Min-max Hypergraph Partitioning[1]

Dan Alistarh, Jennifer Iglesias, Milan Vojnović

March 2015

Technical Report
MSR-TR-2015-15

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

[1]Dan Alistarh is with Microsoft Research, Cambridge, United Kingdom (daalista@microsoft.com). Jennifer Iglesias is with Carnegie Mellon University, Pittsburgh, PA (jiglesia@andrew.cmu.edu). Her work was conducted in part while an intern with Microsoft Research. Milan Vojnović is with Microsoft Research, Cambridge, United Kingdom (milanv@microsoft.com).

Abstract – In many applications, the structure of data can be represented by a hypergraph, where the data items are vertices, and the associations among items are represented by hyperedges. Equivalently, we are given as input a bipartite graph with two kinds of vertices: items, and associations (which we refer to as topics). We consider the problem of partitioning the set of items into a given number of partitions, such that the maximum number of topics covered by a partition is minimized.

This is a natural clustering problem, with various applications such as partitioning of information objects like documents, images, and videos, or allocation of system resources in data centers in the context of distributed streaming processing platforms or graph computation platforms.

In this paper, we introduce an approximation algorithm for the offline version of the problem, which is guaranteed to yield a good approximation for every input instance where a solution of small cost exists. Further, we consider the online version of the problem, in which items arrive online and each item must be assigned irrevocably to one of the partitions at the time of its arrival. We show that a simple greedy online assignment of items is able to recover a hidden co-clustering of vertices under a natural set of recovery conditions. We also report on extensive experimental results, covering both synthetically generated and real-world input bipartite graphs, which demonstrate that the greedy online assignment of items consistently yields superior performance when compared with alternative approaches.

1. INTRODUCTION

In a variety of applications, the structure of data can be represented by a hypergraph, where the data items are represented by vertices and associations among items are represented by hyperedges, i.e. subsets of items. This can be equivalently represented by a bipartite graph that has two kinds of vertices: the vertices representing the *items*, and the vertices representing the associations between items, which we refer to as *topics*. In this bipartite graph, each item is connected to one or more topics. The input can be seen as a graph with vertices belonging to (overlapping) communities.

There has been significant work on partitioning sets of items such that similar items are assigned to the same partition, see, e.g., reference [14] for a survey. This problem arises in the context of *clustering information objects* such as documents, images or videos. For example, in the case of documents, the goal would be to partition documents such that the maximum number of distinct *topics* covered by a partition is minimized, resulting in a parsimonious summary of each partition. Similar problems arise in the context of *data centers and cloud services*, where complex data processing requirements of applications need to be addressed to design a scalable system architecture. In the case of distributed stream processing platforms such as Amazon Lambda [1], or the Microsoft Bing stream processing platform [23], one would like to assign user queries to machines, such that queries which receive streams from similar sources are assigned to the same machine, minimizing machine load. Further, the same difficulties arise when replicating graph data among multiple machines. This problem arises in the context of processing enterprise emails [16], online social networks [25], and platforms for graph data analytics and machine learning, such as Giraph and GraphLab.[1]

Problem Definition. In this paper, we consider the problem of balanced graph partition with submodular graph cut costs. The input to the problem is a set of *items*, a set of *topics*, a set of *partitions*, and a *demand matrix* that specifies which particular subset of topics is associated with each individual item. Given an assignment of the items to partitions, the cost of the assignment is the maximum cost of a partition. The cost of a partition is defined as the number of distinct topics associated with the items assigned to the given partition. For instance, when partitioning a set of documents, the cost of the assignment can be interpreted as the maximum number of distinct topics associated to the items in a partition. In the context of load balancing of tasks in a data center, the cost of the assignment can be interpreted as the maximum processing cost of a machine, which is assumed to be proportional to the total number of distinct requirements of the set of tasks assigned to a partition.

In the online version of the problem, items arrive sequentially in time one at a time, and each item needs to be assigned, irrevocably, to one of the partitions at its arrival time. An online algorithm for assigning items to partitions must have a small computation and storage overhead in order to be practical for deployments at web scale.

The above problem is a natural generalization of a traditional load balancing problem, where each item is associated with a fixed weight (possibly depending on the machine it is assigned to), and the processing cost of a machine is the sum of weights of the items assigned to it. (We obtain this classic variant in our setting if we assume items with non-overlapping sets of topics.) Traditional load balancing is well studied with good approximation guarantees [3]. In contrast, the load balancing problem that we consider is more general, and far less understood. The difficulty stems from the fact that the processing cost of a machine is a more general *submodular function* of the set of items assigned to the machine.[2]

To illustrate the graph partition problem with submodular graph cut costs, let us consider the toy instance of the problem with the input as given in Figure 1. In this example, there are four items and three topics, where the sets of topics associated with individual items are specified by the edges of the given bipartite graph. Suppose we want to partition the set of items over two partitions. Assigning items 1 and 2 to partition 1 and items 3 and 4 to partition 2 results in the cut costs of respective values 2 and 3, thus the maximum cut cost of 3. However, one can do better. The optimum value of the cut cost is equal to 2, which can be achieved, for instance, by assigning items 1 and 3 to partition 1, and items 2 and 4 to partition 2.

Balanced graph partition with submodular cut costs is NP hard. The problem is even more challenging in its online version, as less information is available to the algorithm when decisions are made.

Contribution. We explore both offline and online algorithms for balanced graph partition with submodular cut costs, which we refer to as *min-max submodular multi-way graph cut problem.*

[1] http://giraph.apache.org and http://graphlab.org

[2] Adding a specific item to a machine that is already assigned a set of items A leads to an incremental cost less or equal than if the set of items assigned to this machine was $B \subseteq A$.

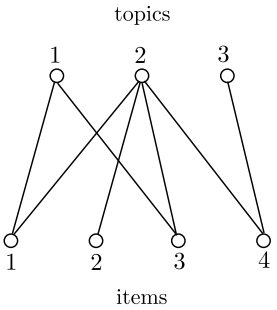


Figure 1: A simple illustrative example of a set of items with overlapping associations to topics.

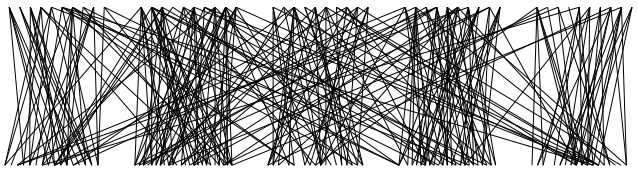


Figure 2: An example of a hidden co-clustering with five hidden clusters.

For the offline problem, we prove that it is possible to obtain non-trivial approximation guarantees by taking advantage of approximate solutions to a balanced hypergraph bisection problem. The balanced hypergraph bisection problem is known to have an $O(\sqrt{\log n})$ approximation [19], where n is the number of items, and there are well-developed off-the-shelf software packages that implement efficient heuristics for this problem, e.g. hMetis [17]. We obtain the approximation guarantee via a recursive partitioning scheme combined with a well-chosen weight function. The guarantee is useful for problem instances which have small optimum min-max submodular graph cut cost. Specifically, if the optimum solution f^* to the problem instance satisfies, for some positive constant c,

$$f^* \leq \frac{1}{1 - c/k} \cdot \frac{m}{k}$$

where m is the number of topics and k is the number of partitions, then our recursive algorithm has $O(\alpha)$ approximation ratio guarantee, where α is the approximation guarantee of the algorithm solving the balanced hypergraph bisection problem. It can be easily shown that m/k is a lower bound for the optimum value of any instance of the min-max submodular graph cut problem, therefore, the above condition on f^* can be interpreted as saying that the problem instance allows an efficient solution. The key ingredient of our analysis is a careful formulation of a specific instance of a balanced hypergraph bisection problem which is approximately solved in a subroutine of our algorithm.

For the online problem, our analysis considers a random bipartite graph input, as for the worst-case inputs the problem is intuitively infeasible. In this input model, each item is associated with a set of topics that is drawn independently from a probability distribution over the subsets of the set of topics. This is a mixture distribution, where the components of the mixture represent the underlying structure of a hidden clustering defined as follows.

The set of topics is assumed to be partitioned into a given number $\ell \geq 1$ of hidden clusters. The distribution over the set of items is defined as a mixture of distributions $p_1, p_2, \ldots, p_l$, each defined over the subsets of topics, where the distribution p_i has a higher mass for a topic from cluster i than for a topic from cluster $j \neq i$. Specifically, p_i is assumed to be such that a topic from cluster i is selected with probability p, and a topic from a cluster $j \neq i$ is selected with probability $q \leq p$. This model is similar in spirit to the popular stochastic block graph model. The given construction also corresponds to a *hidden co-clustering* [10, 11] of the input bipartite graph (see Figure 2 for an example). We consider asymptotically accurate recovery of this hidden co-clustering: a hidden cluster is said to be *asymptotically recovered* if the portion of items from the given hidden cluster assigned to the same partition goes to one asymptotically as the number of items observed grows large.

Our main analytic result is a set of sufficient conditions under which a simple greedy algorithm asymptotically recovers the hidden clusters (Theorem 2). We prove that a sufficient condition for the recovery of hidden clusters is that the number of clusters ℓ is at least $k \log k$, and that the gap between the probability parameters q and p is sufficiently large:

$$q < \frac{\log r}{kr} < \frac{2 \log r}{r} \leq p$$

where r is the number of topics in a hidden cluster. Roughly speaking, this means that if the mean number of topics to which an item is associated with from its corresponding cluster is at least twice as large as the mean number of topics to which an item is associated with from other clusters, then the simple greedy online algorithm guarantees asymptotic recovery of hidden clusters.

The proof is based on a technical coupling argument, where we first show that assigning an item based on the number of topics it has in common with each partition is similar to making the assignment proportionally to the number of *items* corresponding to the same hidden cluster present on each partition. In turn, this allows us to couple the assignment of every hidden cluster with a *Polya urn process* [9] with "rich-get-richer" dynamics, which implies that the policy converges to assigning each item from a hidden cluster to the same partition. This recovery property then implies that this strategy will ensure a constant factor approximation of the optimum assignment.

Further, we provide experimental evidence showing that the practical performance of this greedy online algorithm matches its good predicted behavior, on real-world bipartite graph datasets, outperforming more complex assignment strategies, and even offline approaches.

Roadmap. Section 2 introduces the problem and presents basic results. Section 3 presents an offline approximation, while Section 4 presents our main theoretical results. Experimental results are presented in Section 5, and related work is discussed in Section 6. Technical proofs are deferred to the online companion technical report [**?**].

2. SYSTEM MODEL AND DEFINITIONS

In this section we provide a formal problem formulation, and present some basic results on the computational hardness and lower bounds.

2.1 Problem Formulation

Input. The input includes:

- A set of *items* $N = \{1, 2, \ldots, n\}$,
- A set of *topics* $M = \{1, 2, \ldots, m\}$,
- A set of *partitions* $K = \{1, 2, \ldots, k\}$,
- A *demand matrix* $D = (d_{i,l}) \in \{0,1\}^{n \times m}$ where $d_{i,l} = 1$ indicates that item needs topic l, and $d_{i,l} = 0$, otherwise.[3]

There are multiple different ways to represent a set of items and their corresponding topics. One such representation is by a *bipartite graph* $G = (N, M, E)$ where there is an edge $(i, l) \in E$ if and only if item i needs topic l. An alternative representation is by a *hypergraph* $H = (N, E)$ where a hyperedge $e \in E$ consists of all items that use the same topic.

The Optimization Problem. An assignment of items to partitions is given by $x \in \{0,1\}^{n \times k}$ where $x_{i,j} = 1$ if item i is assigned to partition j, and $x_{i,j} = 0$, otherwise. Given an assignment of items to partitions x, the cost of partition j is defined to be equal to the minimum number of distinct topics that are needed by this partition to cover all the items assigned to it, i.e.

$$c_j(x) = \sum_{l \in M} \min \left\{ \sum_{i \in N} d_{i,l} x_{i,j}, 1 \right\}.$$

As defined, the cost of a partition is a *submodular function* of the items assigned to it. We consider the *min-max submodular graph cut* problem defined as follows:

$$\begin{array}{ll} \text{minimize} & \max\{c_1(x), c_2(x), \ldots, c_k(x)\} \\ \text{subject to} & x \in \{0,1\}^{n \times k} \end{array} \quad (1)$$

hypergraph Partition Formulation. Given an input set of items and topics specified by a hypergraph $H = (N, E)$, the problem can be phrased as that of a graph partition problem where the goal is to partition the set of vertices N into a given number of $k \geq 2$ parts $N_1, N_2, \ldots, N_k$ so as to minimize the maximum load on a part. The load on a part is the number of hyperedges adjacent to that part. A hyperedge is considered to be *adjacent* to a part if at least one of its vertices is assigned to that part.

2.2 Basic Results

NP Completeness. We note that our problem is NP-Complete. The proof follows by reduction from the *subset sum* problem.

LEMMA 1. *The min-max submodular graph cut problem is NP-Complete.*

Lower Bound. The following lemma provides a lower bound on the optimal value of the problem for a given input, using the observation that each topic needs to be made available on at least one partition.

[3]The framework allows for a natural generalization to allow for real-valued demands. In this paper we focus on $\{0,1\}$-valued demands.

LEMMA 2. *For any assignment of items to partitions, the maximum cost of a partition is at least m/k.*

Uniform Random Item Assignment. We now analyze the performance of an algorithm which simply assigns items to randomly chosen partitions upon arrival. Although this is a popular strategy among practitioners, the following result suggests that random assignment is not a good solution.

LEMMA 3. *The expected maximum load of a partition under random assignment is of at least*

$$\left(1 - \frac{1}{m} \sum_{j=1}^{m} \left(1 - \frac{1}{k}\right)^{n_j}\right) \cdot m,$$

where n_j is the number of items which have j as a topic.

The proof follows by simply expressing the probability that a topic j is present on a fixed partition i. As an example, if we assume that $n_j \geq k$ for each topic j, we obtain that the expected maximum load is of at least $(1 - 1/e)m$. This suggests that the performance of random assignment is rather poor, as the maximum load does not decrease as we increase the number of partitions k.

2.3 Hidden Co-Clustering

We consider a set of topics $\mathcal{R}$, partitioned into ℓ clusters $C_1, C_2, \ldots, C_\ell$, each of which contains r topics. Based on these (hidden) clusters, an incoming item τ is associated with (subscribes) to topics as follows. The item τ is first assigned to a "home" cluster C_h, chosen uniformly at random. The item then subscribes to topics inside its cluster, picking each topic independently with some probability p. Next, the item subscribes to topics from a fixed arbitrary "noise" set Q of size $\leq r/2$ outside its home cluster, where each topic in Q is subscribed to independently, with some probability q.[4]

We say that a bipartite graph with n nodes is in $\mathsf{HC}(r, \ell, p, q)$ if it is constructed using the above process, with n items and ℓ clusters with r topics per cluster, where each item subscribes to topics inside its randomly chosen home cluster with probability p, and to topics from the noise set with probability q.

Allocating Items. We iterate the generative process over time, where at each time step t we generate a new item, and consider a set of k partitions $S_1, S_2, \ldots, S_k$. At each time step, the incoming item is immediately assigned to one of the k partitions, together with all its topics, according to some policy. Policies do not know the number of hidden clusters or their size, but can examine previous assignments.

Asymptotic Recovery. We say that an algorithm *asymptotically recovers* the hidden co-clustering $C_1, C_2, \ldots, C_\ell$ if there exists a time t_R and a partition S_i for each cluster C_i such that, after time t_R, each item corresponding to the hidden cluster C_i is assigned to server S_i, with probability $1 - \phi$, where $\phi < 1$ is some error parameter which we will choose to asymptotically go to 0.

***Balanced* Recovery.** The recovery condition above can also be ensured by a trivial algorithm, which places all items

[4]Sampling outside topics from a large set, such as the set of all possible topics, would lead any partition housing many items from a cluster to contain practically *all* possible topics, which renders the problem trivial.

Data: hypergraph $H = (V, E)$, integer k
Result: A partition of V into k parts
Compute the weight function w_H
Run the balanced hypergraph bisection algorithm on H with cost function w_H
Let V_1, V_2 be the bi-partition found
if $k \neq 1$ **then**
    Run the algorithm again with $(H(V_1), k/2)$ and $(H(V_2), k/2)$
    Return the partitions from the results of these two runs
else Return V_1, V_2

Algorithm 1: Offline approximation using recursive solving of balanced hypergraph bisection.

Data: k; $H = (V, E)$, receives H one vertex at a time; c the capacity
Result: A partition of V into k parts
Set $N_1, N_2, \ldots, N_k$ to be the empty sets
while *there are unassigned items* **do**
    Receive the next item t, and its topics R
    Let $I = \{i : |N_i| \leq \min_j |N_j| + c\}$
    Compute $r_i = |N_i \cap R|$ for all $i \in I$
    Place t and its topics R on the partition i with largest r_i, update N_i
    Break ties by choosing the least loaded partition
return $N_1, N_2, \ldots, N_k$

Algorithm 2: The greedy algorithm.

on the same partition. We avoid such solutions by requiring that the ratio between the maximum and average partition load after time t_R is upper bounded by a constant $B > 0$.

3. OFFLINE APPROXIMATION

Since the problem as defined in (1) is NP-hard, as showed in Lemma 1, it is natural to ask for an approximation algorithm. By assigning all the items to the same partition, we obtain a k approximation: this is because assigning all items to the same partition incurs the maximum cost of a partition of value m, and by Lemma 2 the optimum maximum cost of a partition is at least m/k. Therefore, any approximation algorithm of interest must guarantee an approximation ratio smaller than k.

We now provide an approximation algorithm for instances whose optimal solution is close to the lower bound of m/k. The algorithm is based on recursively finding approximate solutions to a balanced hypergraph bisection problem which we define in this section, based on a careful choice of the weights assigned to vertices used as input to the balanced hypergraph bisection problem.

3.1 Algorithm

The *balanced hypergraph bisection* problem takes as input a hypergraph $H = (V, E)$, a cost function of the vertices, $w : V \to \mathbf{R}_+$, and a slackness parameter $\nu \geq 0$. The goal is to find a bi-partition of the vertices such that at most $\nu \sum_{v \in V} w(v)$ of the weight is in each part, and the number of crossing edges is minimized. An edge is crossing if it has vertices in both parts of the bi-partition.

We shall use a specific definition of an induced subgraph: the *induced subgraph* $H(U)$ of $H = (V, E)$ has vertex set U and edge set $F = \{e \cap U | e \in E, e \cap U \neq \emptyset\}$.

We define a weight function for the vertices of a hypergraph $H = (V, E)$ as follows:

$$w_H(v) = \sum_{e \in E, v \in e} \frac{1}{|e|}, \text{ for } v \in V$$

where $|e|$ is the number of vertices in e. These weights may be interpreted as an approximation of the load of a set of vertices when there are few crossing edges. The pseudo-code is given in Algorithm 1.

3.2 Approximation Guarantee

We now provide an approximation guarantee for the min-max submodular graph cut problem.

THEOREM 1. *Let f^* be the optimal cost for the min-max submodular graph cut problem. For any integer $k \geq 2$, given an α-approximation for the balanced hypergraph bisection problem, we get an $4[1 + \alpha(k - m/f^*)]$ approximation algorithm for min-max submodular graph cut.*

An immediate corollary of this theorem is a sufficient condition for a constant-factor approximation to our problem.

COROLLARY 1. *Suppose $f^* \leq m/(k - c)$, for some c not depending on m, n, k. Then, an α-approximation for balanced hypergraph bisection gives a $O(\alpha)$-approximation for min-max submodular graph cut.*

Balanced hypergraph Bisection. Recent breakthrough results by Anand [19, Theorem 6.1.8] gave an $O(\sqrt{\log n})$ approximation algorithm for the balanced hypergraph partition problem. These results are based on semi-definite programming and a complex series of reductions, and are currently not practical. However, the hMetis package [17] offers heuristic answers to balanced hypergraph bisection.

4. ONLINE APPROXIMATION

In this section, we show that the greedy strategy is expected to perform well on a *hidden co-cluster* input.

4.1 The Greedy Online Algorithm

We considered the online version of the problem, where we receive one item at a time together with all its corresponding topics. The item must be immediately and irrevocably assigned to some partition.

In the following, we focus on a simple greedy strategy, described in Algorithm 2. We show that this strategy has a useful cluster recovery property, and that this theoretical property is doubled by good practical performance.

The choice of this greedy strategy may appear unintuitive: its main goal does not appear to be balancing, but rather to cluster similar items. This could in theory lead to large imbalances, and in particular to one partition getting all the topics. To prevent degenerate cases, we add a *balancing constraint* specifying the maximum load imbalance. If adding the item to the candidate partition resulting from greedy would violate the balancing constraint, then the item is assigned to the first partition which can accommodate it without violating the constraint, in decreasing order of intersection size.

4.2 The Recovery Theorem

Our main technical result provides sufficient conditions on the cluster parameters for the greedy strategy to provide *balanced recovery of hidden clusters, with high probability.*

THEOREM 2. *We are given a random input consisting of a hidden co-cluster graph G in $\mathsf{HC}(r, \ell, p, q)$ to be distributed across $k \geq 2$ partitions. If the number of clusters is $\ell \geq k \log k$, $p \geq 2 \log r/r$, and $q < \log r/(rk)$, then the greedy strategy ensures balanced asymptotic recovery of the clusters with high probability in r.*

Coupling and High Probability. In the following, we say that two random processes are *coupled* to mean that their random choices are the same. We say that an event occurs *with high probability (w.h.p.)* if it occurs with probability at least $1 - 1/r^c$, where $c \geq 1$ is a constant.

Proof Overview. The proof of this result is quite involved, and can be summarized as follows. The first step will be to prove that greedy recovers a single cluster w.h.p. when assigning to just two partitions. More precisely, given a sequence of items generated from a single home cluster, and two partitions, a version of the algorithm without balancing constraints will eventually converge to assigning all incoming items to a single partition. This is a main technical step of the proof, and it is based on a coupling of greedy assignment with a "rich get richer" *Polya urn process* [9], and then using the convergence properties of such processes. Further, we extend this coupling claim from two partitions to $k > 2$ partitions, again for a single cluster, showing that, when the input consists of items from a single cluster, greedy will quickly converge to assigning all items to a single partition, w.h.p.

In the next step, we prove that the algorithm will in fact recover ℓ clusters of items in parallel, assigning each of them (i.e., most of their corresponding items) independently at random to one of the partitions, and that this convergence is not adversely affected by the fact that items also subscribe to topics from outside their home cluster. The problem of determining the maximum partition load is then reduced to showing that the maximum number of clusters that may be randomly assigned to a partition is *balanced*, as well as bounding the extra load due on a server to topics outside the home cluster and miss-assignments.

Due to space limitations, we defer complete proofs to the full version of our paper.

Polya Urn Processes. For reference, a Polya urn process [9] works as follows. We start each of $k \geq 2$ urns with one ball, and, at each step t, observe a new ball. We assign the new ball to urn $i \in \{1, \ldots, k\}$ with probability proportional to $(b_i)^\gamma$, where $\gamma > 0$ is a fixed real constant, and b_i is the number of balls in urn i at time t. We shall employ the following classic result.

LEMMA 4 (POLYA URN CONVERGENCE [9]). *Consider a finite k-bin Polya urn process with exponent $\gamma > 1$, and let x_i^t be the fraction of balls in urn i at time t. Then, almost surely, the limit $X_i = \lim_{t \to \infty} x_i^t$ exists for each $1 \leq i \leq k$. Moreover, we have that there exists an urn j such that $X_j = 1$, and that $X_i = 0$, for all $i \neq j$.*

4.2.1 Step 1: Recovering a Single Cluster

Strategy. We first prove that, in the case of a single home cluster for all items, and two partitions ($k = 2$), with no balance constraints, the greedy algorithm with no balance constraints converges to a *monopoly*, i.e., eventually assigns all the items from the cluster onto the same partition, w.h.p. Formally, there exists some convergence time t_R and some partition S_i such that, after time t_R, all future items will be assigned to partition S_i, with probability at least $1 - 1/r^c$.

Our strategy will be to couple greedy assignment with a Polya urn process with exponent $\gamma > 1$, showing that the dynamics of the two processes are the same, w.h.p. There is one serious technical issue: while the Polya process assigns new *balls* based on the *ball* counts of urns, greedy assigns *items* (and their respective topics) based on the number of *topic intersections* between the item and the partition. It is not clear how these two measures are related.

We circumvent this issue by taking a two-tiered approach. Roughly, we first prove that, w.h.p., we can couple the number of items on a server with the number of *unique topics* assigned to the same partition. We then prove that this is enough to couple the greedy assignment with a Polya urn process with exponent $\gamma > 1$ (Lemma 12). This will imply that greedy converges to a monopoly, by Lemma 4.

Notation. Fix a time t in the execution of the greedy assignment process, corresponding to some new item being randomly generated. A topic r is *known* at time t if it has been a topic for some item up to time t. A known topic r is a *singleton* if it has been placed on one partition, but not on others. Otherwise, it is a *duplicate*. In the following, we will focus on the above quantities around the special time $t_0 = r/\log r$, which we shall prove is close to the convergence time. For simplicity, when referring to a value at time t_0, we omit the subscript.

Convergence to a Monopoly. We first prove that one of two things must happen during the algorithm's execution: either one of the partitions gains a constant size advantage, or the algorithm can be coupled with a Polya urn process. In both cases, the algorithm will converge to a monopoly.

LEMMA 5. *Given a hidden cluster input $\mathsf{HC}(r, \ell, p, q)$, with $\ell = 1$, $p \geq 2 \log r/r$ and $q = 0$, for every $t \geq t_0 = r/\log r$, to be allocated onto two partitions, one of the following holds:*

1. *With high probability, the greedy algorithm with a cluster and two partitions can be coupled with a finite Polya urn process with parameter $\gamma > 1$, or*
2. *There exists a constant $\rho > 0$ such that the ratio between the number of singleton topics on the two partitions is $> 1 + \rho$ at time t_0.*

Further, the algorithm converges to assigning all incoming items to a single partition after some time $t \geq r/\log r$, w.h.p.

4.2.2 Step 2: k Partitions and Convergence

Multiple Partitions. Consider now greedy on $k \geq 3$ partitions, but with no load balancing constraint. We now extend the previous argument to this case.

Let $t \geq r/\log r$, and consider the state of the partitions at time t. If there exists a set of partitions which have a constant fraction more singleton topics than the others, it follows by a simple extension of Lemma 12 (considering sets of partitions as a single partition) that these heavier partitions will attract all future items and their topics, w.h.p. The only interesting case is when the relative loads of all partitions are close to each other, say within an ϵ fraction. However, in this case, we can apply Lemma 12 to *pairs* of partitions, to obtain that some partition will gain an monopoly.

LEMMA 6. *Given a single cluster instance in* $\mathsf{HC}(r, \ell, p, q)$ *with* $p \geq 2\log r/r$ *and* $q = 0$ *to be split across* k *partitions, the greedy algorithm with no balancing constraints will recover the cluster onto a single partition w.h.p.*

Speed of Convergence. Note that, by Chernoff bounds, once one of the partitions acquires a constant fraction more topics from the single cluster than the other partitions, it will acquire all future topics w.h.p. By Lemma 12, it either holds that one of the partitions dominates before time $t_0 = r/\log r$, or that we can couple greedy with a Polya urn process with $\gamma > 1$ after this time. The only remaining piece of the puzzle, before we consider the multi-cluster case, is *how fast* the Polya urn process converges to a configuration where some partition contains a constant fraction more topics than the others.

Drinea et al. [12] studied this question. We combine Theorems 2.1 and 2.4 from their paper to bound the convergence time of the algorithm as follows.

THEOREM 3. *Given a hidden co-cluster graph in* $\mathsf{HC}(r, \ell, p, q)$, *with parameters* $p \geq 2\log r/r$, $q = 0$, *and a single hidden cluster, i.e.,* $\ell = 1$, *to be split across* k *partitions, the following holds. There exists a partition* j *such that, after* $2r/\log r$ *items have been observed, each additional generated item is assigned to partition* j, *w.h.p.*

4.2.3 Final Step: The General Case

We now complete the proof of Theorem 2 in the general case with $\ell \geq 2$ clusters and $q > 0$. We proceed in three steps. We first show the recovery claim for general $\ell \geq 2$, but $q = 0$ and no balance constraints, then extend it for any $q \leq \log r/(rk)$, and finally show that the balance constraints are practically never violated for this type of input.

Generalizing to $\ell \geq 2$. A first observation is that, even if $\ell \geq 2$, the topics must be disjoint across clusters if $q = 0$. Also, since we assume no balance constraints, the clusters and their respective topics are independent. The assignment problem for clusters then reduces to throwing ℓ balls (the *clusters*) into k bins (the *partitions*). We use concentration bounds on the result bin loads to understand the maximum number of clusters per partition, which in turn bounds the maximum load.

LEMMA 7. *Assume a clustered bipartite graph* G *with parameters* $\ell \geq k\log k$, $p \geq 2\log r/r$, *and* $q = 0$, *to be split onto* k *partitions with no balance constraints. Then, w.h.p., greedy ensures balanced recovery of* G. *Moreover, the maximum number of topics per partition is upper bounded by* $(1+\beta)r\ell/k$, *w.h.p., where* $\beta < 1$ *is a constant.*

Generalizing to $q > 0$. The next step is to show that, as long as $q < \log r/(rk)$, the greedy process is not adversely affected by the existence of out-of-cluster topics, since out-of-cluster topics have a very low probability of changing the algorithm's assignment decisions.

LEMMA 8. *Given* $q < \log r/(rk)$, *then w.h.p. the greedy process can be coupled with a greedy process on the same input with* $q = 0$, *where* $r/\log r$ *topics have been observed for each cluster of topics.*

We can combine Lemma 16 and Theorem 3 to obtain that greedy converges after $2r/\log r$ items have been observed out of each hidden cluster.

The Capacity Constraint. Finally, we extend the argument to show that the partition capacity constraints do not cause the algorithm to change its decisions, with high probability. The proof follows by noticing that the load distributions are balanced across servers as the algorithm progresses, as items are either distributed randomly (before convergence), or to specific partitions chosen uniformly at random (after convergence).

LEMMA 9. *On a hidden co-cluster input, greedy without capacity constraints can be coupled with a version of the algorithm with a constant capacity constraint, w.h.p.*

Final Argument. Putting together Lemmas 15, 16 and 17, we obtain that greedy ensures balanced recovery for general hidden cluster inputs in $\mathsf{HC}(r, \ell, p, q)$, for parameter values $\ell \geq k\log k$, $p \geq 2\log r/r$, and $q \leq \log r/(rk)$. This completes the proof of Theorem 2.

Moreover, the fact that each cluster is recovered can be used to bound the maximum load of a partition. More precisely, by careful accounting of the cost incurred, we obtain that the maximum load is $2.4r\ell/k$, with high probability, where the extra cost comes from initial random assignments, and from the imperfect balancing of clusters between partitions.

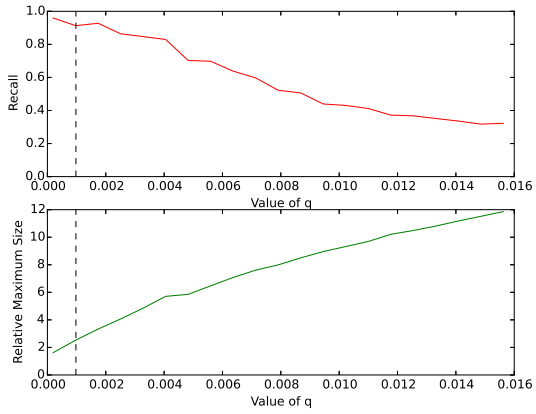
5. EXPERIMENTAL RESULTS

In this section we present experimental performance evaluation results for the online assignment of items to partitions. We conduct two types of analysis. The first is for inputs according to the hidden co-cluster input. The goal is to illustrate the sufficiency of the recovery condition in Section 4 and to evaluate robustness of the algorithm to removing some of the assumptions. The second part considers several real-world instances of bipartite graph inputs covering a wide range of scenarios and scale of data. In brief, the first part of the analysis demonstrates sufficiency of the recovery, while the second part demonstrates good performance of the greedy online assignment considered in this paper in comparison to several other natural alternative assignment strategies.

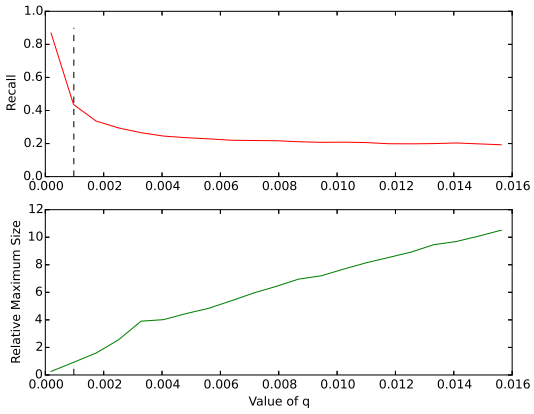
5.1 Hidden Co-Cluster Graphs

Setup. In this section, we validate our theoretical results on recovery through experiments. In particular, we generated hidden co-cluster graphs for various values of parameters r, ℓ, p, q, and $m = r\ell$. We focus on two measures. The first is *recall*, which is defined as follows: for each cluster, we consider the partition that gets the *highest fraction* of topics from this cluster. We then average these fractions for all clusters, to obtain the recall. The second measure is the *maximum partition size*, i.e., the maximum number of topics on a partition after $m\log m$ items have been observed, normalized by m/k, which is a lower bound on the optimum. We expect these two measures to be correlated, however neither one in isolation would be sufficient to ensure that greedy provides balanced recovery.

When generating the random inputs, we select a random home cluster, then subscribe to topics from the home cluster with probability p. When subscribing to topics from outside the home cluster, we pick *every* topic from outside the cluster independently with probability q (so the noise set Q contains all topics).



(a) Testing the sufficient condition



(b) Experiments for non-uniform sources.

Figure 3: Experiments for hidden cluster bipartite graphs. The dotted line upper bounds the analytic recovery threshold.

Testing the Sufficient Conditions. Our first experiment, presented in Figure 3a, fixes the value of p to $2 \log r/r$, and increases the value of q from $p/(10\ell)$ (below the analytic recovery threshold) to $8p/\ell$ (above the recovery threshold). The dotted line represents an upper bound on the recovery threshold $q = p/(4\ell)$. The experiment shown is for $r = 64, \ell = 64$, and $k = 20$. The results are stable for variations of these parameters.

The experiments validate the analysis, as, below the chosen threshold, we obtain both recall over 90%, and partition size within two of optimal. We note that the threshold value we chose is actually *higher* than the value $q = \log r/(rk)$ required for the analysis.

Non-Uniform Clusters. We repeated the experiment choosing home clusters with *non-uniform probability*. In particular, we select a small set of clusters which have significantly more probability weight than the others. The experimental results are practically identical to the ones in Figure 3a, and omitted due to space limitations. These empirical results suggest that non-uniform cluster probabilities do not affect the algorithm's behavior.

Non-Uniform Topics. Finally, in Figure 3b, we analyze the algorithm's behavior if topics have non-uniform probability weights. More precisely, we pick a small set of topics in each cluster which have disproportionately high weight. (In the experiment shown, four sources out of 64 have .1 probability of being chosen.) We observe that this affects the performance of the algorithm, as recall drops at a higher rate with increasing q.

The intuitive reason for this behavior is that the initial miss-classifications, before the algorithm converges, have a high impact on recall: topics with high probability weight will be duplicated on all partitions, and therefore their are no longer useful when making assignment decisions.

5.2 Real-World Bipartite Graphs

Datasets and Evaluation. We consider a set of real-world bipartite graph instances with a summary provided in Table 4. All these datasets are available online, except for Zune podcast subscriptions. We chose the consumer to be the item and the resource to be the topic.

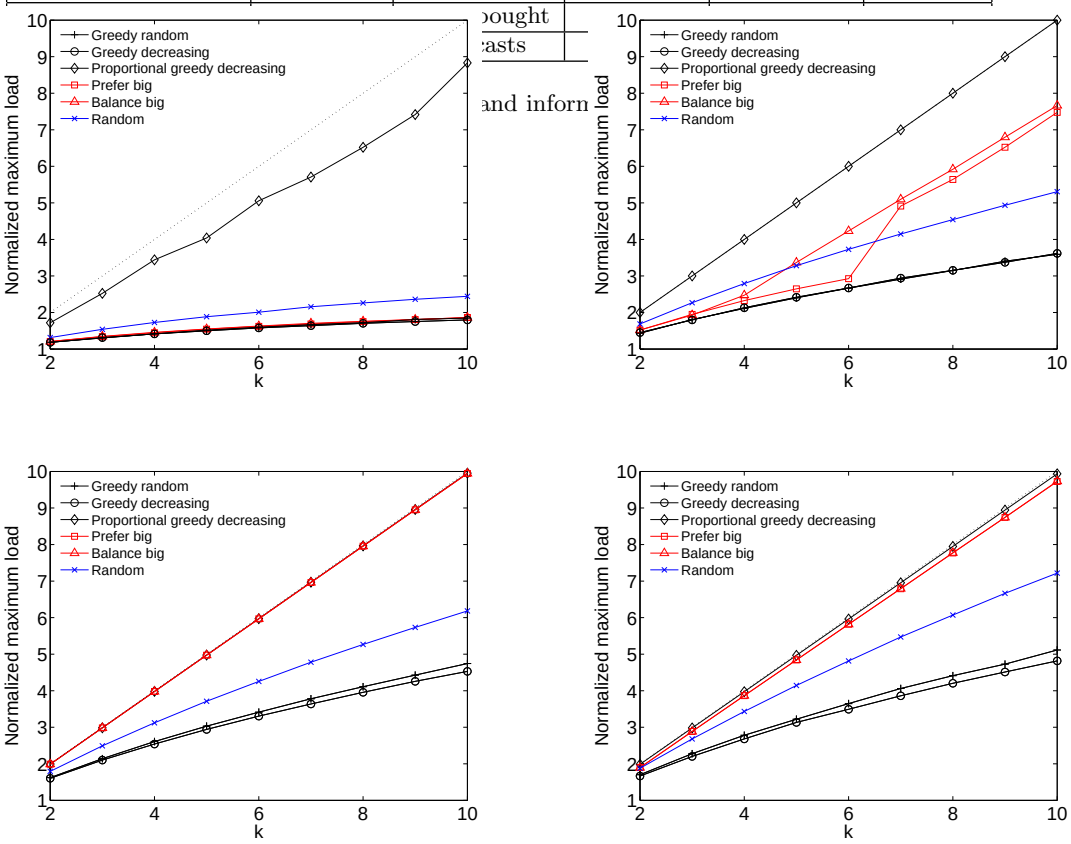
In our experiments, we considered partitioning of items onto k partitions for a range of values going from two to ten partitions. We report the maximum number of topics in a partition normalized by the cost of a perfectly balanced solution m/k, where m is the total number of topics.

Online Assignment Algorithms. We compared greedy to the following other online algorithms:

1. *All-on-One* assigns all items and topics to one partition.
2. *Random* randomly assigns each item to a partition.
3. *Balance Big* receives the items in a random order. It assigns the the large items to the least loaded partition, and the small items according to greedy. An item is considered large if it subscribes to more than 100 topics, and small otherwise.
4. *Prefer Big* receives the items in a random order. It keeps a buffer of up to 100 small items. When it receives a large item it puts it on the least loaded partition. When the buffer is full, it places all the small items according to greedy.
5. *Greedy Random* receives the items in a random order and assigns them to the partition they have the most topics in common with. This is the algorithm we analyzed. We allowed a slack of up to 100 in all cases.
6. *Greedy Decreasing* receives the items in decreasing order of the number of topics. It assigns each item to the partition it has the most in common with, using the same slack constraint as Greedy Random.
7. *Proportional Greedy Decreasing* receives the items in decreasing order of the number of topics. The probability an item is assigned to a partition is proportional to the number of common topics with the partition.

In addition to these online heuristics, we also tried some offline heuristics. We ran into multiple issues in this case, reported below, given the size of our data sets.

Dataset	Items	Topics	# of Items	# of Topics	# edges
Book Ratings	Readers	Books	107,549	105,283	965,949
Facebook App Data	Users	Apps	173,502	13,604	5,115,433



(c) Retail Data

(d) Zune Podcast Data

Figure 5: The normalized maximum load for various online assignment algorithms under different input bipartite graphs versus the numbers of partitions. The dotted line shows the normalized maximum load of All-in-One assignment.

Results. We found that greedy generally outperforms other heuristics (see Figure 5). Also, the performance of greedy is improved if items arrive in decreasing order of number of topics. This observation seems intuitive: the items with larger number of topics would provide more information about the underlying structure of the bipartite graph than the items with smaller number of topics. Interestingly, adding randomness to the greedy assignment made it perform far worse; most times Proportional Greedy approached the worst case scenario. The naive random assignment outperformed Proportional Greedy and regularly outperformed Prefer Big and Balance Big item assignment strategies.

We also tested offline approaches to this problem, such as hMetis [17], label propagation [30], and basic spectral methods. We found that label propagation and spectral methods are extremely time and memory intensive on our inputs, due to the large number of topics and item-topic edges. (The algorithms took more than a day to return, and often ran out of memory on a 16GB machine.) hMetis returns within seconds, however the assignments were not competitive—we note however that hMetis provides balanced hypergraph cuts, which are not necessarily a good solution to our problem. For instance, using the hMetis cuts, whether directly or as an input to the offline algorithm in Section 3, yielded a less efficient assignment than greedy, on all the above inputs.

6. RELATED WORK

Graph partition, or clustering, or community detection problems have been studied under many variants of the objective function and constraints, e.g. see a survey [14].

The min-max multi-way cut problem is related to traditional load balancing problems formulated as *bin packing* and *machine scheduling*, see a survey on online algorithms [3]. The key distinction from our work is that we

consider the more general case of submodular cost functions. Online algorithms have also been studied for various other assignment problems including packing of *virtual circuits*, e.g. [2] and [26], *multicast sessions*, e.g. [15], *virtual machines in data centers*, e.g. [6], and *dynamic data*, e.g. [21]. Much progress has been recently made on approximation algorithms for *online integer linear programming problems with packing and covering constraints*, e.g. see [8] and [4]. Common to all these problems is that they are either of bin-packing or minimum-congestion type where individual requests consume fixed amounts of resources. A general framework for *vector assignment problems* was formulated by [13] under assumption that cost functions are *convex*. A key difference from our work is that we consider submodular cost functions, thus *concave* in the framework of [13]. A related problem is *min-max multiway cut* problem, which was first introduced in [32]. Given an input graph, here the objective is to partition the set of vertices such that the maximum *number of edges* adjacent to a partition is minimized. This has some similarity to our problem; however, we consider a different submodular cost of a partition, where each topic contributes a unit cost if the partition contains at least one item associated with the given topic. More recently, the min-max multi-way cut problem was also studied with respect to *expansion* (the ratio of the sum of weights of edges adjacent to a part of a partition and the minimum between the sum of weights of vertices within the given part and outside of the given part), e.g. [5] [20], which propose approximation algorithms based on semi-definite programs. All these problems are different from the one we consider.

Another related problem is *balanced graph partition*: given a graph the goal is to find a balanced partition of the set of vertices that minimizes the total number of edges cut. The best known approximation ratio for this problem is polylogarithmic in the number of vertices [18]. Streaming algorithms for balanced graph partition problem were recently studied in [28] and [29]. Our work is different in that we consider a *hyper-graph* partition problem and the min-max multi-way cut objective with respect to submodular functions of the vertices. The balanced graph partition problem was also considered for the set of *edges* of a graph [7]. Our problem is related to the bi-criteria optimization problem considered therein, namely edge-partition problem with aggregation. Notable distinctions include that we consider a single-criteria optimization and a maximum of submodular cut costs and not a sum of submodular cut costs. A first theoretical analysis of streaming balanced graph partition was presented in [27] using a framework similar to the one we deploy for the problem of co-clustering. More precisely, this paper gives sufficient conditions for a greedy strategy to recover clusters of vertices in a variant of the stochastic block model. As in our case, the argument uses a reduction to Polya urn processes. There are two main differences from our work: conceptually, we consider a different problem, as we consider the recovery of clusters of topics in a hypergraph, as opposed to clusters of vertices in a graph. Second, technically, we require a two-step reduction to Polya urn processes: the first step shows that, roughly, item placement decisions based on intersection of topics can be coupled with a process deciding on item placement based on the number of items on each partition. The second step reduces the latter random process to a Polya urn game, and is more similar to the graph recovery case.

A popular problem is cluster recovery for graphs in the *stochastic block model*. Tight recovery conditions are known from the work in [24] and [22]. The conditions for recovery of stochastic blocks in [27] are sufficient, not necessary. One-pass streaming algorithms were also recently studied in [31]; their computation model is under weaker assumptions not requiring irrevocable vertex assignments and allowing for memory polynomial in the number of vertices.

7. CONCLUSION

We studied a hypergraph partition where the goal is to partition the set of vertices into a given number of partitions such that the maximum number of hyperedges incident to a partition is minimized. This is a natural graph clustering problem formulation which, in particular, arises in the context of assigning complex workloads in data centers. The formulation is a generalization of traditional load balancing that allows for tasks with overlapping sets of requirements.

We established novel theoretical results on approximation algorithms for this problem. In particular, we provided an approximation algorithm for the offline version of the problem that guarantees to find a good approximation for every instance with a small graph cut cost. For the online version of the problem, we established analytic guarantees for simple and natural greedy online assignment of items for input random bipartite graphs defining associations of items. This amounts to a set of sufficient conditions for recovery of a hidden co-clustering of vertices. The performance of this online item assignment is demonstrated by extensive experiments using synthetic and real-world input bipartite graphs.

There are several interesting questions for further study. One is the theoretical question of worst-case approximation guarantees for the offline version of the problem. The second is the study of upper and lower bounds for online algorithms, and the study of online algorithms under different relaxations of the input bipartite graphs. Another question of interest is to study strategies for dynamic graph inputs with addition and deletion of items and topics.

8. REFERENCES

[1] Amazon Web Services. Aws lambda, 2014.

[2] B. Awerbuch, Y. Azar, S. Plotkin, and O. Waarts. Competitive routing of virtual circuits with unknown duration. In *Proc. of ACM SODA '94*, 1994.

[3] Y. Azar. On-line load balancing. In *Online Algorithms - The State of the Art, Chapter 8*, pages 178–195. Springer, 1998.

[4] Y. Azar, U. Bhaskar, L. Fleischer, and D. Panigrahi. Online mixed packing and covering. In *Proc. of ACM-SIAM SODA '13*, 2013.

[5] N. Bansal, U. Feige, R. Krauthgamer, K. Makarychev, V. Nagarajan, J. SeffiNaor, and R. Schwartz. Min-max graph partitioning and small set expansion. *SIAM Journal on Computing*, 43(2):872–904, 2014.

[6] N. Bansal, K.-W. Lee, V. Nagarajan, and M. Zafer. Minimum congestion mapping in a cloud. In *Proc. of PODC '11*, 2011.

[7] F. Bourse, M. Lelarge, and M. Vojnovic. Balanced graph edge partition. In *Proc. of ACM KDD '14*, 2014.

[8] N. Buchbinder and J. S. Naor. The design of competitive online algorithms via a primal-dual

approach. *Foundations and Trends in Theoretical Computer Science*, 3(2–3), 2007.

[9] F. Chung, S. Handjani, and D. Jungreis. Generalizations of Polya's urn problem. *Annals of Combinatorics*, (7):141–153, 2003.

[10] I. S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 269–274, New York, NY, USA, 2001. ACM.

[11] I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 89–98, New York, NY, USA, 2003. ACM.

[12] E. Drinea, A. M. Frieze, and M. Mitzenmacher. Balls and bins models with feedback. In *Proc. of ACM-SIAM SODA '02*, 2002.

[13] L. Epstein and T. Tassa. Vector assignment problems: a general framework. *Journal of Algorithms*, 48(2):360 – 384, 2003.

[14] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(75), 2010.

[15] A. Goel, M. R. Henzinger, and S. A. Plotkin. An online throughput-competitive algorithm for multicast routing and admission control. *J. Algorithms*, 55(1):1–20, 2005.

[16] T. Karagiannis, C. Gkantsidis, D. Narayanan, and A. Rowstron. Hermes: clustering users in large-scale e-mail services. In *Proc. of ACM SoCC '10*, 2010.

[17] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. *VLSI Design*, 11(3), 2000.

[18] R. Krauthgamer, J. S. Naor, and R. Schwartz. Partitioning graphs into balanced components. 2009.

[19] A. Louis. The complexity of expansion problems. *Ph.D. Thesis, Georgia Tech*, 2014.

[20] A. Louis and K. Makarychev. Approximation algorithm for sparsest k-partitioning. In *Proc. of ACM-SIAM SODA '14*, 2014.

[21] B. M. Maggs, F. Meyer auf der Heide, B. Vöcking, and M. Westermann. Exploiting locality for data management in systems of limited bandwidth. In *Proc. of FOCS '97*, 1997.

[22] L. Massoulié. Community detection thresholds and the weak ramanujan property. In *Proc. of ACM STOC '14*, 2014.

[23] Microsoft. Scalable information stream processing by bing in support of cortana scenarios, 2014.

[24] E. Mossel, J. Neeman, and A. Sly. Reconstruction and estimation in the planted partition model. *Probability Theory and Related Fields*, pages 1–31, 2014.

[25] J. M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez. The little engine(s) that could: Scaling online social networks. *IEEE/ACM Trans. Netw.*, 20(4):1162–1175, 2012.

[26] H. Räcke. Minimizing congestion in general networks. In *Proc. of FOCS '02*, 2002.

[27] I. Stanton. Streaming balanced graph partitioning algorithms for random graphs. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 1287–1301. SIAM, 2014.

[28] I. Stanton and G. Kliot. Streaming graph partitioning for large distributed graphs. In *Proc. of ACM KDD '12*, 2012.

[29] C. E. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic. FENNEL: streaming graph partitioning for massive scale graphs. In *Proc. of ACM WSDM '14*, 2014.

[30] L. Wang, Y. Xiao, B. Shao, and H. Wang. How to partition a billion-node graph. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 568–579, March 2014.

[31] S.-Y. Yun, M. Lelarge, and A. Proutiere. Streaming, memory limited algorithms for community detection. In *Proc. of NIPS '14*, 2014.

[32] Z. Z. Svitkina and E. Tardos. Min-max multiway cut. In K. Jansen, S. Khanna, J. Rolim, and D. Ron, editors, *Proceedings of APPROX/RANDOM*, pages 207–218. 2004.

APPENDIX

A. DEFERRED PROOFS

A.1 Proof of Theorem 1

THEOREM 1. *Let f^* be the optimal cost for the min-max submodular graph cut problem. For any integer $k \geq 2$, given an α-approximation for the balanced hypergraph bisection problem, we get an $4[1 + \alpha(k - m/f^*)]$ approximation algorithm for min-max submodular graph cut.*

PROOF. Without loss of generality, we will assume $k = 2^\ell$ for some integer ℓ. Otherwise, we will lose a factor of 2 in the approximation.

Consider the task-requirement problem for the input $H = (N, E)$ with k machines. Let $N_1, N_2, \ldots N_k$ be the optimal solution of the task-requirement problem, let f^* be the value of this solution of the task-requirement problem, and let g^* be the number of edges cut in this solution.

Let $H' = (U, F)$ be the input graph in some application of the algorithm with $|F| = m'$. We will show there is a partition of H' with imbalance at most $(m' + f^*)/(2m')$ and cost at most g^*. Let $N_i = U \cap N_i$.

We know that $w_{H'}(N_i') = \sum_{v \in N_i'} w_{H'}(v) \leq f^*$. The number of edges with a vertex in N_i' is at most f^* and each edge adjacent to N_i' can contribute at most 1 to the weight of N_i'. An edge e contributes $1/|e|$ to the optimistic cost function of the $|e|$ vertices it is adjacent to.

Now we will use greedy load balancing to make our desired bipartition. Sort the N_i''s from largest to smallest in terms of $w_{H'}$, and place them one-by-one on the current smaller weighted part. Let N_j' be the piece last placed on the larger part. It was placed on the larger part, when its weight did not exceed the final weight of the smaller part. Therefore the imbalance between the two parts can be at most $w_{H'}(N_j') \leq f^*$. This bipartition can also only have edges that were cut in the original solution, so at most g^* edges were cut.

Our approximation algorithm for balanced hypergraph bisection will find a cut with at most αg^* edges and imbalance at most $(m' + f^*)/(2m')$, at each step. The larger part that is created will have at most $m'/2 + f^*/2 + \alpha g^*$ edges adjacent to it. Therefore, the largest part after ℓ iterations will

have at most

$$\frac{1}{2}\left(\frac{1}{2}\left(\frac{m'}{2}+\frac{f^*}{2}+\alpha g^*\right)+\frac{f^*}{2}+\alpha g^*\right)+\cdots$$

edges when the algorithm finishes. This simplifies to

$$\frac{m}{k}+\sum_{i=1}^{\ell}\frac{2^i f^*}{2k}+\sum_{i=1}^{\ell}\frac{2^i\alpha g^*}{k} \leq \frac{m}{k}+f^*+2\alpha g^*.$$

If we consider the number of cut edges in terms of f^*, we get that kf^* counts every cut edge at least twice, and all edges at least once; so $kf^*-m \geq g^*$. Also note that $\frac{m}{k} \geq f^*$. Therefore, we obtain a $2+2\alpha(k-m/f^*)$ approximation when k is a power of 2 and $4+4\alpha(k-m/f^*)$, otherwise.

The algorithm needs to know the value of f^*. This can be overcome by simply running the algorithm multiple times as we know f^* is bounded between m/k and m. □

A.2 Proofs Omitted from Section 4

LEMMA 10. *1. For $0<\epsilon<1$ constant, the number of requirements inside the cluster to which a task τ subscribes is in $[2(1-\epsilon)\log r, 2(1+\epsilon)\log r]$, w.h.p.*

2. The expected number of known requirements by time t is at least $r(1-\exp(-2t\log r/r))$.

3. For any time $t \geq r/\log^2 r$, the number of known requirements is at least $r/\log r$ and the number of singleton requirements is at least $r/(2\log r)$, both w.h.p.

PROOF. The first statement follows by straightforward application of Chernoff bounds. To bound the number of known requirements, notice that, at each step, a specific requirement r is sampled with probability p. Therefore, the probability that r has not been sampled by time t is $(1-p)^t$. Plugging in $p=2\log r/r$, it follows that the expected number of unknown requirements up to t is $r\exp(-2t\log r/r)$, which implies the second claim.

In particular, this number of unknown requirements by time $t=r/\log^2 r$ is at most $r/e^{2/\log r} \leq r(1-3/2\log r)$, by the Taylor expansion. Therefore, the expected number of known requirements up to t is at least $3r/(2\log r)$. By a Chernoff bound, it follows that the number of known requirements up to t is at most $r/\log r$, w.h.p., as required.

To lower bound the number of singleton requirements, notice that it is sufficient to lower bound the number of requirements that have been sampled exactly once up to and including time t. (Such requirements are necessarily singletons.) The probability that a requirement has been sampled *exactly once* is $tp(1-p)^{t-1}$. Since $t \geq r/\log^2 r$, we obtain that the expected number of requirements that have been sampled exactly once is at least $r/(\log re^{1/\log r}) \geq r/\log r$, for large enough r. Hence, the number of singletons up to this point is at least $r/(2\log r)$, w.h.p., which completes the proof. □

LEMMA 11. *Assume that the singleton requirement ratio at time t_0 is $\mu \leq 1/2+\epsilon$, for fixed $\epsilon<1$, and let $\phi(\epsilon)=\left(\frac{1/2+2\epsilon}{1/2-2\epsilon}\right)^2$. Then, the ratio between the requirement-to-task quotients of the two servers is in the interval $[1/\phi(\epsilon),\phi(\epsilon)]$, with high probability.*

PROOF. Without loss of generality, let partition 1 be the more loaded one at t_0, i.e., $\mu=\sigma_1/(\sigma_1+\sigma_2)$. Let T_1 be the set of tasks assigned to the first partition between times $r/\log^2 r$ and $r/\log r$, and T_2 be the corresponding set for partition 2. By the Lemma statement, we have that $\sigma_1/(\sigma_1+\sigma_2) \leq 1/2+\epsilon$ at t_0.

Given this bound, our first claim is as follows. If $\mu \leq 1/2+\epsilon$ at time t_0, then, for all times $r/\log^2 r \leq t \leq r/\log r$, we have that $\mu_t \leq 1/2+2\epsilon$, w.h.p. Also, $|T_1|/(|T_1|+|T_2|) \in [1/2-2\epsilon, 1/2+\epsilon]$, w.h.p.

We focus on the proof of the first statement above, and the second will follow as a corollary. Let us assume for contradiction the converse, i.e., that there exists a time step $r/\log^2 r \leq t \leq r/\log r$ for which $\mu_t > 1/2+2\epsilon$. We will show that, after time t, the relative gap in terms of singleton requirements between the two partitions will increase, with high probability, which contradicts our bound at time t_0.

For this, consider an incoming task τ. If this task is subscribing to a known requirement, which we call case 1, then it will be assigned by the intersection rule. In case 2, it will be placed uniformly at random on one of the partitions. To control this process, we split the execution from time t into blocks of $b=r/\log^4 r$ consecutive tasks. Notice that, by Lemma 10, there are at least $r/\log r$ known requirements after time $r/\log^2 r$, w.h.p. This implies that the probability that a task is *not* assigned by the intersection rule after this time is at most $(1-2\log r/r)^{r/\log r} \leq (1/e)^2$. Therefore, each incoming task is assigned by the intersection rule after this time, with at least constant probability.

Consider now the probability that a case-1 task gets assigned to partition 1, assuming that $\sigma_1/(\sigma_1+\sigma_2) > 1/2+2\epsilon$ at the beginning of the current block. This means that the task has more requirements in common with partition 1 than 2. By simple calculation, this probability is at least

$$\frac{1/2+2\epsilon}{1/2-2\epsilon+7b\log r/r+1/2+2\epsilon} \geq 1/2+7\epsilon/4,$$

where we have pessimistically assumed that *all the tasks* in the current block get assigned to the second partition, and that each such task contains at most $7/3\log r$ new requirements. (This last fact holds w.h.p. by Lemma 10.)

For a task i during this block, let X_i be an indicator random variable for the event that the task gets assigned to partition 1, and fix $X=\sum_i X_i$. We wish to lower bound X, and will assume that these events are independent—the fact that they are positively correlated does not affect the lower bound. We apply Chernoff bounds, to get that, w.h.p., $X \geq (1-\delta)7b\epsilon/4$, that is, the first partition gets at least $(1-\delta)7b\epsilon/4$ *extra* tasks from each block, where $0<\delta<1$ is a constant. On the other hand, the number of case-2 tasks assigned is balanced, since these tasks are assigned randomly. In particular, it can be biased towards partition 2 by a fraction of at most $(1-\delta)\epsilon/4$, w.h.p. We have therefore obtained that partition 1 obtains an *extra* number of tasks which is at least $3b\epsilon/2$ in each block, w.h.p. Summing over $\log^2 r$ blocks, we get that, over a period of $r/\log^2 r$ time steps, partition 1 gets at least $(1/2+3\epsilon/2)r/\log^2 r$ extra tasks, w.h.p.

Notice that, in turn, this task advantage also translates into a similar extra proportion of *new requirements* acquired during each block. In particular, we obtain that the first partition acquires an $(1/2+4\epsilon/3)$ fraction of the new re-

quirements observed in a block, w.h.p. Recall that, by assumption, at the beginning of the process, partition 1 already had a fraction of $(1/2 + 2\epsilon)$ singleton requirements. Therefore, the event that the singleton requirement ratio is balanced by at most $1/2 + \epsilon$ at t_0 has very low probability, as claimed. The proof of the second statement follows by a similar argument.

To complete the proof of Lemma 11, it is enough to notice that, by the previous claim, the ratio between the requirement-to-task quotients of the two partitions is bounded as $\frac{\sigma_1+\kappa}{\sigma_2+\kappa} \cdot \frac{q_2}{q_1} \leq \left(\frac{1/2+2\epsilon}{1/2-2\epsilon}\right)^2$, which completes the proof of Lemma 11. □

LEMMA 12. *Given a hidden cluster input* $\mathsf{Bp}(r, \ell, p, q)$*, for every* $t \geq t_0 = r/\log r$*, one of the following holds:*

1. *With high probability, the greedy algorithm with a cluster and two partitions can be coupled with a finite Polya urn process with parameter* $\gamma > 1$*, or*
2. *There exists a constant* $\rho > 0$ *such that the ratio between the number of singleton requirements on the two servers is* $> 1 + \rho$ *at time* t_0*.*

PROOF. We proceed by induction on the time $t \geq t_0$. We will focus on time $t_0 = r/\log r$, as the argument is similar for larger values of t. Notice that we have two cases at t_0. If there exists a constant $\rho > 0$ such that the ratio between the number of singleton requirement on the two servers is $> 1 + \rho$ at time t, then we are obviously done by case 2.

Therefore, in the following, we will work in the case where the load ratio between the two servers at time t_0 is $\leq 1 + \rho$. W.l.o.g, assume $1 \leq (\sigma_1 + \kappa)/(\sigma_2 + \kappa) \leq 1 + \rho$.

By Lemma 10, the number of singleton requirements at time $t \geq t_0$ is at least constant fraction of r, w.h.p., and it follows that there exists a constant $\epsilon > 0$ such that the singleton ratio at time t_0 is at most $1+\epsilon$. Also, the probability that a task with $3\log r/2$ distinct requirements does not hit any of these known requirements is at most $1/r^{3/2}$. Hence, in the following, we can assume w.h.p. that every incoming task is assigned by the intersection rule.

By Lemma 11, the ratio between the requirements-to-task quotients of the two partitions at time t_0 is at most $\left(\frac{1/2+2\epsilon}{1/2-2\epsilon}\right)^2$, w.h.p. We now proceed to prove that in this case the greedy assignment process can be coupled with a Polya urn process with $\gamma > 1$, w.h.p., noting that this part of the proof is similar to the coupling argument in [27].

By Lemma 10, for $t \geq r/\log r$ steps, at least $2r/3$ requirements have been observed, w.h.p. Therefore, the probability that a task with $3\log r/2$ requirements does not hit any of these known requirements is at most $1/r^{3/2}$. Hence, in the following, we can safely assume that every incoming task is assigned by the intersection rule.

More precisely, when a new task comes in, we check the intersection with the number of requirements on each server, and assign it to the partition with which the intersection is larger. (Or randomly if the intersections are equal.) Given a task τ observed at time $t \geq r/\log r$, let A be the number of requirements it has in common with machine 1, and B be the number it has in common with machine 2.

More precisely, fix $j \geq 0$ to be the size of the total intersection with either machine, and let a and b be the values of the intersections with machines 1 and 2, respectively, conditioned on the fact that $a + b = j$. Let δ be the advantage in terms of *requirements* of machine 1 versus machine 2, i.e. $(\sigma_1+\kappa)/(\sigma_1+\sigma_2+2\kappa) = 1/2+\delta$, and $(\sigma_2+\kappa)/(\sigma_1+\sigma_2+2\kappa) = 1/2 - \delta$, where κ is the number of duplicated requirements. We now analyze the probability that $a > b$.

We can see this as a one-dimensional random walk, in which we start at 0, and take j steps, going right with probability $(1/2 + \delta)$, and left with probability $(1/2 - \delta)$. We wish to know the probability that we have finished to the right of 0. Iterating over i, the possible value of our drift to the right, we have that

$$\Pr[a > b] = \sum_{i=[j/2]+1}^{j} \binom{j}{i} \left(\frac{1}{2}+\delta\right)^i \left(\frac{1}{2}-\delta\right)^{j-i} = \left(\frac{1}{2}+\delta\right)^{[j/2]+1} \sum_{i=0}^{[j/2]} \binom{j}{i} \left(\frac{1}{2}+\delta\right)^{[j/2]-i} \left(\frac{1}{2}-\delta\right)^i.$$

Similarly, we obtain that

$$\Pr[a < b] = \left(\frac{1}{2}-\delta\right)^{[j/2]+1} \sum_{i=0}^{[j/2]} \binom{j}{i} \left(\frac{1}{2}+\delta\right)^i \left(\frac{1}{2}-\delta\right)^{[j/2]-i}.$$

Since $\delta > 0$, we have that the sum on the right-hand-side of the first equation dominates the term on the right-hand-side of the second equation. It follows that

$$\frac{\Pr[a > b]}{\Pr[a < b]} > \frac{\left(\frac{1}{2}+\delta\right)^{[j/2]+1}}{\left(\frac{1}{2}-\delta\right)^{[j/2]+1}}.$$

Since the two quantities sum up to (almost) 1, we obtain

$$\Pr[a > b] > \frac{\left(\frac{1}{2}+\delta\right)^{[j/2]+1}}{\left(\frac{1}{2}+\delta\right)^{[j/2]+1} + \left(\frac{1}{2}-\delta\right)^{[j/2]+1}}.$$

Let δ' be the advantage that the first server has over the second *in terms of number of tasks*, i.e. $1/2 + \delta' = q_1/(q_1 + q_2)$. Using Lemma 11, and setting ϵ to a small constant, we obtain that $\delta \simeq \delta'$. We can therefore express the same lower bound in terms of δ'.

$$\Pr[a > b] > \frac{\left(\frac{1}{2}+\delta'\right)^{[j/2]+1}}{\left(\frac{1}{2}+\delta'\right)^{[j/2]+1} + \left(\frac{1}{2}-\delta'\right)^{[j/2]+1}}.$$

The lower bound on the right-hand-side is the probability that the ball goes in urn 1 in a Polya process with $\gamma = [j/2]+1$. Importantly, notice that, in this process, we are assigning balls (tasks) with probability proportional to the number of *balls (tasks)* present in each bin, and have thus eliminated requirements from the choice. Let β_t be the proportion of singletons at time t, i.e. $(\sigma_1 + \sigma_2)/r$. We can then eliminate the conditioning on j to obtain that

$$\Pr[A > B] \geq \sum_{j=1}^{d} \binom{d}{j} (\beta_t/r)^j (1 - \beta_t/r)^{d-j} \Pr[a > b|j]. \quad (2)$$

The only case where greedy is coupled with a Polya urn process with undesirable exponent $\gamma \leq 1$ is when $j \leq 1$. However, since a task has at least $3\log r/2$ distinct requirements, w.h.p., and $t \geq t_0 = r/\log r$, the probability that we hit $j \leq 1$ requirements is negligible. Therefore we can indeed couple our process to a finite Polya urn process with $\gamma > 1$ at time t_0 in the case where the singleton ratio at t_0 is at

most $1/2+\epsilon$, for ϵ a small constant. We can apply the same argument by induction for all times $t \geq t_0$, noticing that, once the load ratio is larger than a fixed constant, it never falls below that constant, except with low probability. □

LEMMA 13. *Consider a time $t \geq r/\log r$ with $\sigma_1/(\sigma_1+\sigma_2) \geq 1/2+\epsilon$, where $\epsilon > 0$ is a constant. Then, after time t, greedy will assign all tasks to server 1, with high probability.*

PROOF. We split the proof into the following two claims. The first is that, at time $t_0 = r/\log r$, there are at least $r/2e$ singleton requirements in the system, with high probability.

For the proof, notice that, at time t_0, there are r (non-distinct) requirements drawn by the requirements. Since these requirements are chosen randomly at every step, we can model their choices as r balls thrown into r bins, chosen at random. In particular, $r/e^{\frac{r-\log r}{r}} \simeq r/e$ requirements that have been seen *exactly* once, i.e., requirements that are only required by a single task. With high probability, we have at least $r/2e$ such requirements. Since that task can only be assigned to one partition, it follows that such requirements cannot be duplicated. Therefore, we have at least $r/2e$ singleton requirements in the system, w.h.p.
The second claim completes the argument. We show that, under the above assumptions, greedy will assign each task observed after time t_0 to server 1, w.h.p.

For the proof, notice that, since there are $\Theta(r)$ singleton requirements in the system at time t_0, it follows that there exist constants c_1 and c_2 with $c_2 > c_1$ such that the first server has r/c_1 singleton requirements, and the second has r/c_2 singleton requirements. Since each task has $\geq 3\log r/2$ requirements, we can use Chernoff bounds to obtain that the probability that a task sees at least as many singleton requirements from the second (less loaded) server as from the first (more loaded one) is exponentially low. We conclude that, with high probability, all future tasks from the cluster will be assigned to the first server after time t_0. □

LEMMA 14. *Given a single cluster instance in $\mathsf{Bp}(r,\ell,p,q)$ to be split across k partitions by greedy, the algorithm will eventually concentrate the cluster onto a single partition.*

PROOF. Let us now fix two bins A and B. Notice that the argument of Lemma 12 applies, up to the point where we compute $\Pr[A > B]$ in Equation 2. Here, we have to condition on either A or B having the maximum number of intersections, i.e., replacing $\Pr[\#intersections = j] = \binom{d}{j}(\sigma_t/r)^j(1-\sigma_t/r)^{d-j}$ with $\Pr[\#intersections = j|\ A \text{ or } B \text{ is max}]$. Notice that the coupling still works for $j \geq 2$. Therefore, it is sufficient to show that $\Pr[\#intersections \in \{0,1\}] \geq \Pr[\#intersections \in \{0,1\}|\ A \text{ or } B \text{ is max}\ ]$.

This holds since the event (A or B is max) implies that the intersection is less likely to be empty or of size 1. Therefore, the argument reduces to the two bin case. □

LEMMA 15. *Assume a clustered bipartite graph G with $\ell \geq k\log k$, $p \geq 2\log r/r$, and $q = 0$, to be split onto k partitions with no balance constraints. Then, w.h.p., greedy ensures balanced recovery of G. Moreover, the maximum number of requirements per partition in this case is upper bounded by $(1+\beta)r\ell/k$, w.h.p., where $\beta < 1$ is a constant.*

PROOF. Notice that, since the clusters are disjoint and $q = 0$, the requirements must be disjoint. Also, since there is no balance constraint, the clusters and their respective requirements are independent. Fix an arbitrary cluster C_i. Let t_i be the first time in the execution when we have observed $2r/\log r$ tasks from C_i. By Theorem 3, after time t_i there exists a partition P_j such that all future tasks associated to this hidden cluster will be assigned to P_j, w.h.p. Also, note that, by Lemma 10, the expected number of requirements from this cluster that may have been assigned to other partitions by time t_i is at most $r(1-1/e^2)$, which implies that at most $8m/9$ total requirements may have been assigned to other partitions by this time, w.h.p.

To examine the maximum partition load, we model this process as a balls-into-bins game in which $\ell = k\log k$ balls (the clusters) are distributed randomly across k bins (the partitions). The expected distribution per bin is of ℓ/k clusters, and, by Chernoff bounds, the maximum load per bin is $(1+\alpha)\ell/k$, with high probability in k, where $0 < \alpha < 1$ is a small constant. This means that a partition may receive number of requirements of $(1+\alpha)r\ell/k$ from the clusters assigned to it. To upper bound the extra load due to duplicates, first recall that at most $8m/9$ total requirements from each cluster may be duplicated, w.h.p. In total, since clusters are distinct, we obtain that $8r\ell/9$ total requirements will be duplicated, w.h.p. Since these duplicates are distributed uniformly at random, a partition may receive an extra load of $(1+\alpha)8r\ell/9k$ requirements, w.h.p. Choosing small α, we get that the maximum load per partition is bounded by $(1+\alpha)r\ell/k + (1+\alpha)8r\ell/9k \leq 1.9r\ell/k$. We contrast this to the factor obtained by random assignment. □

LEMMA 16. *Given $q < \log r/(rk)$, then with high probability the greedy process can be coupled with a greedy process on the same input with $q = 0$, where $r/\log r$ requirements have been observed for each cluster.*

PROOF. We couple the two processes in the following way. We consider a hidden cluster graph G built with $q = 0$, and a copy of that graph G' where $q = \log r/(rk)$, running the algorithm in parallel on the two graphs. Notice that we can view this process on a task-by-task basis, where in the $q = 0$ copy the algorithm gets presented with a task τ, while in G' the algorithm gets presented with a variant τ' of τ which also has out-of-cluster requirements, chosen uniformly at random from an arbitrary set Q of at most $r/2$ requirements.

The key question is whether the greedy assignments are the same for tasks τ and τ'. We prove that this is indeed the case, with high probability. In particular, we need to show that, w.h.p., the outside requirements are not enough to change the decision based on the intersection argmax.

Given a task τ' in G' which belongs to cluster C_i, notice that, by Lemma 10, it has at least $3\log r/2$ distinct requirements in C_i, w.h.p. Let t_i be the first time when at least $r/\log r$ tasks from C_i have been observed. After time t_i, using Chernoff bounds and the pigeonhole principle, the size of the intersection of τ with one of the k clusters must be of at least $(1-\alpha)(1-1/e)3\log r/2k$, w.h.p., where $\alpha > 0$ is a constant.

We now bound the number of requirements that τ has outside C_i. Since $q < \log r/rk$, it follows that τ may have at most $(1+\beta)\log r/k$ requirements outside C_i, w.h.p., where β is a constant. For small α and β, we get that the number of home cluster requirements of τ exceeds the number of outside requirements, w.h.p. In turn, this implies that the two random processes can be coupled for each cluster starting with t_i, as claimed. □

LEMMA 17. *On a hidden cluster input, greedy without capacity constraints can be coupled with a version of the algorithm with a constant capacity constraint, w.h.p.*

PROOF. We can model the assignment process as follows: during the execution, each of the ℓ clusters has its tasks assigned randomly (at the beginning of the execution), then converges to assigning tasks to a single server. If we regard this from the point of view of each server i at some time t, there is a contribution R_i of requirements which comes from tasks in clusters that are still randomly assigned at t, and a contribution F_i of requirements coming from tasks in clusters that have converged. Notice that both these contributions are *balanced across servers*: each partition has the same probability of getting a random cluster; also, since clusters are assigned independently and $\ell \geq k \log k$, the weight coming from converged clusters is also balanced across partitions. Using concentration bounds for each contribution in turn, it follows that the maximally loaded partition is at most a constant fraction more loaded then the minimally loaded one, w.h.p. $\square$