads exists but is distributed uniformly on the simplex.

Our proof technique focuses on connecting the streaming graph partitioning algorithms with the finite Polya urn process and use many of the results from [8]. We restate the results used here:

Theorem 1 (Theorem 2.1 [8])**.** *Consider a finite Polya process with exponent $\gamma = 1$, k bins and let x_i^t denote the fraction of balls in bin i at time t. Then a.s. $\forall i$, $X_i = \lim_{t\to\infty} x_i^t$ exists. Furthermore these limits are distributed uniformly on the simplex $\{(X_1, X_2, \dots X_k) : X_i > 0, X_1 + X_2 + \dots + X_k = 1\}$.*

Theorem 2 (Theorem 2.2 [8])**.** *Consider a finite k-bin Polya process with exponent γ and let x_i^t denote the fraction of the balls in bin i at time t. Then a.s. the limit $X_i = \lim_{t\to\infty} x_i^t$ exists for each i. If $\gamma > 1$ then $X_i = 1$ for one bin, and $X_i = 0$ for all others. If $\gamma < 1$ then $X_i = \frac{1}{k}$ for all bins.*

Lemma 1 (Lemma 2.3 [8])**.** *Given a Polya process with exponent γ and an arbitrary initial configuration (i.e. finitely many balls arranged in finitely many bins), we restrict attention to any subset of the bins and ignore any balls that are placed in the other bins. Then the process behaves exactly like a finite Polya process with exponent γ on this subset of bins, though the process may terminate after finitely many balls.*

Lemma 1 is particularly important to our analysis as it forms the basis of an inductive argument to extend the analysis in [8] to k bins from 2 bins. We also use the claim that a finite, arbitrary initial configuration does not affect the distribution in the limit.

2.2 The Streaming Model

Our streaming model definition is motivated by the systems problem of loading data onto a cluster of machines and borrows heavily from both the database community's' definition of streaming, and the literature on online algorithms. The particular constraints this adds that because we are loading the data onto machines, a second streaming pass over the data makes little sense - the data is already loaded so we can run a more powerful distributed algorithm. Similarly, it makes more sense to move data around after it is completely loaded and we have access to more information. Finally, we are (mostly) in control of the access pattern for the data - it may be coming from a webcrawler or streamed from disk in an order that we have selected. Also, we may have megabytes worth of data to store about each node so it is reasonable to assume that one machine can hold the $2n \log n$ bits required for a partition look up table (or maintain a distributed hash table).

In our model, we are able to select the order that the vertices arrive. For the paper, we primarily consider a random ordering but also look at an adversarial ordering for the lower bounds. This is done for analysis reasons. For n vertices, the set of permutations S_n defines all possible orderings. For a random ordering, each permutation is picked with equal probability. An adversarial ordering is any probability distribution over the permutations, including one that picks the worst possible ordering for the algorithm.

Our graphs are discovered by a local search method or 'crawler', so when a vertex arrives so do its incident edges. However, the partitioner does not have access to store all previous edges seen - while all edges arrive with a vertex, we can only use them for partitioning when the second endpoint arrives. We generate an approximately balanced vertex partitioning of the graph with k partitions. The capacity of each partition, C, is enough to hold all the vertices, i.e. $kC = (1+\epsilon)n$. Our evaluation metric is the number of edges cut. This is not affected by the directionality of an edge, so we assume undirected.

In summary, our major additional constraints are that we assume that only one pass can be made over the data, the algorithm has access to both the current load of each machine on the cluster and the location of each vertex that has been previously seen, and a vertex is not moved after it has been placed into some partition. These assumptions directly come from the design of modern distributed systems and are motivated observing that a distributed algorithm is significantly more powerful than a streaming one.

3 Related Work

Many variants of graph partitioning have been studied since the earliest days of Computer Science. The variant considered in this paper, balanced k-cut, has been shown to be NP-hard by Andreev and Rắcke [4], even when one relaxes the balance constraint on the partition sizes. They also give an LP-based solution that obtains an $O(\log n)$ approximation. Another full-information solution was found by Even et al. who use an LP solution based on spreading metrics to also obtain an $O(\log n)$ approximation algorithm [13]. If one ignores the balance constraint, a popular approach is to use the top k eigenvectors [24]. Recently, this approach was theoretically validated as an extension of Cheeger's inequality [18, 19]. One can also use any balanced 2-partitioning algorithm to obtain an approximation to a balanced k-partitioning when k is a power of 2, losing at most an additional $\log n$ factor [5].

From a heuristic perspective, there are numerous full information graph partitioning systems available that do not have theoretical performance guarantees. These include METIS [16], PMRSB [7], and Chaco [15].

Another approach, relevant for our limited information setting, is *local partitioning* algorithms. The goal here is not to obtain a balanced cut but given a starting node to find a good cut around that node. Spielman and Teng were the first to develop this style of algorithm [29]. Anderson, Chung and Lang improved upon Spielman and Teng's work by using personalized PageRank vectors to find a good local cut [2]. Addressing the same problem, Anderson and Peres use the evolving graph process to obtain similar results [3]. While local partitioning is similar in spirit, it is not the same as a streaming algorithm.

The main focus of this paper is on streaming algorithms and there is significant related work in this area as well. First, noting the connection between graph partitioning and PageRank is Das Sarma et al.'s work on computing the PageRank of a graph with multiple passes [11]. Closer to our setting, Bahmani et al. incrementally compute an approximation of the PageRank vector with only one pass [6]. However, just computing the approximate PageRank vector is not sufficient for finding a graph partitioning with only one pass over the data. Das Sarma et al. extend their techniques to find sparse cut projections within subgraphs, again using multiple passes over the stream [10]. Cut projections are not the same as finding balanced cuts.

An alternate model, *semi-streaming*, assumes that we have $O(n\text{poly}\log n)$ storage space so that all vertices can be stored but the edges arrive in some order. In this setting, Ahn and Guha [1] give a one pass $\tilde{O}(n/\epsilon^2)$ space algorithm that sparsifies a graph such that each cut is approximated to within a $(1+\epsilon)$ factor. Kelner and Levin [17] produce a spectral sparsifier with $O(n \log n/\epsilon^2)$ edges in $\tilde{O}(m)$ time. While sparsifiers are a great way of reducing the size of the data, this reduction would then require an additional pass over the data to compute a partitioning which is out of the scope of the problem at hand. Finally, lower bounds are known with regards to the space complexity of both the problem of finding a minimum and maximum cut. Zelke [34] has shown that this cannot be computed in one pass with $o(n^2)$ space.

Finally, analyzing algorithms on random graph models has a long history. In particular, it is quite common to analyze graph partitionings on random graphs with planted partitions [23, 21]. This is done because recovering a planted partition is equivalent to finding the 'right' answer. Closest to the spirit of this work is the Condon-Karp algorithm [9]. They analyze a simple randomized greedy algorithm on a planted partition model and show that with high probability, their algorithm can recover a planted l-partition.

However, their algorithm does not fit into our framework as it uses two passes over the data to generate a bipartition and recursion to extend the first bipartition into an l-partition. In this paper, we trade off the multiple passes for a stronger dependence on the gap between p and q. Another partitioning system, Fennel [31], has been developed using heuristics inspired by Condon-Karp with great success.

4 Lower Bounds

The first important question is whether any algorithm can do well on all graphs in our streaming model. The unfortunate answer is no. Intuitively, with only one pass, important edges may be hidden either intentionally by an adversary or unintentionally by randomness. The proofs are fairly simple and involve only considering a cycle graph. They are included in the Appendix.

Theorem 3. *One-pass streaming balanced graph partitioning with an adversarial stream order can not be approximated within $o(n)$.*

Theorem 4. *One-pass streaming balanced graph partitioning with a random stream order can not be approximated within $o(n)$.*

5 Analysis of Algorithms on Random Graphs

The experiments in [30] showed that one heuristic, Linear Deterministic Greedy (LDG), was clearly the best tried. However, Linear Randomized Greedy(LRG), differs only by selecting a partition proportionally to the distribution of edges instead of from the maxima, yet performed much worse than LDG. This raises the question - can we theoretically explain the difference in performance? In this section, we introduce slightly simpler variants, arg max Greedy (LDG) and Proportional Greedy (LRG), and analyze their performance on McSherry's random graph model. Our analysis clearly explains the difference observed in the experiments.

5.1 Algorithms

The two algorithms studied in this paper are very similar: when a vertex v arrives, a score for each partition P_i of the number of edges from v to P_i, $S_i = |\Gamma(v) \cap P_i|$, is calculated. If the partition is full, its score is set to 0. If all scores are 0, then the vertex is assigned to some partition with minimal load. If a score is non-zero, then the arg max Greedy Algorithm assigns the vertex uniformly at random to a partition in $\arg\max S_i$. By contrast, the Proportional Greedy Algorithm uses the scores as a distribution, assigning the vertex to partition i with probability $S_i / \sum S_j$. The versions of these algorithms from [30] differ only in that the score for each partition is weighted by the current load of the partition, i.e. $S_i(1 - \frac{|P_i|}{C})$. In practice, the algorithms keep the partitions nearly balanced, meaning this tiebreaker is only used in cases of tied number of edges and when there are no edges where [30] prefers the least-loaded partitions.

Algorithm 1 arg max Greedy

```
Input: $G$, $k$, $C$, $\pi$
  $P_1, \cdots, P_k = \emptyset$
  for $t = 1, 2, \ldots n$ do
    for $i = 1, 2, \ldots k$ do
      $S_i = |\Gamma(\pi(t)) \cap P_i|$
      if $|P_i| = C$ then
        $S_i = 0$
    if all $S_i = 0$ then
      Pick $i$ from $\arg\min_{j \in [k]}\{|P_j|\}$ u.a.r.
    else
      Pick $i$ from $\arg\max_{j \in [k]}\{S_j\}$ u.a.r.
    $P_i = P_i \cup \pi(t)$
```

Algorithm 2 Proportional Greedy

```
Input: $G$, $k$, $C$, $\pi$
  $P_1, \cdots, P_k = \emptyset$
  for $t = 1, 2, \ldots n$ do
    for $i = 1, 2, \ldots k$ do
      $S_i = |\Gamma(\pi(t)) \cap P_i|$
      if $|P_i| = C$ then
        $S_i = 0$
    if all $S_i = 0$ then
      Pick $i$ from $\arg\min_{j \in [k]}\{|P_j|\}$ u.a.r.
    else
      Pick $i$ proportional to $S_i$
    $P_i = P_i \cup \pi(t)$
```

One of the key insights of this paper is that when these algorithms are used on random graphs, we can write both down as random processes. In particular, we can let the random process generate the graph while also partitioning it at the same time. This reduction is discussed in Section 5.3. The proof proceeds by analyzing the random process versions of the algorithms, rather than those given in Algorithms 1 and 2.

The random processes generate a multi-edge $G(n,p)$ graph. For the extended $G(\Psi, P)$ analysis, we only consider Algorithm 1 and Algorithm 3 with the correct modification of $E_i^{(t)}$.

Algorithm 3 arg max Greedy Process on $G(n,p)$

```
Input: $p$
  Set $P_1, P_2, \dots P_k = \emptyset$
  for $t = 1, 2, \dots n$ do
    For $1 \le i \le k$, draw $E_i^{(t)} \sim B(|P_i|, p)$
    if $\sum_{i=1}^k E_i^{(t)} = 0$ then
      Assign $t$ to $\arg\min_{j \in [k]}\{|P_j|\}$
    else
      Assign $t$ to $\arg\max_{j \in [k]}\{E_j^{(t)}\}$
```

Algorithm 4 Proportional Greedy Process on $G(n,p)$

```
Input: $p$
  Set $P_1, P_2, \dots P_k = \emptyset$
  for $t = 1, 2, \dots n$ do
    For $1 \le i \le k$, draw $E_i^{(t)} \sim B(|P_i|, p)$
    if $\sum_{i=1}^k E_i^{(t)} = 0$ then
      Assign $t$ to $\arg\min_{j \in [k]}\{|P_j|\}$
    else
      Assign $t$ to $P_i$ with prob. $E_i^{(t)} / \sum_{j=1}^k E_j^{(t)}$
```

5.2 Result and Proof Outline

We now focus on proving the following two statements. The first is that the Proportional Greedy Algorithm can not recover an embedded partition in a $G(\Psi, P)$ graph, no matter the parameters or size of the graph. The second result is that the arg max Greedy Algorithm can recover the embedded partition, provided the components are dense enough, the cut between them is sparse enough, and there are enough components.

Theorem 5. *Given a $G(\Psi, P)$ graph with $P_{i,i} = p$, $P_{i,j} = q$ and $l > k \log k$ equally sized components $|C|$, where $p > \frac{2 \log n}{|C|}$, $p > 3(k + \sqrt{k} + 1)lq$, and $q = O((k^{2.4} \log l)^{-1})$, arg max Greedy Algorithm will recover an embedded partition from a random stream ordering.*

The proof proceeds in several stages. First, we ignore the capacity constraint and consider Algorithms 3 and 4 on a single $G(n,p)$ component. Does the algorithm eventually learn it is a component and place it in the same partition? We show that Algorithm 4 is equivalent to a finite Polya urn process with $\gamma = 1$ and distributes the component over all the partitions. However, Algorithm 3 can be coupled to a finite Polya urn process with $\gamma > 1$. It will asymptotically place the entire $G(n,p)$ component in one partition. This argument starts with 2 partitions and is extended to k bins using an induction argument.

That Algorithm 3 will correctly (not) partition a connected component forms the basis of our argument that it can be extended to the $G(\Psi, P)$ model. Intuitively, with the correct parameters, each component of $G(\Psi, P)$ is placed in a single partition. The primary technical difficulties faced are the inclusion of the capacity constraint, requiring bounds on the component sizes, and the addition of intra-cluster edges, which serve to 'confuse' the algorithm about to which component a vertex belongs. By setting the parameters of the model correctly, we can overcome these challenges.

5.3 Analysis on a Single $G(n,p)$ Component

Algorithms 3 and 4 are obtained from Algorithms 1 and 2 by considering the process in terms of Polya urns. The finite Polya urn process has k bins and the t^{th} ball is assigned to bin i with probability proportional to $(m_i^{(t)})^\gamma$ where $m_i^{(t)}$ is the load of the i^{th} bin at time t. Translating Algorithm 1 and 2 to Polya Urn processes involves identifying each ball with a vertex and each bin with a partition. There are two differences from the standard Polya Urn process. First, with prob. $(1-p)^t$, the t^{th} vertex (ball) does not have edges to vertices already seen and is placed in the least loaded partition (urn). The second is we do not assign the vertex (ball) based on the load of the partition (urn) but on a binomial random variable based on the load. Specifically, $E_1^{(t)}, \dots E_k^{(t)}$ are the random variables representing the number of edges to each of the k partitions. Each $E_i^{(t)}$ is drawn from $B(m_i^{(t)}, p)$. The following connection is how we created Algorithms 3 and 4.

- Algorithm 1 assigns the vertex to a partition in $\arg\max_{j \in [k]}\{E_j^{(t)}\}$, breaking ties at random.
- Algorithm 2 assigns it to bin i proportional to $E_i^{(t)}$

Algorithm 4 Analysis. The total number of edges from vertex t is a random variable $E^{(t)} \sim B(t, p)$. Each edge is distributed according to $m_i^{(t)}$ i.e. with prob. $\frac{m_i^{(t)}}{t}$ it connects to the i^{th} bin. Each of the $E^{(t)}$ edges are i.i.d. and are given equal weight so Algorithm 2 assigns balls proportional to $(m_i^{(t)})^\gamma$ where $\gamma = 1$.

Theorem 6. ***Algorithm 4 on*** $G(n,p)$ *Let* $0 \leq p < 1$*. Let* x_i^t *be the fractional load of partition i at time t of Algorithm 4. Then almost surely* $\lim_{t\to\infty} x_i^t = X_i$ *exists and for all i,* $X_i > 0$*.*

Proof. When there are edges, this process is exactly a finite Polya urn process with $\gamma = 1$. The result then follows directly from Theorem 1. Let there be k bins. At time t, each has load $m_i^{(t)}$. Let $E^{(t)}$ be the total number of edges drawn by the process. Assume $E^{(t)} > 0$ as $E_t^{()} = 0$ is dealt with later. We allow multiple edges in our model, so consider the edges being distributed to the k partitions with replacement, i.e. each of the $E^{(t)}$ edges goes to partition i with probability $\frac{m_i^{(t)}}{t-1}$. Let $E_i^{(t)}$ be the number to partition i. Note that $\sum_{i=1}^k E_i^{(t)} = E^{(t)}$. Now $\Pr[\text{Algorithm 4 picks bin } i] = E_i^{(t)}/E^{(t)}$. However, $E_i^{(t)} \sim B(E^{(t)}, \frac{m_i^{(t)}}{t-1})$, showing that this assignment is proportional to $m_i^{(t)}$ as desired. This is exactly a finite Polya urn process with $\gamma = 1$.

The remaining detail is the modification of the process when $E^{(t)} = 0$ when the algorithm assigns the vertex to the least loaded bin. If this situation has a constant probability throughout the process, then it makes the distribution of the balls more uniform, satisfying the theorem statement that all bins contain a non-zero fraction of the balls. If it becomes unlikely as the process progresses, i.e. $p > \frac{\log n}{n}$, then we apply Theorem 1 and Lemma 1 from [8] to say that after $O(\frac{\log n}{p})$ vertices have arrived, we begin the $\gamma = 1$ Polya Urn process with an arbitrary finite initial configuration. From Theorem 1, we get that $X_i > 0$ for all i. $\square$

Thus, the randomized algorithm does not have a concentration result - for a $G(n,p)$ component ($p < 1$) the Proportional Greedy algorithm does not learn that it is a component and distributes it over all partitions.

Corollary 1. *Given a single isolated* $G(n,p)$ *component, for any value p, Algorithm 2 distributes this component over all k partitions.*

Algorithm 3 Analysis. The key insight about why Algorithm 3 has a concentration result is that by preferring the $\arg\max$, once some partition has a slightly higher load than the other, it is very likely to be assigned the next vertex. As the gap in the loads grow, the larger partition becomes more likely to receive the next vertex until it is impossible for the smaller partition to compete. However, there are a few challenges.

First, with probability $(1-p)^t$, the $t+1^{th}$ vertex does not have any edges to previously seen vertices. In this case, it is placed in the least loaded bin, decreasing the gap in the loads. If it happens too often, the gap does not grow. Since $(1-p)^t \approx e^{-pt}$, once $t = O(\frac{\log n}{p})$, this does not happen w.h.p., provided $p > \frac{\log n}{n}$. We only expect $\frac{1}{p}$ vertices to arrive with no edges and they are concentrated when $t < \frac{1}{p}$.

The second challenge is that when the vertex has 1 edge, the $\arg\max$ distribution is the same as Algorithm 4. However, this can be dealt with in the same manner as having no edges. Again, we expect $\frac{1}{p}$ vertices to have only 1 edge and primarily when $\frac{1}{p} \leq t \leq \frac{2}{p}$. Therefore, we need $p > \frac{2\log n}{n}$.

The final challenge is that we can not couple Algorithm 3 to a finite Polya urn process with $\gamma > 1$ until $\frac{2}{p}$ vertices have arrived, meaning we do not start with a uniform load distribution. Lemma 1 shows that we can start with an arbitrary finite initial configuration and obtain the same concentration results.

Theorem 7. *Let p be any value between* $\frac{2\log n}{n}$ *and 1. Let* x_i^t *be the fractional load of partition i at time t of Algorithm 3. Then almost surely* $\lim_{t\to\infty} x_i^t = X_i$ *exists and one* $X_j = 1$*, while all others are 0.*

This statement follows from Theorem 2. Our analysis for Algorithm 3 relies on the probability that bin i receives a ball at time t or $\Pr\left[E_i^{(t)} = \arg\max_{j\in[k]}\{E_j^{(t)}\}\right]$ for $E_i^{(t)} \sim B(m_i^{(t)}, p)$. It is intuitive that bins with a higher load should have a much higher probability of being the $\arg\max$, yet the binomial distribution does not have a nice closed form expression for $\Pr[X \geq k]$. Even if we condition on $E^{(t)} = \sum_{i=1}^k E_i^{(t)} = x$ so we can express the $E_i^{(t)}$ as a multinomial distribution, a nice closed form solution eludes us. Therefore, our proof consists of several lemmas.

Lemma 2. *Given a $G(n,p)$ graph with $p > \frac{2\log n}{n}$, after $O(\frac{\log n}{p})$ steps, Algorithm 3 with 2 partitions can be coupled to a finite Polya urn process with $\gamma > 1$.*

Proof. Let $A = E_1^{(t)}$ and $B = E_2^{(t)}$ and A^j, B^j be the loads conditioned on $E^{(t)} = j$ i.e. $A^j + B^j = j$. Let δ be the comparative advantage of A over B, i.e. $\frac{1}{2} + \delta = \frac{m_1^{(t)}}{t}$ and $\frac{1}{2} - \delta = \frac{m_2^{(t)}}{t}$. We analyze $\Pr\left[A^j > B^j\right]$.

$$\Pr\left[A^j > B^j\right] = \sum_{i=\lfloor j/2\rfloor+1}^{j} \binom{j}{i} (\frac{1}{2}+\delta)^i (\frac{1}{2}-\delta)^{j-i} = (\frac{1}{2}+\delta)^{\lfloor j/2\rfloor+1} \sum_{i=\lfloor j/2\rfloor+1}^{j} \binom{j}{i} (\frac{1}{2}+\delta)^{i-\lfloor j/2\rfloor-1} (\frac{1}{2}-\delta)^{j-i}$$

$$= (\frac{1}{2}+\delta)^{\lfloor j/2\rfloor+1} \sum_{i=0}^{\lfloor j/2\rfloor} \binom{j}{i-\lfloor j/2\rfloor} (\frac{1}{2}+\delta)^i (\frac{1}{2}-\delta)^{j-\lfloor j/2\rfloor-1-i} = (\frac{1}{2}+\delta)^{\lfloor j/2\rfloor+1} \sum_{i=0}^{\lfloor j/2\rfloor} \binom{j}{i} (\frac{1}{2}+\delta)^{\lfloor j/2\rfloor-i} (\frac{1}{2}-\delta)^i$$

We similarly express $\Pr\left[B^j > A^j\right]$ as follows.

$$\Pr\left[B^j > A^j\right] = \sum_{i=\lfloor j/2\rfloor+1}^{j} \binom{j}{i} (\frac{1}{2}-\delta)^i (\frac{1}{2}+\delta)^{j-i} = (\frac{1}{2}-\delta)^{\lfloor j/2\rfloor+1} \sum_{i=0}^{\lfloor j/2\rfloor} \binom{j}{i} (\frac{1}{2}-\delta)^{\lfloor j/2\rfloor-i} (\frac{1}{2}+\delta)^i$$

Because $\frac{1}{2}+\delta > \frac{1}{2}-\delta$, we have that $\sum_{i=0}^{\lfloor j/2\rfloor} \binom{j}{i}(\frac{1}{2}+\delta)^{\lfloor j/2\rfloor-i}(\frac{1}{2}-\delta)^i > \sum_{i=0}^{\lfloor j/2\rfloor} \binom{j}{i}(\frac{1}{2}-\delta)^{\lfloor j/2\rfloor-i}(\frac{1}{2}+\delta)^i$. Therefore,

$$\Pr\left[A^j > B^j\right] > \frac{(\frac{1}{2}+\delta)^{\lfloor j/2\rfloor+1}}{(\frac{1}{2}-\delta)^{\lfloor j/2\rfloor+1}} \Pr\left[B^j > A^j\right].$$

From this, and the fact that these two quantities sum to 1, we conclude that

$$\Pr\left[A^j > B^j\right] > \frac{(\frac{1}{2}+\delta)^{\lfloor j/2\rfloor+1}}{(\frac{1}{2}+\delta)^{\lfloor j/2\rfloor+1} + (\frac{1}{2}-\delta)^{\lfloor j/2\rfloor+1}}$$

This lower bound is the probability that the ball goes in urn 1 in a Polya process with $\gamma = \lfloor j/2\rfloor + 1$. When $j \geq 2$, we can couple our process to a finite Polya urn process with a desirable concentration result. We remove the conditioning on $E^{(t)} = j$ to get $\Pr[A > B]$

$$\Pr[A > B] = \sum_{j=1}^{t} \binom{t}{j} p^j (1-p)^{t-j} \Pr\left[A^j > B^j\right] \tag{1}$$

The only case where we are mixing in a process that has an undesirable exponent ($\gamma = 1$) is when $j = 0$ or 1. The probability of this case is less than $\frac{1}{n}$ when $t > \frac{2\log n}{p}$. According to Lemma 1, this constitutes a finite arbitrary configuration and the concentration results hold after $t > \frac{2\log n}{p}$. □

This proof shows that the algorithm can eventually be coupled with a finite Polya urn process with $\gamma > 1$. Lemma 1 shows that the initial configuration when the process takes off does not affect the concentration results. Moreover, we bound the total expected number of vertices to arrive with $j = 0$ or 1 by

$$\frac{1-(1-p)^n + 1 - p}{p} \approx \frac{2 - e^{-pn} - p}{p} \leq \frac{2}{p}.$$

Combining Lemma 1 and 2 shows that for 2 partitions Algorithm 3 concentrates the process into 1 bin. In order to extend the process to k partitions, we present the following Lemma. It follows the proof technique of Theorem 2 in [8] and utilizes Lemma 1

Lemma 3. *Consider Algorithm 3 with k partitions on a $G(n,p)$ graph with $p > \frac{2\log n}{n}$. Let x_i^t be the fractional load of the i^{th} partition at time t. Then a.s. the limit $X_i = \lim_{n,t\to\infty} x_i^t$ exists for each i. For exactly one i, $X_i = 1$.*

Proof. To extend the analysis of Lemma 2 from 2 partitions to k, we use induction and condition on each pair of bins. Of the k bins, select 2 and call them A and B. We modify Lemma 2's Equation 1 by substituting

$$\Pr\left[E^{(t)} = j\right] = \binom{t}{j} p^j (1-p)^j \text{ with } \Pr\left[E^{(t)} = j | A \text{ or } B \text{ is in the argmax}\right].$$

Given that our coupling to the Polya Urn process is unaffected, we just must show that

$$\Pr\left[E^{(t)} = 0, 1\right] > \Pr\left[E^{(t)} = 0, 1 | A \text{ or } B \text{ is in the argmax}\right].$$

The $E^{(t)} = 0$ case is simple since $\Pr\left[E^{(t)} = 0 | A \text{ or } B \text{ is the max}\right] = 0$. This is because we only use the argmax process when $E^{(t)} \geq 1$ (otherwise we would have assigned the vertex to the least loaded partition). When $E^{(t)} = 1$, this is equivalent to exactly 1 edge being placed and the probability that, of the k bins, it selects an endpoint in A or B is exactly $\frac{m_A^{(t)} + m_B^{(t)}}{t}$. Thus

$$\Pr\left[E^{(t)} = 1 | A \text{ or } B \text{ is the max}\right] = \frac{m_A^{(t)} + m_B^{(t)}}{t} \binom{t}{1} p(1-p)^{t-1} \leq \binom{t}{1} p(1-p)^{t-1} = \Pr\left[E^{(t)} = 1\right]$$

The result now follows from Theorem 2. □

Proof of Theorem 7: Combining Lemmas 2, 1, and 3, we conclude that Algorithm 3, with k partitions, asymptotically approaches a fractional load of 1 in one partition when run with $p > \frac{2 \log n}{n}$. □

Corollary 2. *Given a single $G(n,p)$ component, for any value $p > \frac{2 \log n}{n}$, Algorithm 1 eventually concentrates this component into 1 partition as $n \to \infty$.*

This analysis leaves open the question of how long the process must run before one partition dominates the others. This question has been studied by Drinea, Frieze and Mitzenmacher [12]. While they analyze the convergence rates for 2 bins, the proofs can be extended to k bins via the union bound. In the theorem B_0 is the name for one of the two bins and all-but-δ dominant means that B_0 contains at least a $1 - \delta$ fraction of the balls thrown. ϵ_0 is the initial amount that the two bins are separated by after n balls and is a constant depending on λ, say $\frac{1}{100\lambda}$.

Theorem 8 (Theorem 2.4 from [12])**.** *Assume that we throw balls into the system until B_0 is all-but-δ dominant for some $\delta > 0$. Then, if $\lambda > 1$, with probability $1 - e^{\Omega(n_0)}$, B_0 is all-but-δ dominant when the system has $2^{x+z} n_0$ balls, where $x = \log_{1+\frac{\lambda-1}{5+4(\lambda-1)}} \frac{0.4}{\epsilon_0}$ and $z = \log_{\frac{2\lambda}{\lambda+1}} \frac{0.1}{\delta}$.*

Lemma 4 extends this theorem to k bins.

Lemma 4 (Lemma 4.1 from [12])**.** *Suppose that when n balls are thrown into a pair of bins, the probability that neither is all-but-δ dominant is upper-bounded by $p(n, \delta)$. Here, we assume $p(n, \delta)$ is non-increasing in n. Then when $1 + kn/2$ balls are thrown into k bins, the probability that none is all-but-γ dominant is at most $\binom{k}{2} p(n, \delta)$ for $\gamma = \delta/(\delta + (1-\delta)/(k-1))$*

To summarize these results on the convergence rate, we find that the attachment process starts in earnest after $\frac{1}{p}$ vertices have arrived. After $\frac{2}{p}$ vertices have arrived, we claim the exponent in the process is greater than 1. From Lemma 4 the probability we do not get an all-but-ϵ domination is inversely polynomial in the number of partitions, $1/\epsilon$ and the number of vertices. The bound given by Theorem 8 holds for $\lambda = 2$ but is loose since λ value increases every after every round of $\frac{1}{p}$ vertices.

Comparisons. From these results, we conclude that the reason that Algorithm 2 fails to concentrate the component is the strict proportionality of its assignments. If instead it used any exponent greater than 1 on its scores, i.e. assign to i proportional to S_i^γ, the concentration result would hold. In particular, there is a huge spectrum of greedy algorithms of the style of arg max Greedy and Proportional Greedy. Amongst these, arg max Greedy provides the strongest possible preference towards concentration.

5.4 Extending to $G(\Psi, P)$ graphs and capacity constraints

Wiith no capacity constraints, the arg max Greedy approach is able to asymptotically place a single $G(n,p)$ component into one partition. Specifically, it initially places vertices in all partitions, but concentrates the component into one partition once it begins to see edges. In contrast, the Proportional Greedy approach always cuts the component into k pieces. We extend the analysis for arg max Greedy to graphs that consist of many good clusters, overcoming two challenges - capacity constraints and 'bad' inter-cluster edges.

The two challenges require restrictions to both Ψ and P. The capacity constraint ($\mathcal{C}$) can be violated if clusters are of size c and more than $\frac{\mathcal{C}}{c}$ communities chose a the same bin for their large component. From the analysis of throwing m balls into n bins, we know that the expected maximum load (w.h.p.) is $\frac{\log n}{\log\log n}$ when $m = n$ and $O(\frac{m}{n})$ when $m > n \log n$ [26]. If, for each cluster the location of its large component is chosen u.a.r. from the bins, then the balls-and-bins maximum load analysis allows us to argue that for small enough clusters, the slack required, $C = (1+\epsilon)\frac{n}{k}$, is also small. We use a small amount of slack to account for initial mistakes. These mistakes are the result of not seeing edges at the beginning of the process.

For simplicity, our proof first assumes that all clusters are of the same size and q, the prob. of inter-cluster edges, is 0. Now we can run l finite Polya Urn processes simultaneously and independently. Next, we show a non-zero bound on q that bounds the prob. of the process failing to find a cut on the inter-cluster edges is small. The assumption that the clusters are of equal size can be relaxed by adjusting the parameters in P.

Lemma 5. *Given a $G(\Psi, P)$ graph where $P_{i,j} = 0$, and $\forall i, |P_{i,i}| > 2\log n/|C_i|$, let $x_j^{(i)(t)}$ be the fraction of C_i that partition j holds at time t. With no capacity constraints, Theorem 7 guarantees that, as n grows, for each cluster i, if $\lim_{t\to\infty} x_j^{(i)(t)} = X_j^{(i)}$, then for some j, $X_j^{(i)} = 1$ while all others are 0.*

Proof. This follows directly from Theorem 7 and the fact that when $P_{i,j} = 0$, the individual components can not interact with one another. □

Next, we relax the constraint that there are no edges between components to obtain a bound that still does not necessarily respect capacity constraints.

Lemma 6. *Given a $G(\Psi, P)$ graph with $P_{i,i} = p$, $P_{i,j} = q$, all l clusters of equal size and $p > \frac{2\log n}{|C_i|}$. Let $x_j^{(i)(t)}$ be the fraction of C_i that partition j holds at time t. With no capacity constraints and k partitions, if $p > 3(k + \sqrt{k} + 1)lq$ then for each cluster i, if $\lim_{t\to\infty} x_j^{(i)(t)} = X_j^{(i)}$, then for some j, $X_j^{(i)} = 1$.*

Our goal is to bound the number of 'bad' inter-cluster edges away from the number of 'good' intra-cluster edges. We assume worst case distributions so these bounds can safely be relaxed in practice.

Consider component C_i. A natural condition is that there are more expected intra-cluster edges than inter-cluster so $p|C_i| > q(n - |C_i|)$. We require a few more properties. The first is that the inequality holds with reasonable probability so $p|C_i| - \sqrt{p|C_i|} > q(n - |C_i|) + \sqrt{q(n - |C_i|)}$. The second is that we maintain the separation at every step of the execution of the process so $p|C_i|\frac{t}{n} - \sqrt{p|C_i|\frac{t}{n}} > q(n-|C_i|)\frac{t}{n} + \sqrt{q(n-|C_i|)\frac{t}{n}}$. Finally, the total number of bad edges should be no more than the arg max of the good edges as this guarantees that the bad edges does not affect the concentration results for each component. This adds a factor of k to the bound so we must always guarantee there are at least k 'good' edges for each 'bad' edge.

Proof of Lemma 6: Let the edges from a vertex to its own component be 'good' and its external edges be 'bad'. The separation between the good and bad edges can be achieved through the use of Chernoff bounds. In particular, at time t, we expect that $|C_i|\frac{t}{n}$ vertices in C_i have already arrived. Using a Chernoff bound to justify using the expectation, we claim that with probability at least $1 - \delta$. Let the next vertex, v, be from C_i . Let $E_i^{(t)}$ be the total number of edges from v to the C_i vertices that have already arrived.

$$E_i^{(t)} > p|C_i|\frac{t}{n} - \sqrt{\log(1/\delta)p|C_i|\frac{t}{n}}.$$

The bad edges, B^t, are drawn from $B(q, (n-|C_i|)\frac{t}{n})$. For clarity, we approximate $n-|C_i|$ as $l|C_i|$. Again, with probability at least $1-\delta$, we claim that $B^t < ql|C_i|\frac{t}{n} + \sqrt{\log(1/\delta)ql|C_i|\frac{t}{n}}$.

We set $\delta = 1/e$ to obtain constant probability at least 1/2. This assumption is supported by the experimental results in the next Section. We include bounds that hold with high probability in the Appendix.

To add the constraint that the bad edges are less than the $\arg\max\{E_i^{(t)}(j)\}$, we note that the worst case is that all of the bad edges connect to one partition. This can happen if the rest of the graph may not be evenly distributed over the partitions, or we are observing a deviation in the distribution of bad edges. Given this it is sufficient that the number of bad edges is bounded away from the average number of good edges, so we use the condition that $p|C_i|\frac{t}{n} - \sqrt{p|C_i|\frac{t}{n}} > k[ql|C_i|\frac{t}{n} + \sqrt{ql|C_i|\frac{t}{n}}]$.

To extract meaningful restrictions on p and q from this equation, we note that $p|C_i|\frac{t}{n} - \sqrt{p|C_i|\frac{t}{n}} > k$ when $t > \frac{(k+\sqrt{k}+1)n}{p|C_i|}$. Similarly, $ql|C_i|\frac{t}{n} + \sqrt{ql|C_i|\frac{t}{n}} < 1$ when $t < \frac{(1/2(3-\sqrt{5})n}{ql|C_i|}$. We find that $\frac{(k+\sqrt{k}+1)n}{p|C_i|} < \frac{(1/2(3-\sqrt{5})n}{ql|C_i|}$ exactly when $p > (k + \sqrt{k} + 1)lq/(\frac{1}{2}(3 - \sqrt{5}))$. Simplifying, $p > 3(k + \sqrt{(k)} + 1)lq$ is sufficient. The gap between the left and right hand sides is monotonically increasing after this point, guaranteeing that all decisions will be made correctly with constant probability. □

Provided $k > 2$, $k + \sqrt{k} + 1 < 2k$ so this bound is more simply $p > 3 * 2klq = 6klq$. If we make stronger assumptions about the distribution of the vertices within the bins at any finite time, i.e. that they are approximately balanced, then we can drop the $(k + \sqrt{k} + 1)$ factor and obtain that $p > 3lq$ is sufficient.

The remaining technical point is the capacity constraints. Since no aspect of the algorithm load balances when edges exist, our only hope is that the components concentration points are distributed uniformly over the partitions. A standard balls-and-bins analysis then tells us how many components are assigned to each partition. With n balls and n bins, we expect the max load is $O(\log n)$ balls. However, with $n \log n$ balls and n bins, we expect the same max load - $O(\log n)$. With $m > n \log n$ balls, the max load approaches m/n.

This approach requires that the inter-component edges have no affect on the concentration location for each component. When $q = 0$ and there are no 'bad' edges, the location of the concentration of each component is uniform because of the random ordering of the stream. If $p = 1$, then the component grows exactly where the first vertex in the component is placed.

Other values of p and q require a more sophisticated argument. We exploit the gap between p and q to argue that many intra-component edges are seen before any inter-component edges. If the process has run long enough that we can use Lemma 4, then one partition contains more than half of the vertices that have arrived. Then, the $\arg\max$ is never changed by 'bad' edges and the processes do not affect each other.

Lemma 7. *Given a $G(\Psi, P)$ graph with $P_{i,i} = p$, $P_{i,j} = q$ satisfying both Lemma 6 and $q = O((k^{2.4}\log l)^{-1})$, with the number of clusters $l > k \log k$ and all clusters of equal size $|C_i|$, with high probability the maximum load of the partitions is bounded by $(1 + \epsilon)\frac{n}{k}$, where ϵ is a function of p, l and k.*

Proof. The locations of the concentration for each component are uniformly distributed. This can be done by noting that the partition that contains the maximum for each component is all-but-δ dominant. Applying Theorem 8 and Lemma 4 obtains that $q = O((k^{2.4}\log l)^{-1})$. The exact calculation is in the Appendix.

Given the components are uniformly distributed over the partitions, this is a 'balls-and-bins' process with l balls and k bins. If $l = ck \log k$ then, with high probability, the maximum load is $d_c \log k$ where d_c is a constant depending on c [26]. When $l >> k \log k$, with high probability, the maximum load is at most $\frac{l}{k} + \sqrt{2\frac{l}{k}\log k}$. From this we conclude that the clusters are nearly evenly distributed amongst the bins.

Finally, ϵ needs to be set so that the capacity constraints are not be violated by either of the two sources. The first is the distribution of vertices before any edges appear. This is in expectation $\frac{1}{p}$ vertices, and each partition holds $\frac{1}{pk}$ of them. The other source of slack required is the exact maximum load. This constant depends on l's relationship to k and can be obtained from [26]. □

6 Experimental Evaluation

The proofs in the previous sections show that for a certain range of parameters and size graph, the algorithm succeeds in recovering a good partitioning. It leaves open some interesting questions that can be experimentally evaluated:

- What is the relationship between ϵ, the load balancing factor in Lemma 7 and k, the number of partitions, and l, the number of components?

- How tight are the bounds? It is necessary that the density of edges within components is $p > \frac{2 \log n}{|C_i|}$ or that the gap between p and q, the probability of edges between components is at least $p > 6klq$?
- Are the convergence rates tight? For what size graph do we begin to recover the partitioning?
- When we are asymptotically recovering the partitioning, can we quantify how many mistakes we are making, i.e. how many vertices are separated from their components at the end of the process?

These questions are ideals candidate for experimental simulation. In fact, experimental results here can lead to a much better understanding of the algorithm than theoretical worst case bounds. In the following, using values that satisfy Lemma 6, we generate $G(\Psi, P)$ graphs and see how well arg max Greedy recovers the embedded cut.

Evaluation. Given a setting of the parameters, we generate a random $G(\Psi, P)$ graph and run the algorithm 25 times, each with a different random ordering. After each run, for each component in the $G(\Psi, P)$ graph, we its largest part in the partitioning i.e. if C_i is the component, and $P_1, P_2, \cdots P_k$ the final partitioning, we calculate $\max_{j \in k} |C_i \cap P_j|/|C_i|$. The theorems predicts that for all components, this value approaches 1 as the graph grows. Note that it can never be worse than $\frac{1}{k}$ for k partitions.

6.1 Load Balancing Factor

Understanding the load balancing factor required is the first step to understanding the other constraints. This is because if the load balancing factor is set too low, we will see this in the error calculations. To understand the slack required, we explore two settings of p and q, $p = 1$ and $q = 0$ or $q = \frac{p}{6kl}$ where l, the number of components is larger than $k \log k$. Now, for each size graph, we run the algorithm 20 times and record the number of partitions that hit their capacity constraints. We also vary l to understand how its relationship with k affects the required slack.

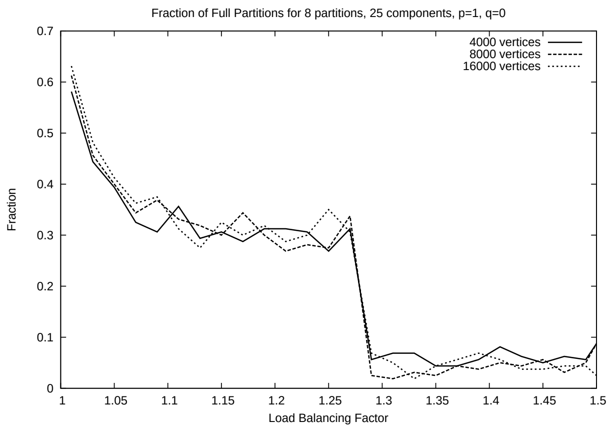

Figure 1: Load balancing is not a function of the size of the graph

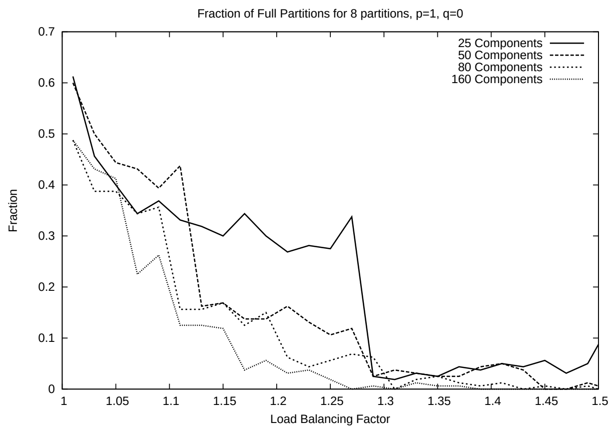

Figure 2: Increasing the number of components improves the load balancing.

We include 3 figures to demonstrate the relationship. The first, Figure 6.1 shows the fraction of full partitions when ϵ is allowed to range from 0.01 to 0.5 for graphs of size 4,000, 8,000 and 16,000. There is no difference between the threshold point in these graphs. The second, Figure 2 shows that fixing p, q and k but increasing l, the number of components, yields significantly better load balancing factors. The third Figure 3 shows that whether $q = 0$ or $q = 0.002 = p/6kl$, the load balancing appears the same.

6.2 Density Requirement

Lemma 6 requires that each component have edge density at least $p > \frac{2 \log n}{|C_i|}$. To explore whether this is necessary, we can fix values for q, k and l and let p range above and below $\frac{2 \log n}{|C_i|}$. For each run, we measure

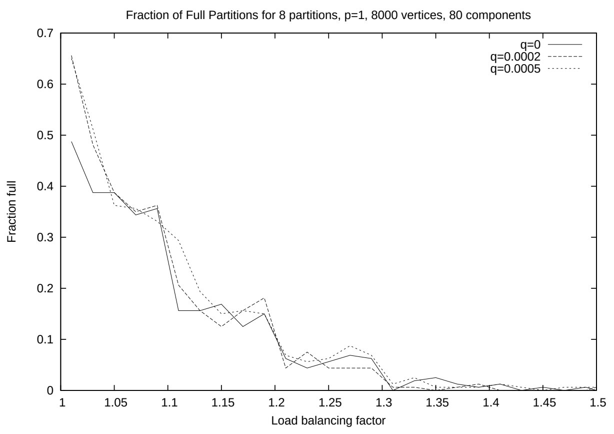

Figure 3: q does not play a large role in load balancing. Note that $q = 0.0005$ is above the threshold required by the theorems.

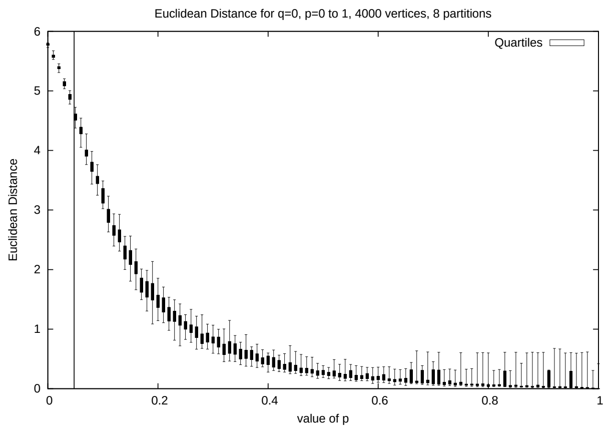

Figure 4: For fixed q, k, l values, as p increases, the error in the partitioning generated drops to 0. The vertical bar marks the value required by the theorems.

the error from the perfect solution by looking at the Euclidean distance between the length-l vector of the values of $\max_{j \in k} |C_i \cap P_j|/|C_i|$ and the all-ones vector.

Though not pictured in Figure 4, the graph size shifts the 'elbow' of the graph to the left with a sharper transition, matching the bound of the theorem.

6.3 Constraints on q

As in the experiments to understand the density factor, we can also fix values for p, k and l and let q range above and below $\frac{p}{6kl}$. Is the factor of k necessary? We measure the error by Euclidean distance as above.

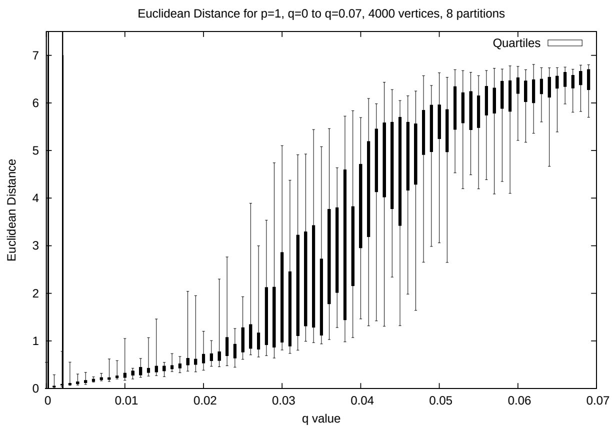

Figure 5: For fixed p, k, l values, as q increases, the error in the partitioning increases from 0 to maximum error. The leftmost bar at 0.00026 marks the theorems' requirement, while the second at 0.0021 is $q = p/6l$.

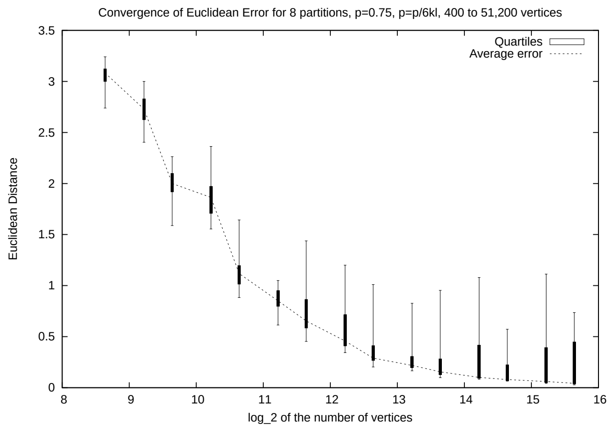

Figure 6: This graph shows that for fixed p, k, l, q values, as the size of the graph increases, the error in the partitioning generated drops to 0.

We clearly see the effect that increasing q has on the algorithm's ability to recover the partitioning in Figure 5. While the value required by the theorems seems unnecessarily small (and can only be seen by zooming in on this page), dropping the required factor of k and using $q = 0.02$ obtains an average error of only 0.07 over 25 runs when the maximum error is 7.

6.4 Convergence Rate

The values given by the Theorems in [12] about the rate of convergence imply a somewhat pessimistic bound - $q = O((k^{2.4} \log l)^{-1})$. We can evaluate this bound by fixing p, q, k and l and letting the size of the graph grow. As it grows, we can measure the Euclidean distance to find how quickly it is able to obtain good results in terms of recovering the partitioning.

The settings for the algorithm in Figure 6 were $p = 0.75, q = \frac{p}{6kl}, k = 8, l = 100$. The graph size range from 400 to 51,200 vertices. We see that as the size of the graph increases, the euclidean distance from the optimal partitioning solution quickly drops. For 51,200 vertices, the median error for 25 runs is only 0.04. This is despite the fact that the theorem required that $q < 0.000013$ whereas we used $q = 0.00015625$.

7 Conclusions and Future Work

We studied the problem of streaming balanced graph partitioning. We showed lower bounds on the approximation ratio obtainable by any algorithm and then analyzed two variants of a randomized greedy algorithm on a random graph model with embedded balanced k-cuts. On these graphs, we were able to explain previous experimental results showing that the arg max Greedy algorithm is able to recover a good partitioning while the Proportional Greedy variant is not. Our proof connects the greedy algorithms with finite Polya urn processes and exploits concentration results about those processes.

There are many future directions. One is improving the parameters of the analysis as the experiments show larger amounts of noise are fine. Another direction is exploring what other types of graphs the algorithm works and fail on. Finally, [30] considered additional stream orderings. Experimentally, the algorithms all performed better on BFS and DFS orderings than the random ordering. Developing techniques for analyzing streaming graph algorithms on BFS and DFS orders is an open problem.

Acknowledgments The author was supported by the NSF Graduate Fellowship and NSF grants CCF-0830797 and CCF-1118083. The author would also like to thank Anindya De, Gabriel Kliot, Miklos Racz, Satish Rao, Alexandre Stauffer and Ngoc Mai Tran for their helpful discussions.

References

[1] K. J. Ahn and S. Guha. Graph sparsification in the semi-streaming model. *ICALP*, 2009.

[2] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *FOCS*, 2006.

[3] R. Andersen and Y. Peres. Finding sparse cuts locally using evolving sets. In *STOC*, 2009.

[4] K. Andreev and H. Răcke. Balanced graph partitions. *Theory of Computing Systems*, 2006.

[5] S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM*, 2009.

[6] B. Bahmani, A. Chowdhury, and A. Goel. Fast incremental and personalized pagerank. *PVLDB*, 2010.

[7] S. Barnard. PMRSB: Parallel multilevel recursive spectral bisection. In *Supercomputing*, 1995.

[8] F. K. Chung, S. Handjani, and D. Jungreis. Generalizations of polya's urn problem. *Annals of Combinatorics*, 2003.

[9] A. Condon and R.M. Karp. Algorithms for graph partitioning on the planted partition model. In *RANDOM*, 1999.

[10] A. Das Sarma, S. Gollapudi, and R. Panigrahy. Sparse cut projections in graph streams. In *European Symposium on Algorithms (ESA)*, 2009.

[11] A. Das Sarma, S. Gollapudi, and R. Panigrahy. Estimating pagerank on graph streams. *Journal of the ACM*, 2011.

[12] E. Drinea, A. Frieze, and M. Mitzenmacher. Balls and bins models with feedback. In *SODA*, 2002.

[13] G. Even, J. Naor, S. Rao, and B. Schieber. Fast approximate graph partitioning algorithms. *SIAM J. Comput*, 28(6):2187–2214, 1999.

[14] J.E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, OSDI'12, pages 17–30, Berkeley, CA, USA, 2012. USENIX Association.

[15] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *SC*, 1995.

[16] G. Karypis and V. Kumar. Multilevel graph partitioning schemes. In *ICPP*, 1995.

[17] J. Kelner and A. Levin. Spectral sparsification in the semi-streaming setting. *STACS*, 2011.

[18] J. R. Lee, S. O. Gharan, and L. Trevisan. Multi-way spectral partitioning and higher-order cheeger inequalities. In *STOC*, 2012.

[19] A. Louis, P. Raghavendra, P. Tetali, and S. Vempala. Many sparse cuts via higher eigenvalues. In *STOC*, 2012.

[20] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. Hellerstein. GraphLab: A new framework for parallel machine learning. In *Uncertainty in AI (UAI)*, 2010.

[21] K. Makarychev, Y. Makarychev, and A. Vijayaraghavan. Approximation algorithms for semi-random partitioning. In *STOC*, 2012.

[22] G. Malewicz, M. Austern, A. Bik, J. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. Principles Of Distributed Computing (PODC), 2009.

[23] F. McSherry. Spectral partitioning of random graphs. In *FOCS*, pages 529–537, 2001.

[24] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Neural Information Processing Systems (NIPS)*, 2001.

[25] J Nishimura and J Ugander. Restreaming graph partitioning: Simple versatile algorithms for advanced balancing. Proc. 19th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD), 2013.

[26] M. Raab and A. Steger. "balls into bins" - a simple and tight analysis. In *RANDOM*, 1998.

[27] S. Salihoglu and J. Widom. Gps: A graph processing system. In *Scientific and Statistical Database Management*. Stanford InfoLab, July 2013.

[28] M. Sarwat, S. Elnikety, Y. He, and G. Kliot. Horton: Online query execution engine for large distributed graphs. In *ICDE*, 2012.

[29] Daniel A. Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning. 2008.

[30] I. Stanton and G. Kliot. Streaming graph partitioning for large distributed graphs. In *ACM KDD*, 2012.

[31] C. E. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic. Fennel: Streaming graph partitioning for massive scale graphs. 2012.

[32] L. Vaquero, F. Cuadrado, D. Logothetis, and C. Martella. xdgp: A dynamic graph processing system with adaptive partitioning. arXiv, 2013.

[33] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *NSDI*, 2012.

[34] M. Zelke. Intractability of min- and max-cut in streaming graphs. *IPL*, 111(3):145 – 150, 2011.

A Lower Bound Proofs

Theorem 9. *One-pass streaming balanced graph partitioning with an adversarial stream order can not be approximated within $o(n)$.*

Proof. Without loss of generality, we seek a balanced 2 partitioning. Consider a graph that is a cycle over n vertices with edges such that $(i, i+1) \mod n \in E$ for $1 \leq i \leq n$. Let the ordering be all odd nodes, then all even, i.e. $1, 3, 5 \ldots n-1, 2, 4, 6 \ldots n$. Assume that n is even. The optimal balanced partitioning cuts 2 edges. However, the given ordering reveals no edges until $\frac{n}{2}$ vertices arrive. Until the edges arrive, we have no way of distinguishing which vertices are 'near' each other. In particular, note that this ordering is indistinguishable from one where the odd vertices are given in a random order, or one where the odd nodes are interspersed with unconnected even nodes, i.e. $1, n-2, 3, n-4, 5, n-6 \ldots$. Thus, no algorithm can do better than cutting $\frac{n}{2}$ edges in expectation. This generalizes to k partitions. □

Theorem 10. *One-pass streaming balanced graph partitioning with a random stream order can not be approximated within $o(n)$.*

Proof. Again, we seek a balanced 2 partition for a cycle graph with a random ordering. Consider the t^{th} vertex to arrive in this ordering.

$$\Pr[t \text{ arrives with no edges }] = \Pr[\text{ both neighbors arrive after } t] = \tfrac{n-t}{n}\tfrac{n-t-1}{n-1}$$

so the number of vertices that we expect to arrive with no edges is

$$\mathbf{E}[\# \text{ with no edge}] = \textstyle\sum_{t=1}^{n} \frac{t}{n}\frac{t+1}{n-1} \approx \frac{1}{n^2}\sum_{t=1}^{n} t^2 - t = \frac{1}{n^2}(\frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} + \frac{n(n+1)}{2})$$

Therefore, asymptotically, we expect $\frac{n}{3}$ vertices to arrive with no edges. As before, when a vertex arrives with no edges, we are not able to determine which other vertices it is 'near'. For each of these, we expect to cut 1 edge, providing us with our lower bound. □

B High Probability Bounds for Lemma 6

The experiments justify the assumption that we only need the following two statements to hold with constant probability:

$$E_i^{(t)} > p|C_i|\frac{t}{n} - \sqrt{p|C_i|\frac{t}{n}}$$

$$B^t < ql|C_i|\frac{t}{n} + \sqrt{ql|C_i|\frac{t}{n}}$$

Requiring each to hold with probability $1 - \delta$ increases the gap required from $p > 3(k + \sqrt{k} + 1)kql$ by adding a dependency on δ. In particular, redoing the calculations, we have that

$$p|C_i|\frac{t}{n} - \sqrt{\log(1/\delta)p|C_i|\frac{t}{n}} > k$$

exactly when

$$t > \frac{n}{p|C_i|}(k + \log(1/\delta)/2 + \sqrt{k\log(1/\delta) + (\log(1/\delta)^2/4})$$

Similarly,

$$ql|C_i|\frac{t}{n} + \sqrt{\log(1/\delta)ql|C_i|\frac{t}{n}} < 1$$

exactly when

$$t < \frac{n}{ql|C_i|}(1 + \log(1/\delta)/2 - \sqrt{\log(1/\delta) + (\log(1/\delta)^2/4})$$

Solving these two equations as in Lemma 7 gives us a similar relationship that $p > f(\delta)kql$.

C Calculation of q for Lemma 7

In order to prove Lemma 7 we need to understand for a given setting of p and q how much interaction between the components there is at the t^{th} vertex. In particular, for the t^{th} vertex, we expect that there are be $p\frac{t}{l}$ edges from that vertex to its own component (good edges) and $q\frac{(l-1)t}{l}$ edges to other components (bad edges). Provided $t < \frac{l}{q(l-1)}$, we do not expect any bad edges so the components do not interact at all.

When we do begin to see bad edges, we can appeal to Lemma 4. If it is the case that for the given component, one partition contains a $1/2 + x$ fraction of the component that has arrived to this point, and all other partitions split the remaining $1/2 - x$ fraction then we can argue that the bad edges do not affect the concentration of the process provided the arg max for the good edges is not changed by the addition of the bad edges. Specifically, we are concerned with $t = \frac{l}{q(l-1)} > \frac{1}{q}$ so we can find x by solving:

$$(1/2+x)p\frac{t}{l} - \sqrt{(1/2+x)p\frac{t}{l}} > ((1/2-x)p\frac{t}{l} + \sqrt{(1/2-x)p\frac{t}{l}}$$

The above equation gives the distribution of the good edges at time t. Substituting that $t = \frac{1}{q}$ and there is only one bad edge, we need that

$$(1/2+x)\frac{p}{ql} - \sqrt{(1/2+x)\frac{p}{ql}} > ((1/2-x)\frac{p}{ql} + \sqrt{(1/2-x)\frac{p}{ql}}$$

This results in

$$x = \pm\sqrt{\frac{2(p/ql)^3 - (p/ql)^4}{4(p/ql)^4}}$$

From this, we can gather that a sufficient γ value required for Lemma 4 is $\gamma = \frac{1}{2} - \sqrt{1/2(p/ql)}$. Lemma 4 gives a formula for translating this γ into a δ value for Theorem 8. Solving for δ we get that

$$\delta = \frac{\gamma}{k-1-(k-2)\gamma}.$$

Plugging in our γ value, we obtain that

$$\delta = \frac{1/2 - \sqrt{1/2(p/ql)}}{k-1-(k-2)(1/2 - \sqrt{1/2(p/ql)})}.$$

We can simplify this by claiming that $\delta < \frac{1}{k}$ is sufficient.

The failure probability that we need to obtain from Theorem 8 for Lemma 4 is at most $\frac{c}{k^2 l}$ to use a union bound and still obtain a constant probability of success for the whole process. Therefore, we need to set $n_0' = n_0 + 2\log k + \log l$.

From here, we can obtain a number of balls thrown before we can obtain this level of concentration. In particular, we need $2^{x+z}n_0$ balls, where $x = \log_{1+\frac{\lambda-1}{5+4(\lambda-1)}} \frac{0.4}{\epsilon_0}$ and $z = \log_{\frac{2\lambda}{\lambda+1}} \frac{0.1}{\delta}$. The x term allows us to obtain up to all-but-0.1 dominance, while the second improves the result to all-but-δ dominance. Therefore, if $k \le 10$, then we only need $n_0 2^x$ balls. More generally, substituting that $\epsilon_0 = 1/5\lambda$ and $\delta = \frac{1}{k}$, this value becomes:

$$(2\lambda)^{1/\log_2(5\lambda/(1+4\lambda))}(0.1k)^{1/\log_2(2\lambda/(\lambda+1))}n_0'$$

The interesting thing about the process is that as more vertices arrive, the λ value increases. From this, we can immediately claim that this equation dramatically over-estimates the number of vertices needed before 2 bins would obtain a state with all-but-$\frac{1}{k}$ dominance. In particular, for the p and q values required by Lemma 6, we have $p = 6klq$ so λ reaches a value of $3k$ before we expect to see bad edges. Unfortunately, the best we can assume is that $\lambda = 2$ obtaining the following value:

$$4^6.578(0.1k)^2.4n_0' \approx 9127n_0'(0.1k)^{2.4}$$

It is certainly possible to set q to $1/9127n_0'(0.1k)^{2.4}$ but it is a significantly different bound from $p > 6klq$.