Optimizing Iterative Algorithms for Social Network Sharding

Zishi Deng
Computer Science & Eng.
New York University
New York, US
zd475@nyu.edu

Torsten Suel
Computer Science & Eng.
New York University
New York, US
torsten.suel@nyu.edu

***Abstract*—There has recently been significant interest in applications that require computations on massive graph structures, including scenarios where the graph is too large to be processed on a single machine. In this case, the graph needs to be partitioned into subgraphs that can be assigned to individual machines, in a process called graph or social network sharding. Given the sizes of the graphs involved, it is necessary or at least highly desirable that the partitioning itself can also be performed in a distributed manner, instead of running a sequential partitioning algorithm on a single node.**

We study such distributed algorithms for graph sharding, where the goal is to create subgraphs of roughly equal size that minimize the number of edges crossing subgraph boundaries. In particular, we focus on two well-known approaches that can be efficiently implemented in MapReduce and related distributed computing paradigms: the Balanced Label Propagation algorithm of Ugander and Backstrom, and the method of Duong et al. based on the Bayesian Stochastic Block Modeling approach of Hofman and Wiggins. Our contributions are as follows: (1) We perform the first direct experimental comparison of the two approaches, which were independently proposed and published. (2) We propose and evaluate several enhancements of Balanced Label Propagation that result in improved graph shardings. (3) We propose and evaluate hybrid methods that perform label propagation both on individual nodes, as suggested by Ugander and Backstrom, and on stochastic blocks inferred using the approach of Duong et al.

***Index Terms*—social networks, community detection, network sharding, label propagation, graph partitioning**

I. Introduction

Over the last few years, social networks have seen rapid growth in data sizes and system loads. For example, Facebook reported 1.45 billion daily active users in March 2018. The resulting graph structures, with billions of nodes and hundreds of billions of edges, are too large to be stored on a single machine, and thus need to be partitioned across many machines. The problem of finding a good partitioning of the graph, called *graph sharding*, is the focus of this paper. In particular, the problem is to find a partitioning of the graph into shards satisfying constraints on shard sizes, such that the number of non-local edges, i.e., edges connecting nodes located in different shards, is minimized.

978-1-6654-3902-2/21/$31.00 ©2021 IEEE

Graph partitioning has of course been studied for a number of decades, and many basic problems are known to be NP-hard. This includes the Minimum Bisection problem of finding a partition of a graph into two shards of equal size such that the number of crossing edges is minimized [8], and the Balanced Graph Partitioning problem of partitioning a graph into k roughly equal size components while minimizing the number of crossing edges [1]. For the latter problem, there is no constant-factor approximation unless $P = NP$, and the best upper bound achieves an approximation factor of $O(\lg^2 n)$ [1]. Despite these theoretical limits, a number of heuristic algorithms have been proposed that achieve impressive results on many real-world graphs. An example is the widely used METIS software package based on multi-level graph partitioning schemes in [12].

1.1 Social Network Sharding.

However, many of these approaches are not suitable for the extremely large graphs encountered in social networking applications. These new scenarios, often called graph or social network sharding, pose several additional challenges. First, algorithms need to scale to very large graphs. Second, they need to allow for a distributed implementation, preferably in popular frameworks such as MapReduce [6] or Pregel [15]. Third, they should allow efficient updates or iterative refinements to the current solution. Finally, they need to perform well on the types of graphs commonly encountered in social networks.

In this paper, we study social network sharding, where we partition a graph into m shards such that each shard i contains approximately c_i nodes while minimizing crossing edges. Thus, different shards can have different target sizes, and we allow some bounded deviation from these sizes. We focus on two well-known approaches to this problem, the Balanced Label Propagation (BLP) method of Ugander and Backstrom [18], and the method of Duong et al. [7] based on the Bayesian Stochastic Block Modeling approach (SBM) of Hofman and Wiggins in [9]. These methods were independently and concurrently proposed in 2013, and both allow for highly efficient distributed implementations in MapReduce and related frameworks. However, they are also technically quite different.

In particular, BLP [18] starts from an initial assignment of nodes to shards, and redistributes nodes in a series of iterations. In each iteration, nodes are moved if this decreases the number of adjacent non-local edges, subject to the solution of a Linear Program that guarantees size constraints on the shards. In contrast, SBM [7] assumes a Stochastic Block Model for graph generation and then uses an iterative algorithm of Hofman and Wiggins [9] to infer a much larger number of densely connected subgraphs (or blocks). In the second phase, these blocks are greedily assigned to shards. For both approaches, the running time is dominated by an iterative computation where nodes are relabeled based on a form of neighborhood voting, and both map naturally to distributed computing paradigms such as MapReduce [6] or Pregel [15].

1.2 Our Contribution.

Our goal in this paper is to experimentally compare the BLP and SBM approaches, to analyze their limitations, and to study optimizations of the BLP and SBM and combinations of the two approaches. To do so, we re-implemented both algorithms from scratch, and compared the partition quality of SBM to those of the METIS package and the BLP approach with several initial assignments, including initial assignments based on METIS and SBM. We then implemented several extensions of BLP that periodically disrupt the BLP relabeling process, including one that iteratively moves blocks of nodes rather than individual nodes between shards, and one that applies a version of the Kernighan-Lin (KL) algorithm [13]. Our results suggest that initializing BLP with SBM and periodically running KL swaps achieves good partition quality on large social networks.

The remainder of this paper is organized as follows. Next, we give more detailed descriptions of the BLP and SBM approaches, and discuss related work. Section 3 describes our optimizations to the BLP approach, and Section 4 contains our experimental evaluation. Finally, Section 5 provides concluding remarks.

II. Background and Related Work

We now describe the BLP and SBM approaches in more detail, and discuss other related work. But first, we formally define the social network sharding problem, using the formulation in [18].

Problem Definition: Given an undirected graph $G = (V, E)$, an integer m, and lower and upper size constraints $S_1, \ldots S_m$ and $T_1, \ldots, T_m$, the goal is to partition V into m shards $V_1, \ldots, V_m$ such that $S_i \leq |V_i| \leq T_i$, and the number of non-local edges (with endpoints in different shards) is minimized [18].

Unless stated otherwise, we choose S_i and T_i a few percentages below and above $|V|/m$, respectively, aiming for shards of roughly equal size. However, all approaches also apply when shards vary in size.

2.1 The BLP Approach

We now describe Ugander and Backstroms' Balanced Label Propagation (BLP) algorithm [18] in more detail. We define the local degree of a node as the number of adjacent edges that connect to nodes in the same shard. Thus, the goal is to maximize the sum of the local degrees of all nodes.

Suppose we have a current assignment of nodes to shards. If by moving a node v from its current shard to another shard, we can increase its local degree from d to some $d' > d$ – assuming all other nodes stay put – then we say that the move has a benefit of $d' - d$. The idea is that in each iteration, we determine for each node its most profitable move, assuming one exists, and breaking any ties (say, at random). We call this a node's preferred move. A natural approach would try to move each node according to its preferred move.

However, the result might violate the shard size constraints given by the S_i and T_i. Instead, we aim to select a subset of preferred moves with maximum total benefit (sum of benefits of the moves) such that size constraints are satisfied afterwards. This selection problem can be stated as follows [18]:

$$\begin{aligned} &\max_X \sum_{i,j} f_{ij}(x_{ij}) \ s.t. \\ S_i - |V_i| \ &\leq \ \sum_{i \neq j} (x_{ij} - x_{ji}) \ \leq \ T_i - |V_i|, \ \forall i \\ &0 \ \leq x_{ij} \ \leq \ P_{ij}, \ \forall i, j. \end{aligned} \tag{1}$$

Here, the value x_{ij} of the solution is the number of nodes that are selected to move from V_i to V_j, $f_{ij}(x_{ij})$ is the total benefit from executing the x_{ij} preferred moves from V_i to V_j with the highest benefits, and P_{ij} is the total number of preferred moves from V_i to V_j. Note that in the above formulation, the x_{ij} are integers. However, as shown in [18], $\sum_{i,j} f_{ij}(x_{ij})$ is a piecewise-linear concave function, allowing the above constraints to be rewritten into a Linear Program as follows:

$$\begin{aligned} &\max X, Z \sum_{i,j} z_{i,j}, s.t. \\ S_i - |V_i| \ &\leq \ \sum_{i \neq j} (x_{ij} - x_{ji}) \ \leq \ T_i - |V_i|, \ \forall i \\ &0 \ \leq x_{ij} \ \leq \ P_{ij}, \ \forall i, j \\ &-a_{ijk} \times x_{ij} + z_{ij} \leq b_{ijk}, \forall i, j, k, \end{aligned} \tag{2}$$

where the a_{ijk} and b_{ijk} are derived from the utility functions f_{ij} as the gradients and y-intercepts of the linear pieces of f_{ij}. This LP can be solved using a standard LP solver such as lpsolve [3]. Thus, each iteration of BLP involves the following steps:

(1) Determine the preferred moves of all nodes, compute the functions $f_{ij}(x_{ij})$ by a prefix sum over the sorted list of preferred moves from V_i to V_j, and compute the resulting a_{ijk} and b_{ijk}.
(2) Solve the resulting LP using an LP solver.
(3) Move nodes according to the x_{ij} of the LP solution, and update any data structures.

Steps (1) and (3) map naturally to a distributed computing paradigm such as MapReduce, but Step (2) requires an LP solver, and most available solvers are sequential. Thus, it is important that this does not become a performance bottleneck. The number of variables in the above LP is at most $2m(m -$

1), while the number of constraints is bounded by $2m^2 + Km(m-1)$, where m is the number of shards and K is the maximum number of linear pieces of any of the f_{ij}. State-of-the-art LP solvers can deal efficiently with LPs of hundreds of thousands to a few million constraints, and as argued in [18] they can easily handle most interesting cases. For even larger numbers of shards and linear pieces, [18] also proposed an approximation of the utility gain f_{ij} that performs well in practice.

2.2 The SBM Approach.

We now describe the SBM algorithm of Duong et al. [7] based on the Stochastic Block Model. Recall that in BLP, nodes were directly assigned to shards in a series of iterations that maximize the number of internal edges. In SBM, by contrast, we first identify smaller clusters of nodes where each cluster has a lot of internal edges. Then clusters are assigned to shards with a simple greedy method.

The main challenge in SBM is to infer the clusters. This is done using a random graph model called Stochastic Block Model [10], which assumes an underlying block or community structure in the graph where edges between nodes in the same block are more likely to exist than edges between different blocks. This model is a good match for social networks, where connections are more likely to exist between people who share underlying traits such as interests, background, location, or offline social connections. An algorithm by Hofman and Wiggins [9] proposed an iterative algorithm that, given an observed graph, infers the most likely underlying block structure under a Bayesian approach.

Block Inference.

This algorithm is at the heart of the SBM method, and thus we briefly outline it; for full details, see [7], [9]. The input is a graph $G = (V, E)$ and the intended number of communities (blocks) K. The output is a community mapping $\vec{z}$ with biases for cluster sizes $\vec{\pi}$ and edge existence bias $\vec{\theta}$. Hence, $p(\vec{\pi}, \vec{\theta}, \vec{z}|G)$ is the probability of a certain output given observed network G. Moreover,

$$p(G, \vec{z}|\vec{\pi}, \vec{\theta}) = \theta_+^{m_{++}}(1-\theta_+)^{m_{+-}}\theta_-^{m_{-+}}(1-\theta_-)^{m_{--}}\prod_{k=1}^{K}\pi_k^{n_k}, \tag{3}$$

where m_{++} and m_{+-} are the total number of edges and non-edges within communities, respectively, while m_{-+} and m_{--} are the total numbers of edges and non-edges across communities, respectively. Using Bayes' Theorem, we get:

$$p(\vec{\pi}, \vec{\theta}, \vec{z}|G) = \frac{p(G, \vec{z}|\vec{\pi}, \vec{\theta})p(\vec{\pi}, \vec{\theta})}{p(G)}, \tag{4}$$

where $p(G, \vec{z}|\vec{\theta}, \vec{\pi})$ can be estimated from G using (2.3), $p(\vec{\pi}, \vec{\theta})$ is the prior belief on the parameter values, and $p(G)$ is a constant representing the marginal likelihood of observing G under the given prior belief.

The algorithm initially assigns each node to a random community, and then repeatedly takes a discounted vote over a node's neighbors' assignments using weights J and J', where J weighs the local term related to edges within clusters, and J' and $\vec{h}$ balance this with global discounts based on the number of possible edges within clusters, and cluster sizes, respectively. J and J' are defined based on a set of hyperparameters that is adjusted between iterations. By adding observed counts (m_{++}, m_{+-}), (m_{-+}, m_{--}) to pseudo counts (α_+0, β_+0), (α_-0, β_-0), the posterior calculations are reduced in [7] to simple algebraic updates on hyperparameters $(\alpha_+, \beta+)$, and (α_-, β_-). Edge probabilities θ_+ and θ_- within and between communities are modeled as Beta distributions, while cluster size is a Dirichlet distribution over $\vec{\pi}$.

Each iteration can be implemented in time $O(V + E)$, and maps naturally to a MapReduce paradigm. Thus, very large graphs can be processed efficiently.

Mapping Clusters to Shards

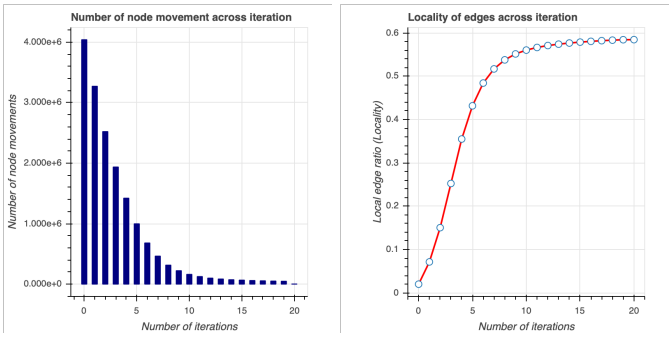
After running the discounted voting algorithm to convergence, we get a number H of clusters of various sizes. Usually, we have $H << K$ as many of the K randomly initialized clusters become empty, while others increase in size. Recall that the goal of graph partitioning is to take a graph $G = (V, E)$ and partition it into a configuration with m partitions V_i such that $S_i \leq |V_i| \leq T_i$. Duong et al [7] describe a greedy algorithm $BlockShard$ that maps the H clusters to k shards. The algorithm completely fills each shard with clusters before moving to the next shard. If the current shard is empty, the largest unassigned cluster is picked; otherwise, the unassigned cluster that is most tightly connected with nodes already in the shard is selected. If a selected cluster does not completely fit into the current shard, it spills over into adjacent shards that then become the current shard. While the algorithm is sequential, it is very fast when using a suitable heap-based data structure for selecting the next cluster.

2.3 Implementation and Preliminary Results.

We re-implemented BLP [18] and SBM [7] in C^{++}. More complete details and results are provided in subsequent chapters, and the link to the source code is also provided. For now, we present some preliminary results to illustrate the two algorithms and their limitations. For these runs, we use a LiveJournal social graph of almost 4M nodes and 34.68M edges, and partition into 50 equal-sized partitions with $\pm 3\%$ leniency (i.e., $S_i = 0.97 \cdot |V|/m$ and $T_i = 1.03 \cdot |V|/m$).

We see that BLP almost converges in about 20 iterations, starting from a random assignment. After a few iterations, only few nodes move between shards, and in the end we achieve about 60% local edges.

Next, we compare both methods on LiveJournal, for 10, 30, 50, 70, and 90 shards. Results are shown in Figure 2. We see that SBM significantly outperforms BLP with random initialization for all data points, and in particular for larger numbers of partitions. As expected, locality decreases with



(a) Node moves / Iterations (b) Locality / Iterations

Fig. 1: BLP on LiveJournal, for 50 shards.

more shards. Similar relative behavior was observed on other graphs, as will be seen later.

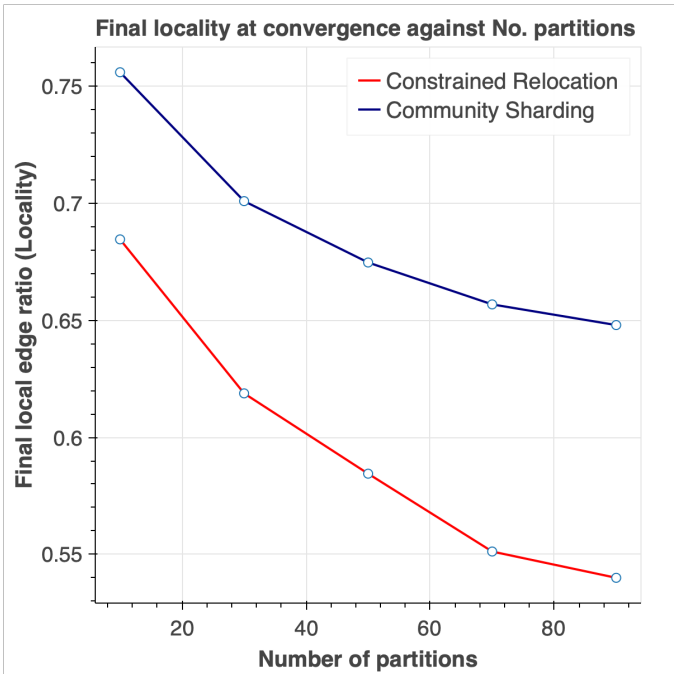


Fig. 2: Performance comparison of BLP and SBM on LiveJournal, for varying numbers of shards.

2.4 Limitations of BLP and SBM.

We now discuss some limitations of the two approaches that we plan to address. First, as also observed in [18], the performance of BLP depends significantly on the initial assignment. In [18], an initial assignment based on geography is shown to significantly outperform random assignment; however, this data is not available to us. Given that SBM outperforms BLP in Figure 2, it would be natural to use SBM to get an initial assignment for BLP.

Second, BLP may converge while there are still many potentially profitable moves, since it only considers the most profitable move for each node. Consider a case where node A wants to move from shard V_i to V_j, while node B would prefer to move from shard V_j to V_k, although it would also get some more limited benefit from moving to V_i. If these are the only available moves, then BLP is stuck in the current configuration if capacity constraints do not allow V_j and V_k to grow in size, while it would actually make sense to swap A and B. Note that this decision to only consider the most profitable move is inherent to the reduction to an LP in the approach. A similar case involves a benefit of, say, 4 for moving A from V_i to V_j, and a benefit of -2 for moving B in the other direction – this would lead to a better assignment but is not considered under BLP.

Third, BLP only considers the impact of moving individual nodes. Imagine a clique of, say, ten nodes currently at V_i where each node is also connected to two nodes (not necessarily the same two nodes) at V_j, Under BLP, we would not move any of these nodes to V_j, even though moving all of them together would in fact be profitable. Thus, it would be desirable to have a way to move clusters of nodes, say as identified by SBM.

In summary, BLP often gets stuck in local optima that are far away from the best solution. SBM uses a very smart algorithm for finding clusters, but the subsequent greedy assignment to shards seems simplistic. Also, while BLP can use additional iterations to refine the assignment under graph updates, it is unclear how to support updates in SBM.

2.5 Other Related Work.

As discussed, our focus here is on optimizing the approaches in [7], [18]. There are many related and interesting avenues of research that we do not try to compare to. For example, work in [11] considers partitioning schemes that support very fast updates to the graph structure. There is a growing body of research on streaming approaches to graph partitioning that only scan the graph structure a single or a small number of times; see, e.g., [2], [17]. We are also not trying to optimize the wall-clock time of our methods, but focus on the quality of the resulting partitioning – we refer to [16] for highly efficient implementations of the BLP approach. Finally, [4] considers the related problem of minimizing the number of cut vertices in a graph.

III. Optimizations of Graph Sharding Techniques

We focus on optimizing the above two graph sharding algorithms, by addressing some of the above-mentioned limitations. We note that BLP has an "initialization + refinement" framework that provides great flexibility to combine multiple initialization and refinement techniques to produce the best result. Furthermore, its iterative process enables a quick one-pass refinement as the graph changes. Within this "initialization + refinement" framework, we explore three different initialization techniques - random, SBM, and METIS, as well as three modifications to the refinement process - probability-based disruption, community relocation, and pairwise swaps based on Kernighan-Lin.

Definition of Convergence.

To introduce these modifications, we need to define convergence of the iterative refinement. We define $Y(i)$ as the

fraction of local edges $((loc_{E(i)})/E)$ in graph $G = (V, E)$ after i iterations of iterative refinement. We say that an iterative process has converged if the increase in locality over the previous d iterations falls below a threshold, or formally:

$$\frac{Y(i) - Y(i-d)}{Y(i-d)} < \varepsilon_1 \tag{5}$$

We usually use $d = 1$. The purpose of the three new types of refinements is to act as disruptors that are invoked whenever the standard BLP relocation process converges, as determined by ε_1. At this point, we call one iteration of the disruptor, and then return to the standard BLP process until it converges again. We also need to define a condition for the convergence of the overall algorithm, as follows: whenever the standard BLP process converges again after a disruption, we compare the current state to that just before the disruption, and terminate the overall algorithm when that improvement in locality is below a certain value ε_2. Thus, convergence is controlled by ε_1 and ε_2.

3.1 Probability-Based Disruption.

The purpose of this disruptor is to randomly shake up the current shard assignment, by throwing nodes out of shards to make space so that other, more suitable, nodes can move in. This is achieved by selecting nodes from each shard and placing them into a common pool, and then randomly assigning nodes in the common pool back to the shards.

We considered several methods to select nodes into the random pool. Our first approach simply selects a certain percentage of nodes from each shard at random. However, this did not perform well overall, in that after we converge again, we end up roughly in the same place as before the disruption. Thus, we need to be smarter about identifying nodes to place into the pool. For example, we might want to select nodes with low local locality, that is, nodes with a small value of $lr(n) = \frac{loc(n)}{degree(n)}$. Thus, an alternative approach could simply select nodes based on $lr(n)$, by picking in each shard the nodes with the lowest value of $lr(n)$.

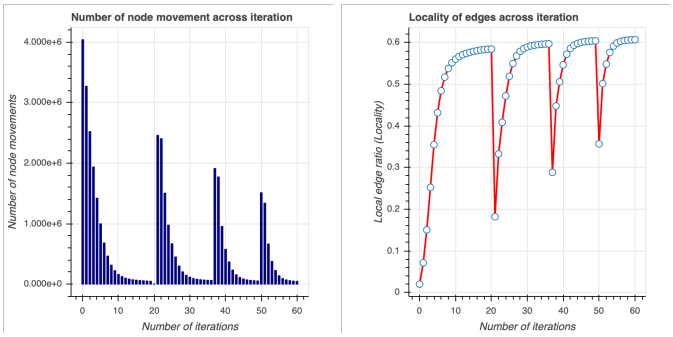
We found that an approach that combines picking at random with selection by $lr(n)$ does even better. In this approach, we select each node n with probability $prob(n) = \alpha(1 - lr(n))$, where α determines the magnitude of the disruption. Thus, nodes with low locality are more likely to be picked for the pool. We show pseudocode for the method in Algorithm 1:

We implemented all three approaches, but focus here on the last one as it dominates the other two methods. We start out with a fairly large α that is decreased by a decay factor with each disruption. In Figure 3, we show the resulting performance on the LifeJournal graph.

This method reaches global convergence after 61 iterations, and the final locality of 0.607 improves modestly over BLP's value of 0.584, but is still far away from the result of 0.675 obtained by SBM. We will later evaluate this approach in more detail, including when applied to SBM as an initial shard assignment.

Algorithm 1 ProbDisrupt

```
for Shard V_i ∈ V_{1...k} do
  moveCount_V_i ← 0
  for node V_i[n] ∈ V_i do
    prob(V_i[n]) ← α(1 − lr(n))
    if rand(0, 1) < prob(V_i[n]) then
      put node V_i[n] into the common pool
      moveCount_V_i ← moveCount_V_i + 1
    end if
  end for
end for
```



(a) Node moves / Iterations (b) Locality / Iterations

Fig. 3: ProbDisrupt performance on LiveJournal

3.2 Clustered Constrained Relocation.

Our next idea for disruption attempts to address the third limitation of BLP, which is only moving individual nodes, which can leave clusters of nodes stranded in the wrong shard. The basic idea is to extend the Constraint Relocation in BLP to clusters, as follows: First, we use SBM to compute clusters. Then, once BLP has converged as determined by ε_1, we run one round of cluster movement, where each cluster is treated as an individual node in BLP. That is, we basically collapse each cluster into a single node, with suitable local and non-local degrees and a size corresponding to the number of nodes in the cluster. We then determine for each such cluster node which other shard it would like to move to, if any, and then set up a slightly modified LP to pick a set of moves that satisfies capacity constraints.

Here, the modification of the LP is to take the size of a cluster into account in the capacity constraints. One challenge that arises is that the solution returned by the LP solver may tell us to move a fraction of a cluster to satisfy capacity constraints; in this case we "randomly round" the solution to full clusters by moving the full cluster with probability p if the solution tells us to move a p-fraction of the cluster. This results in a very slight temporary relaxation of the capacity constraints. After clusters are moved, we continue with standard BLP iteration on individual nodes, which will remedy any violation of the capacity constraint in the next iteration.

3.3 Round-Robin Kernighan–Lin Swaps.

Our third idea again addresses the second limitation of BLP, by allowing moves that are not preferred moves, and by allowing swaps of nodes where one node obtains negative benefit. We achieve this by pairing up shards and applying the classical algorithm of Kernighan and Lin [13] to swap nodes between paired-up shards. Thus, nodes in each shard are ordered in descending order of the benefit of moving to the other shard in the pair, and we swap pairs in position $1, 2, 3, \ldots$ of their respective lists as long as the benefit from swapping a pair is positive. We note that this removes the need for an LP solver for this step, which was required in BLP to meet capacity constraints under multi-way moves.

However, when pairing up two arbitrary shards, there are often only a limited number of profitable swaps. To see significant movements from this approach, we perform a complete round-robin pairing of all shards. That is, every shard is paired up with every other shard during the disruptive step, for a total of $m(m-1)/2$ KL swaps where m is the number of shards. We note that this can be efficiently implemented by performing an initial scan over all edges to determine the local degree of each node, and the benefit of moving a node to any other shard with which it shares at least one edge. The benefits are then sorted, and the information is maintained as swaps are performed, allowing each round of KL swaps to be done without first scanning all edges or sorting by benefit again.

3.4 Preliminary Results.

We now take a first look at the benefits of the last two disruption methods on the performance of BLP, in Figure 4, using again the LiveJournal graph of 4M nodes with 50 partitions. We look at two initial assignments, SBM (in blue) and an assignment obtained by the sequential METIS package (in red), which we treat as a baseline that is not scalable to larger networks and distributed environments. We omit random initialization as it performed significantly worse than the others, placing it outside the range of this chart, and we also omit ProbDisrupt as the benefits are very limited as shown before. Our naming conventions are that the refinement method is first (BLP in this case), followed by the disruption method if any, followed by the initial assignment in parentheses. Thus, BLP-MC(SBM) means BLP with Moving Clusters (clustered relocation) as the disruptor and an initial assignment based on SBM.

As we see, METIS achieves a better initial assignment than SBM (red versus blue horizontal line). When we perform BLP steps on the initial assignment, SBM significantly narrows the gap to METIS, and in fact does better than METIS without BLP steps. Adding movement of clusters as disruptor (graphs with circles) gives a noticeable boost when starting from METIS (graphs with squares), but very little benefit when starting from SBM – most likely because clusters are already placed into the right shards by the initial greedy assignment of SBM. Finally, using KL swaps as a disruptor (graphs with triangles) gives a significant boost for both methods, and in particular for SBM as initial assignment. In the end, BLP-KL(METIS) and BLP-KL(SBM) achieve basically the same partitioning quality.

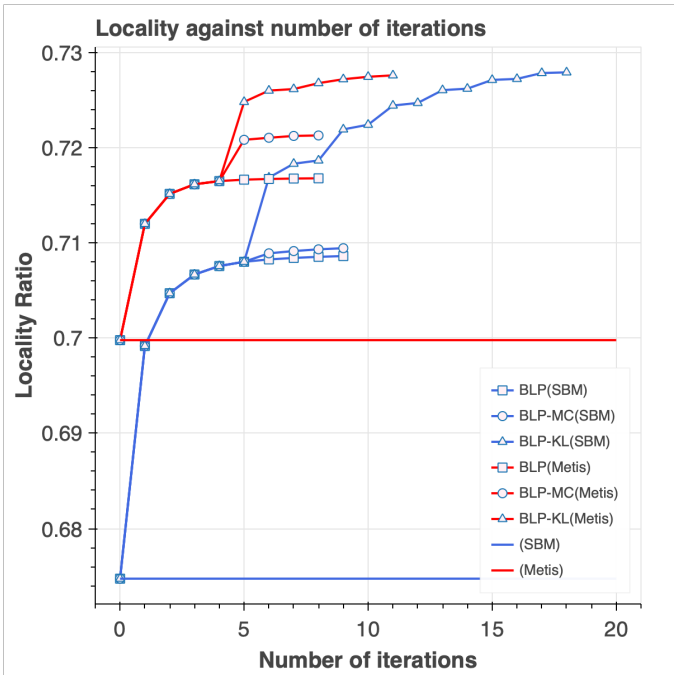


Fig. 4: Performance of Cluster and KL Disruptions

In Figure 5, we summarize the final locality achieved for random, SBM, and METIS initialization, and the contribution of the refinements due to BLP and disruption. We see that while random initialization benefits a lot from the refinements, it never catches up with SBM and METIS.

IV. Full Experimental Evaluation

We performed an evaluation over a number of data sets, including social networks and other graphs, with varying numbers of shards. The data sets are available from the Stanford SNAP collection [14], and are summarized in the following table. All graphs are undirected.

Graph Name	Type	Nodes	Edges
LiveJournal	Social	3,997,962	34,681,189
Orkut	Social	3,072,441	117,185,083
FB Athletes	Social	13,866	86,858
FB Companies	Social	14,113	52,310
roadNet CA	Road	1,965,206	5,533,214
Enron Email	Comm	36,692	367,662

All of our methods were implemented in C^{++} and run on a single node [1]. We did not optimize for running time, as our goal here is to explore the potential of the various optimizations to improve partitioning quality. An implementation in MapReduce or Pregel is also left for future work.

In Figure 6, we show results for the final partitioning locality on LiveJournal, as the number of shards is varied

[1] https://github.com/SteveDengZishi/Graph-Partitioning-Research

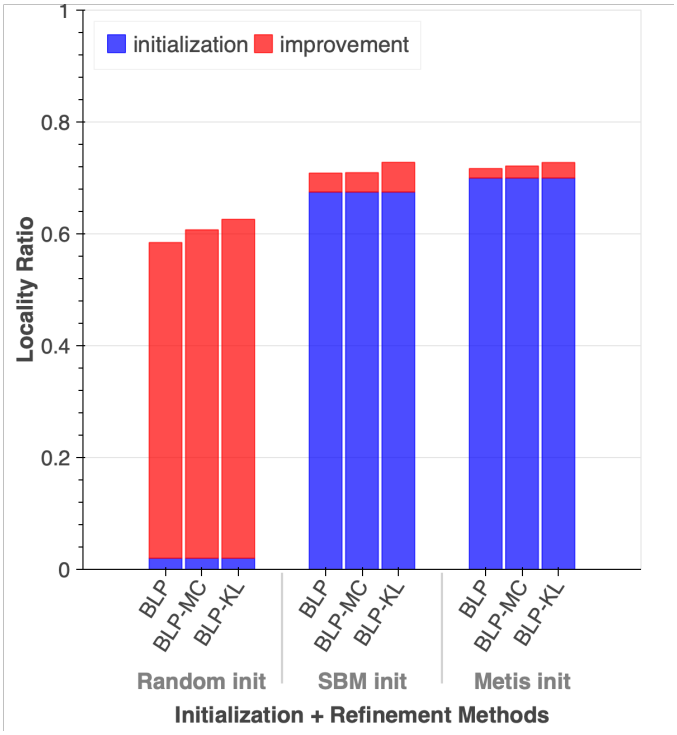


Fig. 5: Random / SBM / Metis Init Comparison

from 10 to 90. Of course, locality decreases with more shards. All methods with random initialization are at the bottom, as expected. We note that just METIS, with no refinement, does well for small numbers of shards, but worse as the number of shards increases. SBM benefits significantly from adding BLP refinement, and also from KL swaps as disruptors.

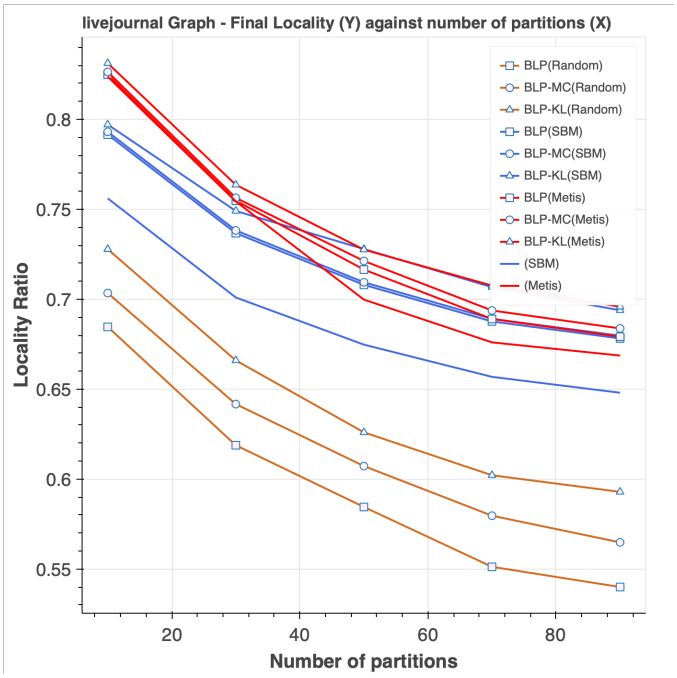


Fig. 6: Final locality comparison on LiveJournal

Next, in Figure 7, we show result for Orkut. We notice that METIS does badly on one data point (30 shards), but does well for the rest. Both SBM and METIS initial assignments benefit significantly from adding BLP refinement, and METIS gets additional significant benefits from KL Swaps.

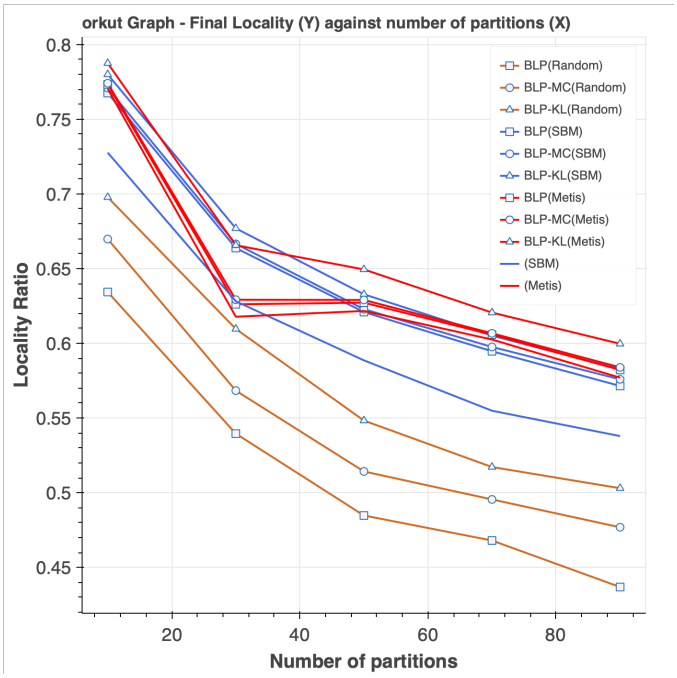


Fig. 7: Final locality comparison on Orkut

Next, in Figure 8 and Figure 9, We show results for two small data sets on Facebook athletes network and companies page network. We observe METIS does really well on small data sets and leaves very little room for refinement. SBM has a significant boost from refinement, but minor benefits from KL Swaps or cluster moves. The high-diameter companies page network appears to get more benefit from cluster movements than from KL Swaps.

Next, we look at email-Enron, an email graph, in Figure 10. We notice that METIS does very well, but refinement adds little value to it. SBM is clearly worse than METIS, but adding BLP to SBM gives significant improvements, while KL Swaps give minor extra benefits.

Finally, in Figure 11 we look at RoadNet data, a very different type of data set that is arguably much easier to partition. Here, METIS completely dominates, getting an almost perfect partitioning, with no potential for further improvement via refinement. SBM also sees very little improvement from any refinement steps. In summary, we see significant benefits from adding BLP refinement and, to a lesser degree, KL Swaps, to initial assignments based on SBM and METIS, for the social graphs and email network, but not for the high-diameter and almost planar RoadNet and the companies network data.

V. Concluding Remarks

In this paper, we have studied distributed algorithms for graph sharding, with a focus on optimizing the BLP and SBM

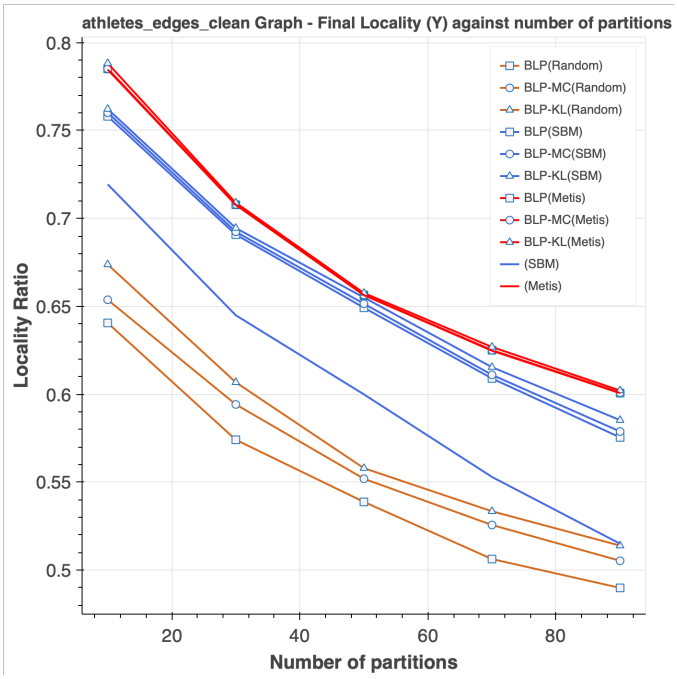


Fig. 8: Final locality comparison on FB-Athletes Pages

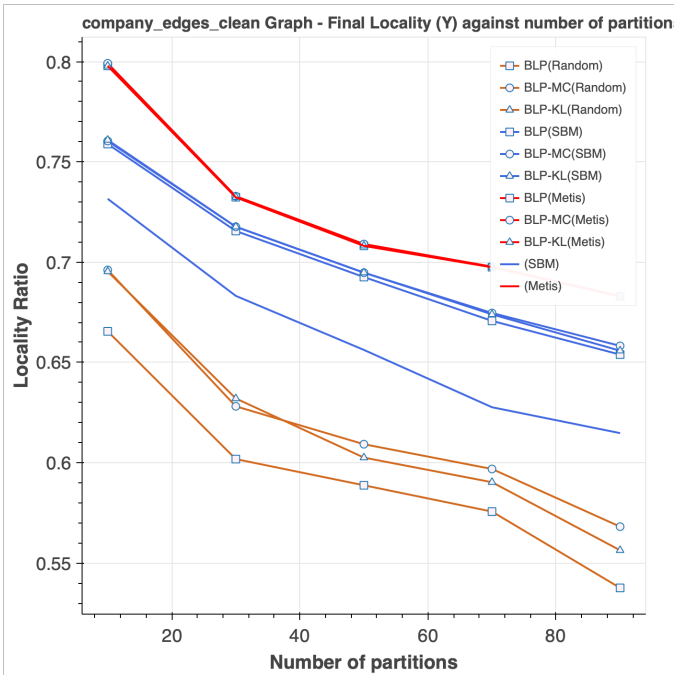


Fig. 9: Final locality comparison on FB-Company Pages

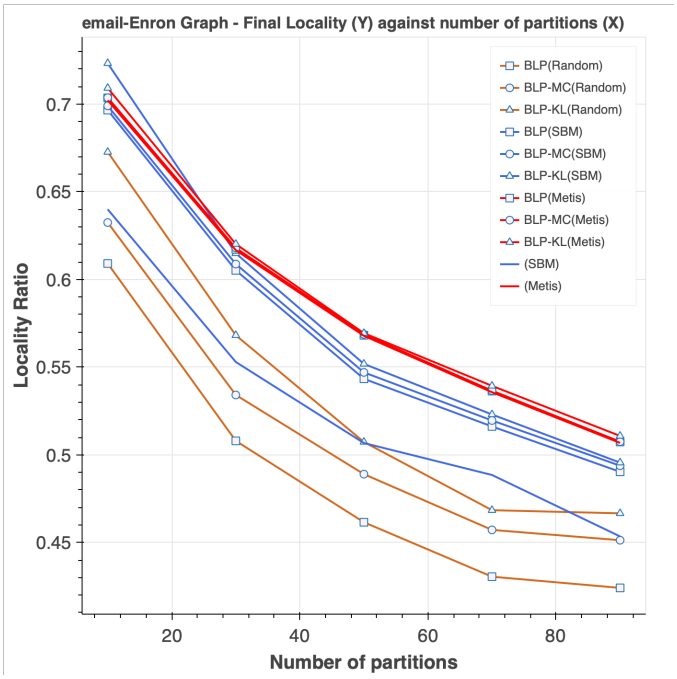


Fig. 10: Final locality comparison on email-Enron

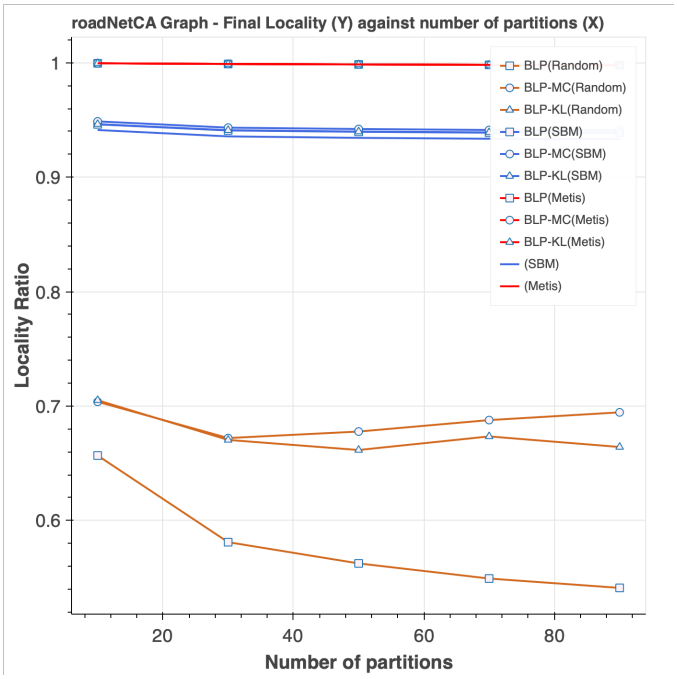


Fig. 11: Final locality comparison on RoadNet-CA

approaches proposed in [18] and [7]. Our results show that a combination of SBM and BLP has significant potential, and is often competitive with widely used sequential methods such as METIS, and that additional benefits can be obtained by adding disruptions based on cluster moves and KL (Kernighan-Lin) Swaps. The success of KL Swaps in particular suggests that the approach of constraining multi-way moves via an LP taken in the basic BLP approach adversely affects the resulting partition quality.

In future work, we plan to do evaluations on additional data sets, including web graphs and additional social networks. We also plan to implement our methods in a distributed environment, most likely MapReduce [6], to measure resulting running times on even larger graphs. Beyond these immediate goals, the problem of how to best perform graph sharding of large social networks in distributed environments remains an

interesting research challenge with many open questions and the potential for further significant gains.

Acknowledgements. This research was supported by NSF Grant IIS-1718680

REFERENCES

[1] K. Andreev and H. Raecke, *Balanced Graph Partitioning*, ACM Symp. on Parallel Algo. and Arch., 2004.

[2] A. Awadelkarim and J. Ugander, *Prioritized Restreaming Algorithms for Balanced Graph Partitioning*, ACM Conf. on Knowledge Discovery and Data Mining, 2020.

[3] M. Berkelaar, K. Eikland, and P. Notebaert, *The lpsolve package*, http://lpsolve.sourceforge.net.

[4] F. Bourse, M. Lelarge, M. Vojnovic, *Balanced Graph Edge Partition*, ACM Conf. on Knowledge Discovery and Data Mining, 2014.

[5] L. Dhulipala, I. Kabuljo, B. Karrer, G. Ottaviano, S. Pupyrev, A. Shalita, *Compressing Graphs and Indexes with Recursive Graph Bisection*, ACM Conf. on Knowledge Discovery and Data Mining, 2016.

[6] J. Dean and S. Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*, OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA (2004), pp. 137-150

[7] Q. Duong, S. Goel, J. Hofman, and S. Vassilvitskii, *Sharding social networks*, ACM Conf. on Web Search and Data Mining, 2013.

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman and Company, 1979.

[9] J. Hofman and C. Wiggins, *Bayesian Approach to Network Modularity*, Phys. Rev. Letters 100, 2008.

[10] P. Holland and S. Leinhardt, *An Exponential Family of Probability Distributions for Directed Graphs*, Journal of the Amer. Statistical Assoc. 76, 1981.

[11] J. Huang and D. J. Abadi, *LEOPARD: Lightweight Edge-Oriented Partitioning and Replication for Dynamic Graphs*, Symp. on Very Large Databases, 2016.

[12] G. Karypis and V. Kumar, *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*, Version 5.1.0, U. Minnesota, 2013.

[13] B. Kernighan and S. Lin, *An Effective Heuristic Procedure for Partitioning Graphs*, Bell Systems Technical Journal, 1970.

[14] J. Leskovec and A. Krevl, *SNAP Datasets*. http://snap.stanford.edu/data.

[15] G. Malewicz, M. Austern, A. Bik, J. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, *Pregel: A System for Large-Scale Graph Processing*, SIGMOD Conf., 2010.

[16] G.Slota, S. Rajamanickam, K. Devine, and K. Madduri, *Partitioning Trillion-Edge Graphs in Minutes*, IEEE Int. Parallel and Distributed Proc. Symp., 2017.

[17] C. Tsourakakis, C. Gkantsidis, B. Radunović, and M. Vojnovic, *FENNEL: Streaming Graph Partitioning for Massive Scale Graphs*, ACM Conf. on Web Search and Data Mining, 2014.

[18] J. Ugander and L. Backstrom, *Balanced Label Propagation for Partitioning Massive Graphs*, ACM Conf. on Web Search and Data Mining, 2013.