Graph Partitioning via Balanced Label Propagation

Johan Ugander
Cornell University
jhu5@cornell.edu

Lars Backstrom
Facebook
lars@fb.com

Summary

We introduce an efficient algorithm, **balanced label propagation**, for precisely partitioning massive graphs while greedily maximizing edge locality, the number of edges that are assigned to the same shard of a partition. By combining the computational efficiency of label propagation – where nodes are iteratively relabeled to the same `label' as the plurality of their graph neighbors – with the guarantees of **constrained optimization** – guiding the propagation by a **linear program** constraining the partition sizes – our algorithm makes it practically possible to partition graphs with billions of edges.

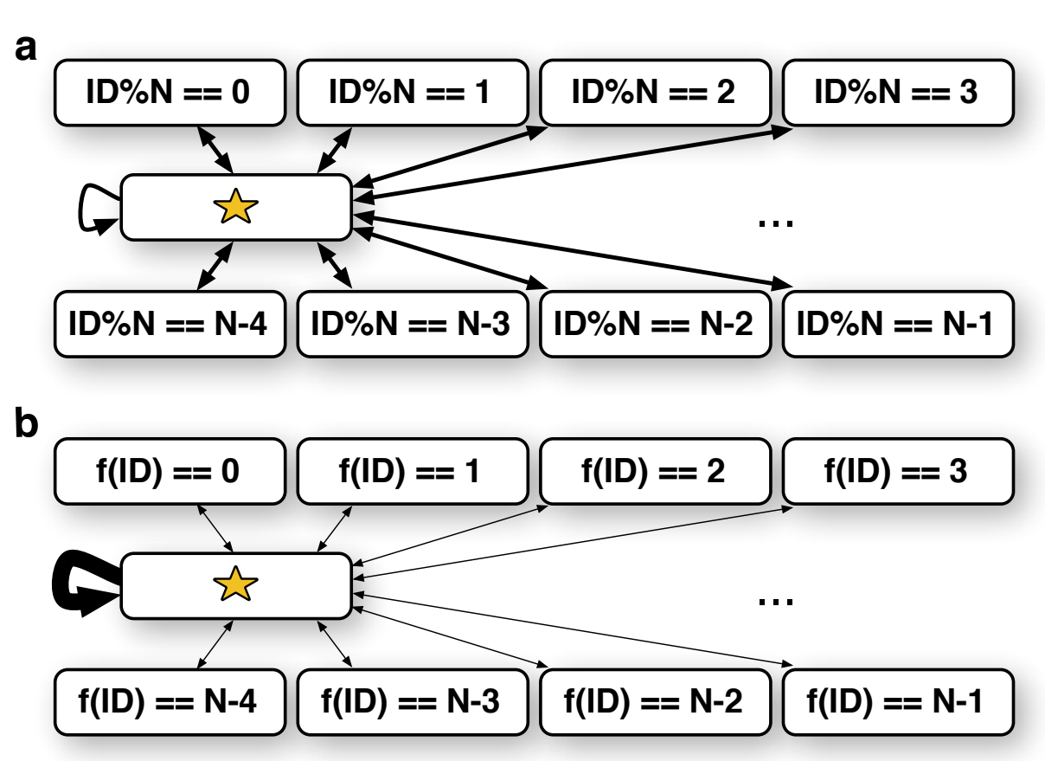
We evaluate our algorithm on the Facebook social graph, and also study its performance when partitioning Facebook's friend recommendation **`People You May Know**' service (PYMK), the distributed system responsible for the feature extraction and ranking of the friends-of-friends of all active Facebook users. We observed **average query times** and **average network traffic** levels that were **50.5%** and **37.1%** (respectively) when compared to the previous naïve random sharding.

Distributing large graphs

Many graph algorithms require traversing a graph. Distributing the graph means **distributing the edge list,** not just the node data. We want to greatly increase the likelihood that a node is located on the same partition as its graph neighbors. Basic vs. smart approach:

(a) Aggregating info about neighbors when the graph is partitioned by node ID modulus N.

(b) Aggregating when the graph is partitioned using a shardmap f.



PYMK friend recommendation: Want to rank Friends-of-Friends (FoFs) of a user u. First, query is sent to machine m_u. Then, for each neighbor v, a query must be issued to machine m_v, to retrieve the nodes two hops from u. The results of these queries are then aggregated to compute the features that are input to the machine learning phase, which outputs the final ranked list of FoFs.

Label propagation with constraints

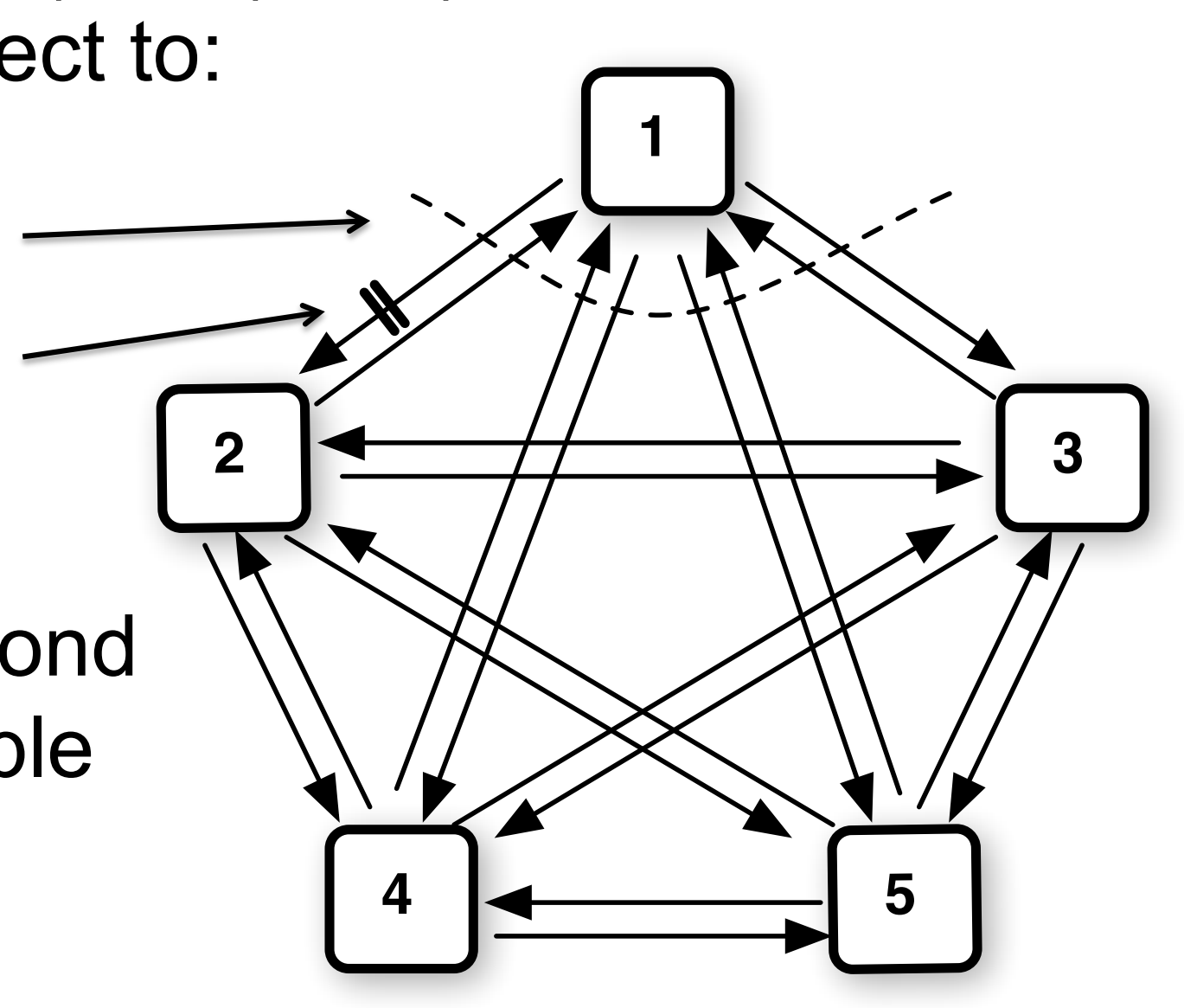
Label propagation: Each node of a graph is initialized with an individual label, and nodes iteratively adopt the label of the plurality of their neighbors until convergence.

Problems: No guarantees about (1) how many labels in final solution, (2) relative sizes of label sets, or (3) quality of partitioning.

We can fix (1) and (2) with constraints: Require n partitions: $V_1,\ldots,V_n$, and or each partition, constrain sizes: $S_i \leq |V_i| \leq T_i$. Given a feasible initial condition, we can propagate subject to:

$$S_1 - |V_1| \leq \sum_{i\neq 1}(x_{i1} - x_{1i}) \leq T_1 - |V_1|$$

$$0 \leq x_{12} \leq P_{12}$$



Where the first is a balance constraint (don't overflow partition 1), and the second is a population constraint (only P_{12} people want to move, can't move more).

Basic idea now is to move nodes ordered by how much their colocation increases. Solve the following convex problem:

Problem 1 (Constrained relocation) *Given a graph $G = (V, E)$ with the node set partitioned into n shards $V_1, \ldots, V_n$, and size constraints $S_i \leq |V_i| \leq T_i$, $\forall i$, the constrained relocation problem is to maximize:*

$$\max_X \textstyle\sum_{i,j} f_{ij}(x_{ij}) \qquad s.t. \tag{1}$$

$$\begin{array}{rcccll} S_i - |V_i| & \leq & \sum_{j\neq i}(x_{ij} - x_{ji}) & \leq & T_i - |V_i|, & \forall i \\ 0 & \leq & x_{ij} & \leq & P_{ij}, & \forall i, j. \end{array} \tag{2}$$

Here P_{ij} is the number of nodes that want to move from i to j, and $f_{ij}(x) = \sum_{k=1}^{x} u_{ij}(k)$ is the relocation utility function between shard i and j, and $u_{ij}(k)$ is the utility gain of the kth ordered person seeking to move from i to j.

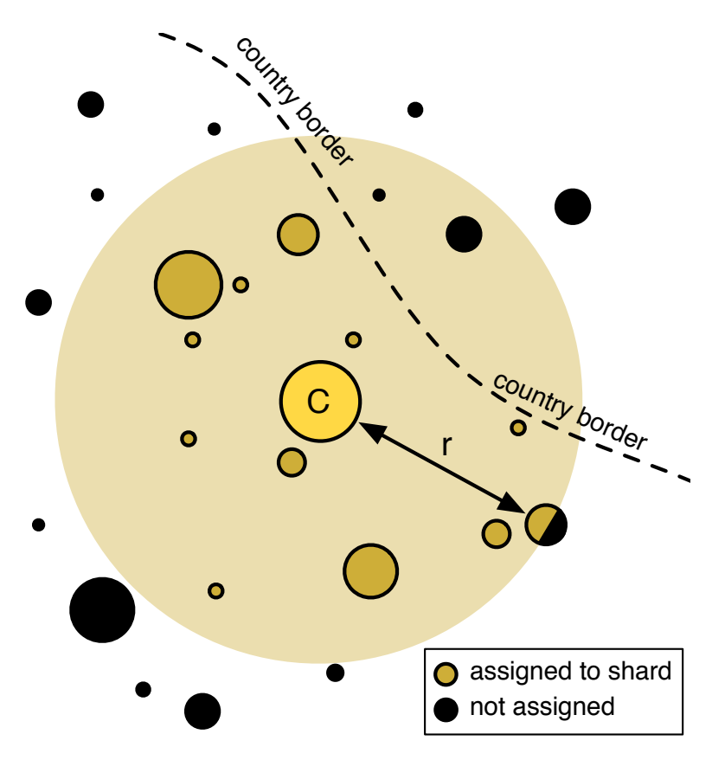
Because f_{ij} are piecewise-linear concave, reduces to an LP.

Iteration: (1) Determine where every node would prefer to move, and how much each node thinks it'll gain from relocation. **(2)** Sort the node gains for each partition pair and construct the Constrained Relocation LP. **(3)** Solve the LP, which determines how many nodes should be moved, in order, between each partition pair. **(4)** Move these nodes. This constitutes one iteration.

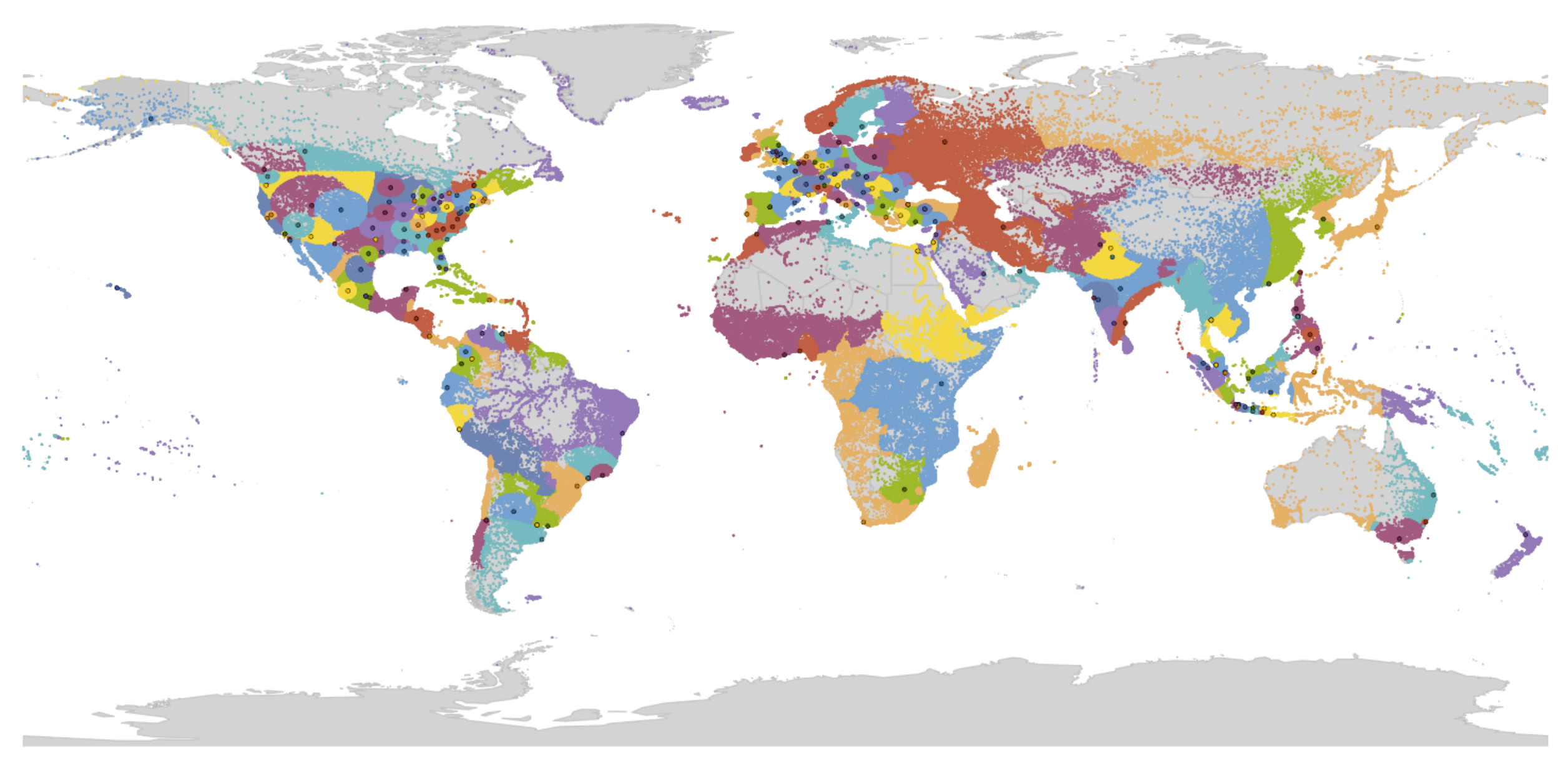
Geographic Initialization

For social networks with geographic metadata, we can initialize significantly better than random, in proportions that are feasible w.r.t. sizing constraints. Consider all users as mapped to cities with latitude and longitudes.

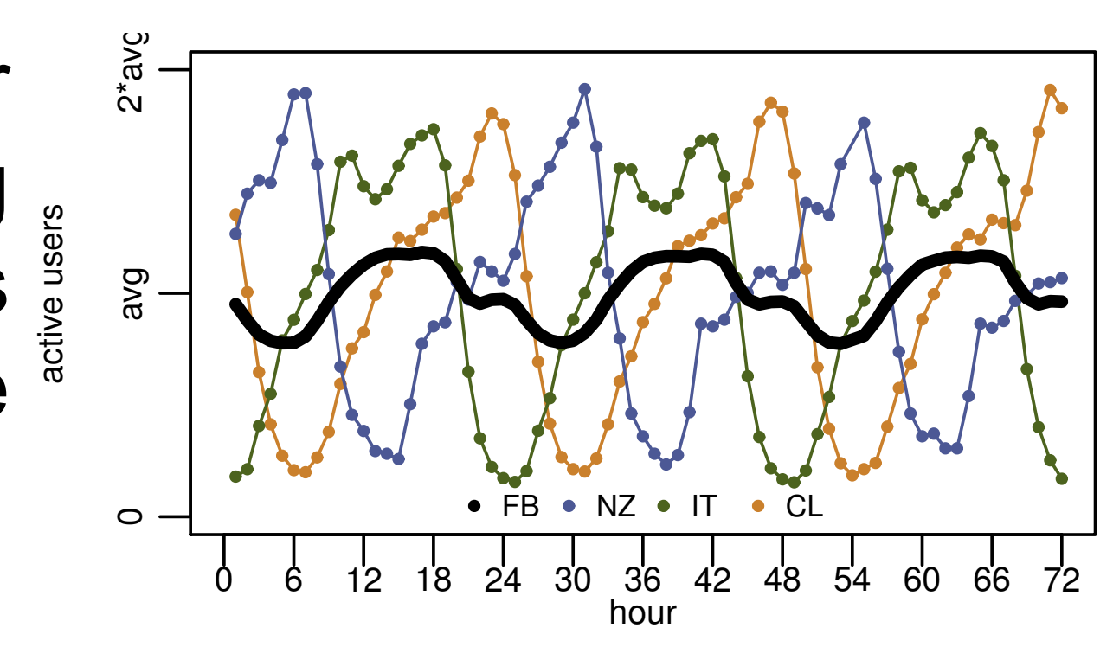
Greedy algorithm: One country at a time. Build partitions by `inflating a balloon' around most populous city not yet allocated. Repeat.



Example of 234 partitions of Facebook:

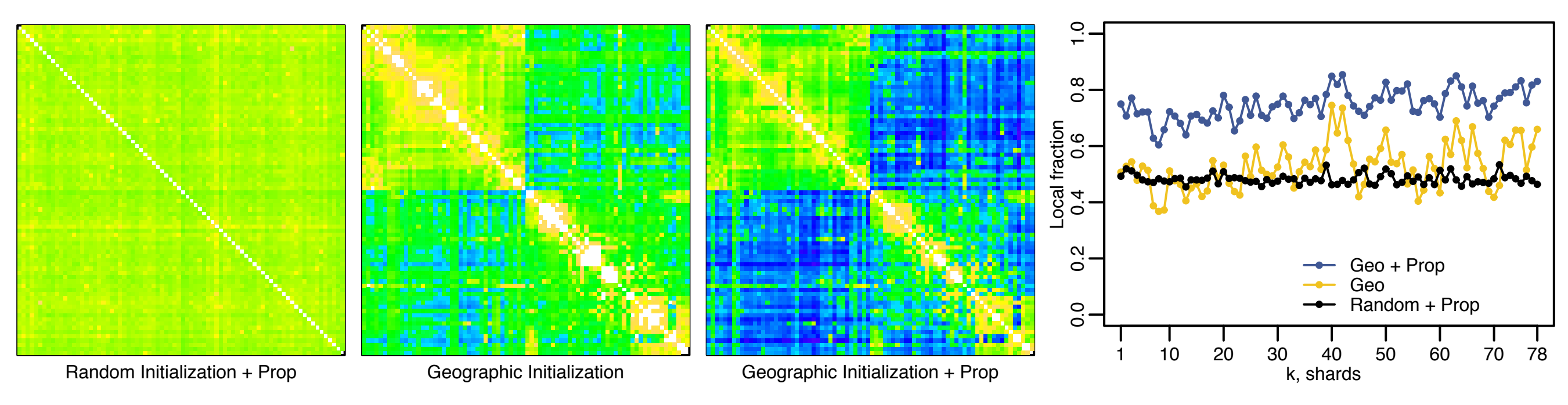


When using geography to partition for **realtime** computation, **oversharding** can be important: create more shards than there are machines and distribute cyclically to mitigate peak loads. Ex:



Partitioning performance

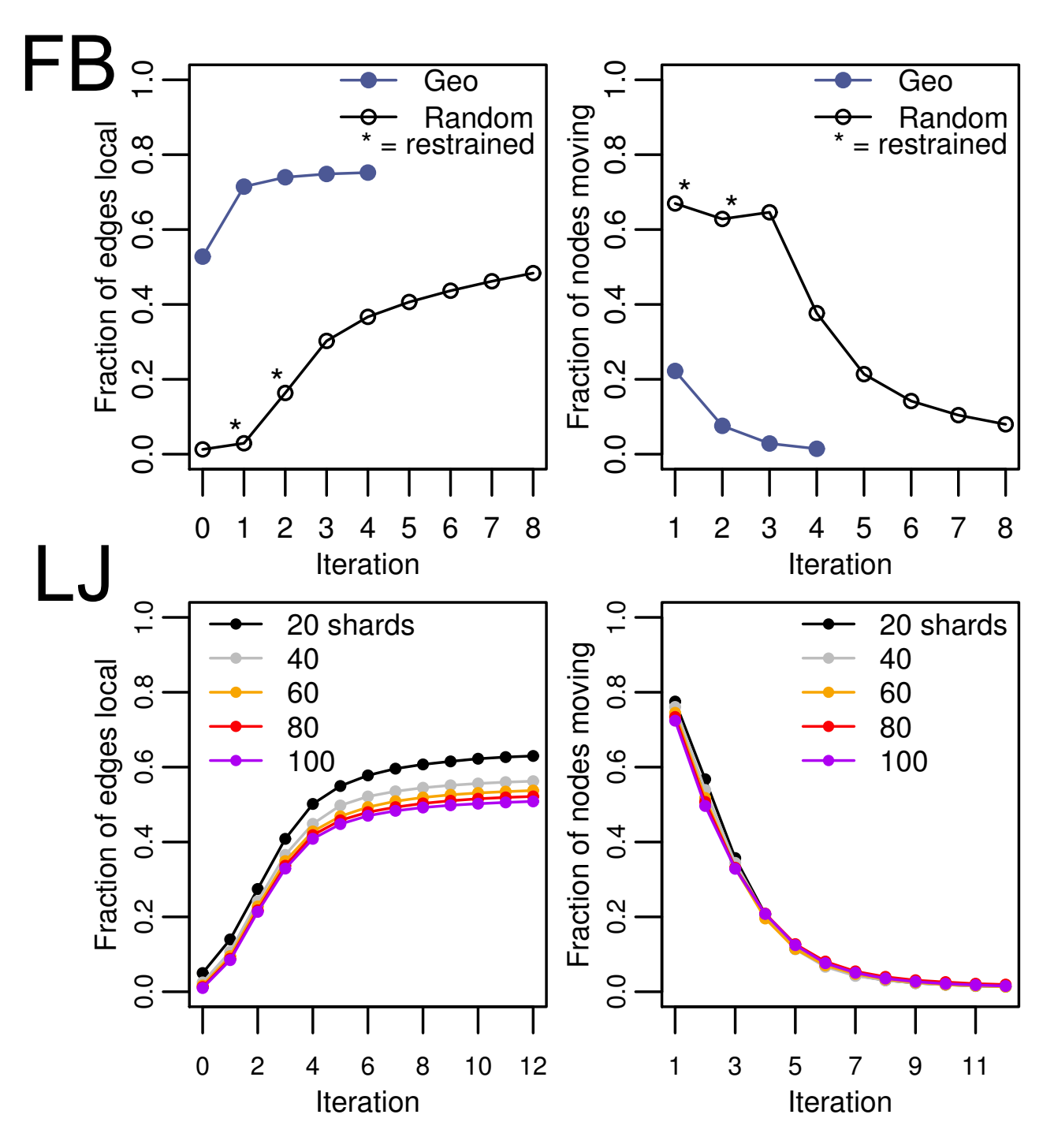
Evaluate performance by looking at **matrices of edges between 78 shards** under three different shardings: (1) BLP with random initialization, (2) geographic initialization only, and (3) 1-step BLP after geographic initialization. All matrices share the same logarithmic color scale. Also, fraction of local edges for each shard.



Iterative improvement shown to the right, examining both the FB graph and a public LiveJournal graph.

Local fraction of edges and **fractions of nodes moving** are plotted as a function of iteration.

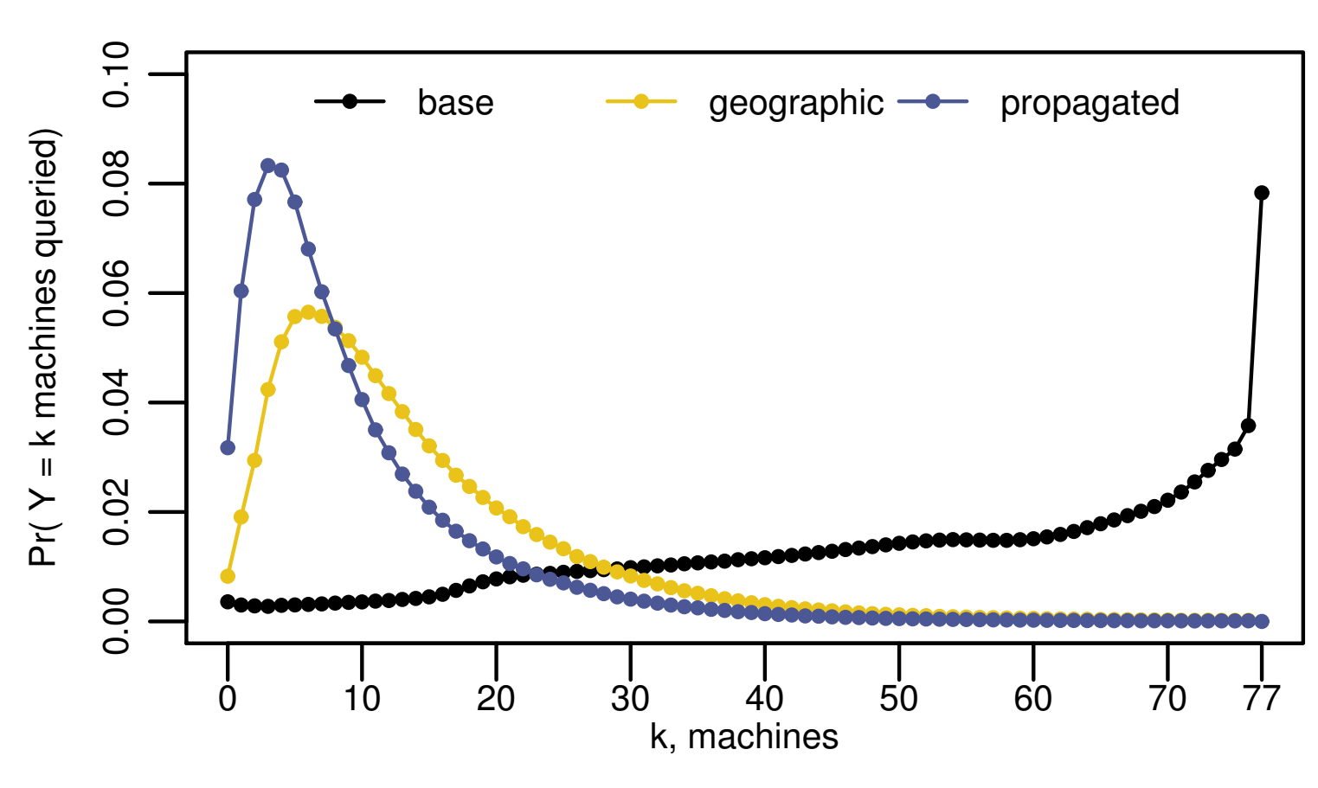
Note that (1) only a **few iterations are required** and (2) for LJ, quality is **nearly independent of number of partitions sought**. This is perhaps surprising, yet consistent with the BLP algorithm being highly local.



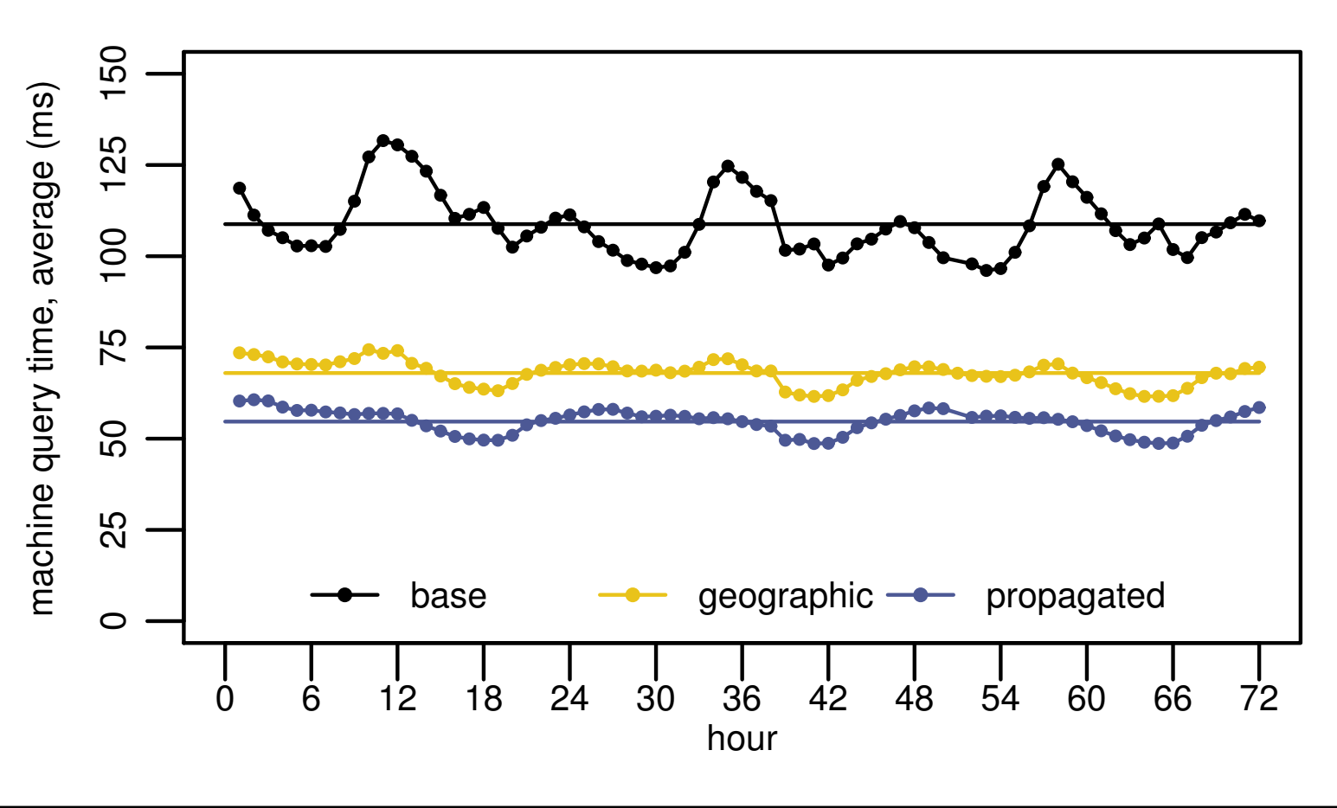
Realtime deployment (PYMK)

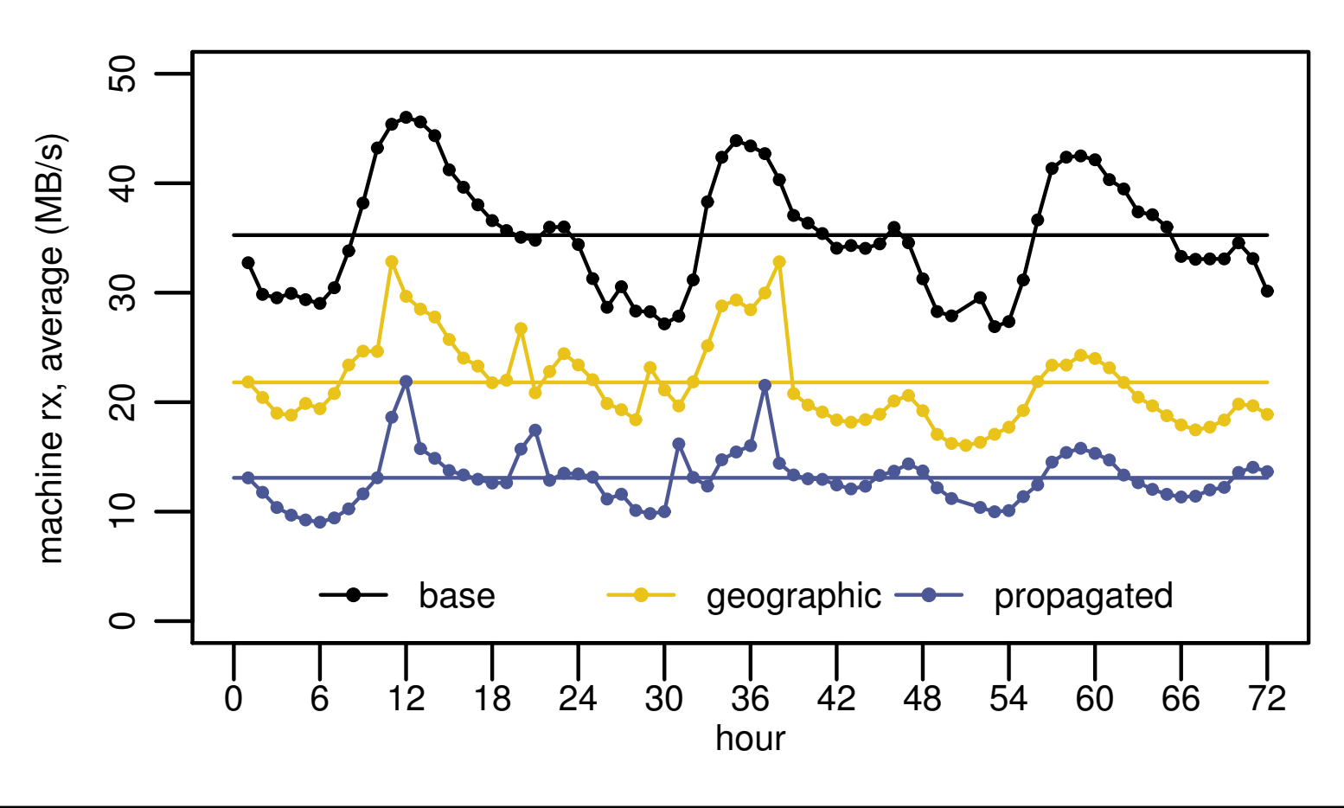
Ran three systems in parallel for 72 hours, equally load-balanced. Each system had 78 machines.

The **median number of machines** queried for baseline, geographic init, and geographic + propagated systems were 59 / 12 / **9 machines**.



Next, examined **query time** and **network traffic.** Geographic initialization again performed well, but BLP propagation helped significantly. In total, query time was reduced to **50.5%** and network traffic was reduced to **37.1%** of baseline naïve configuration.





Paper: http://bit.ly/nips12-blp
Poster: http://bit.ly/nips12-blp2