PuLP: Scalable Multi-Objective Multi-Constraint Partitioning for Small-World Networks

George M. Slota[1,2] Kamesh Madduri[2]
Sivasankaran Rajamanickam[1]

[1]Sandia National Laboratories, [2]The Pennsylvania State University
gslota@psu.edu, madduri@cse.psu.edu, srajama@sandia.gov

BigData14 28 Oct 2014

Highlights

- We present PuLP, a multi-constraint multi-objective partitioner designed for small-world graphs
- Shared-memory parallelism
- PuLP demonstrates an average speedup of 14.5$\times$ relative to state-of-the-art partitioners
- PuLP requires 8-39$\times$ less memory than state-of-the-art partitioners
- PuLP produces partitions with comparable or better quality than state-of-the-art partitioners for small-world graphs

Overview

- PuLP: Partitioning Using Label Propagation
  - Overview
    - Graph partitioning formulation
    - Label propagation
    - Using label propagation for partitioning
  - PuLP Algorithm
    - Degree-weighted label prop
    - Label propagation for balancing constraints and minimizing objectives
    - Label propagation for iterative refinement
  - Results
    - Performance comparisons with other partitioners
    - Partitioning quality with different objectives

- **Graph Partitioning**: Given a graph $G(V, E)$ and p processes or tasks, assign each task a p-way disjoint subset of vertices and their incident edges from G
  - Balance constraints – (weighted) vertices per part, (weighted) edges per part
  - Quality metrics – edge cut, communication volume, maximal per-part edge cut
- We consider:
  - Balancing edges **and** vertices per part
  - Minimizing edge cut (EC) **and** maximal per-part edge cut (EC_{max})

Overview

Partitioning - Objectives and Constraints

- Lots of graph algorithms follow a certain iterative model
  - BFS, SSSP, FASCIA subgraph counting (Slota and Madduri 2014)
  - computation, synchronization, communication, synchronization, computation, etc.
- Computational load: proportional to vertices and edges per-part
- Communication load: proportional to total edge cut and max per-part cut
- We want to minimize the maximal time among tasks for each comp/comm stage

Overview

Partitioning - Balance Constraints

- Balance vertices and edges:

$$(1-\epsilon_l)\,\frac{|V|}{p} \quad \le |V(\pi_i)| \quad \le (1+\epsilon_u)\,\frac{|V|}{p} \tag{1}$$

$$|E(\pi_i)| \quad \le (1+\eta_u)\,\frac{|E|}{p} \tag{2}$$

- ϵ_l and ϵ_u: lower and upper vertex imbalance ratios
- η_u: upper edge imbalance ratio
- $V(\pi_i)$: set of vertices in part π_i
- $E(\pi_i)$: set of edges with both endpoints in part π_i

- Given a partition Π, the set of *cut edges* $(C(G,\Pi))$ and cut edge per partition $(C(G,\pi_k))$ are

$$C(G,\Pi) = \{\{(u,v) \in E\} \mid \Pi(u) \neq \Pi(v)\} \quad (3)$$
$$C(G,\pi_k) = \{\{(u,v) \in C(G,\Pi)\} \mid (u \in \pi_k \vee v \in \pi_k)\} \quad (4)$$

- Our partitioning problem is then to minimize total edge cut EC and max per-part edge cut EC_{max}:

$$EC(G,\Pi) = |C(G,\Pi)| \quad (5)$$
$$EC_{max}(G,\Pi) = \max_k |C(G,\pi_k)| \quad (6)$$

- (Par)METIS (Karypis et al.), PT-SCOTCH (Pellegrini et al.), Chaco (Hendrickson et al.), etc.
- Multilevel methods:
  - *Coarsen* the input graph in several iterative steps
  - At coarsest level, partition graph via local methods following balance constraints and quality objectives
  - Iteratively *uncoarsen* graph, refine partitioning
- **Problem 1**: Designed for traditional HPC scientific problems (e.g. meshes) – limited balance constraints and quality objectives
- **Problem 2**: Multilevel approach – high memory requirements, can run slowly and lack scalability

- **Label propagation**: randomly initialize a graph with some p labels, iteratively assign to each vertex the maximal per-label count over all neighbors to generate clusters (Raghavan et al. 2007)
  - Clustering algorithm - dense clusters hold same label
  - Fast - each iteration in $O(n+m)$, usually fixed iteration count (doesn't necessarily converge)
  - Naïvely parallel - only per-vertex label updates
  - *Observation*: Possible applications for large-scale small-world graph partitioning

Overview

Partitioning - "Big Data" Approaches

- Methods designed for small-world graphs (e.g. social networks and web graphs)
- Exploit label propagation/clustering for partitioning:
  - Multilevel methods - use label propagation to coarsen graph (Wang et al. 2014, Meyerhenke et al. 2014)
  - Single level methods - use label propagation to directly create partitioning (Ugander and Backstrom 2013, Vaquero et al. 2013)
- **Problem 1**: Multilevel methods still can lack scalability, might also require running traditional partitioner at coarsest level
- **Problem 2**: Single level methods can produce sub-optimal partition quality

PuLP : **P**artitioning **U**sing **L**abel **P**ropagation

- Utilize label propagation for:
  - Vertex balanced partitions, minimize edge cut (PuLP)
  - Vertex and edge balanced partitions, minimize edge cut (PuLP-M)
  - Vertex and edge balanced partitions, minimize edge cut and maximal per-part edge cut (PuLP-MM)
  - Any combination of the above - multi objective, multi constraint

- PuLP-MM Algorithm
  - Constraint 1: balance vertices, Constraint 2: balance edges
  - Objective 1: minimize edge cut, Objective 2: minimize per-partition edge cut
  - Pseudocode gives default iteration counts

Initialize p random partitions
Execute 3 iterations degree-weighted label propagation (LP)
for $k_1 = 1$ iterations **do**
    for $k_2 = 3$ iterations **do**
        Balance partitions with 5 LP iterations to satisfy constraint 1
        Refine partitions with 10 FM iterations to minimize objective 1
    for $k_3 = 3$ iterations **do**
        Balance partitions with 2 LP iterations to satisfy constraint 2
          and minimize objective 2 with 5 FM iterations
        Refine partitions with 10 FM iterations to minimize objective 1

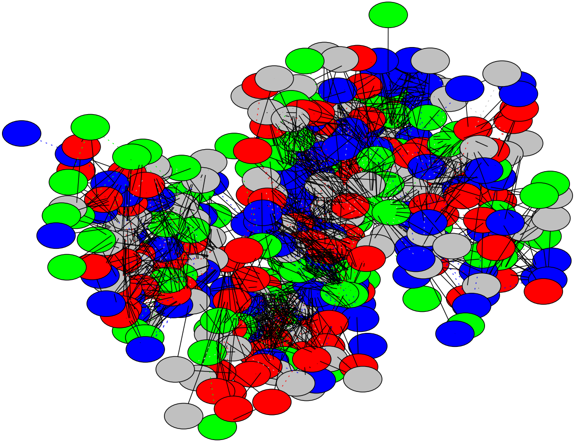
Algorithms

Primary Algorithm Overview

Initialize p random partitions
Execute degree-weighted label propagation (LP)
for k_1 iterations **do**
    for k_2 iterations **do**
        Balance partitions with LP to satisfy vertex constraint
        Refine partitions with FM to minimize edge cut
    for k_3 iterations **do**
        Balance partitions with LP to satisfy edge constraint and minimize max per-part cut
        Refine partitions with FM to minimize edge cut

Algorithms
Primary Algorithm Overview

Randomly initialize p partitions ($p = 4$)

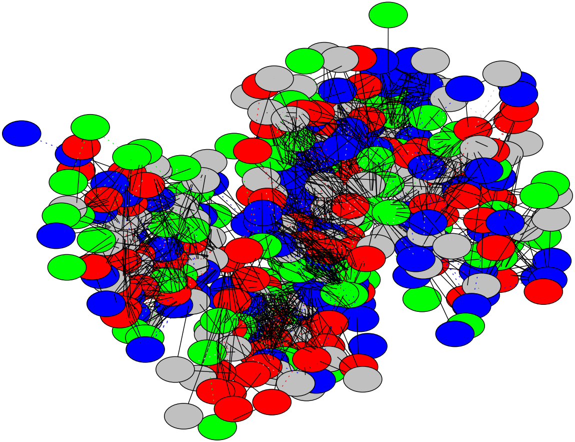


Network shown is the Infectious network dataset from KONECT (http://konect.uni-koblenz.de/)

- After random initialization, we then perform label propagation to create partitions
- **Initial Observations**:
  - Partitions are unbalanced, for high p, some partitions end up empty
  - Edge cut is good, but can be better
- **PuLP Solutions**:
  - Impose loose balance constraints, explicitly refine later
  - Degree weightings - cluster around high degree vertices, let low degree vertices form boundary between partitions

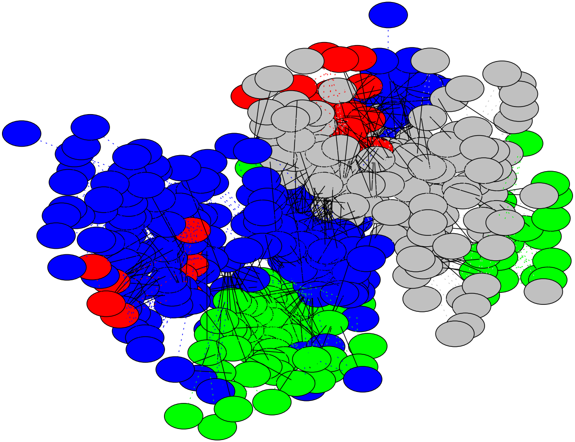
Initialize p random partitions
Execute degree-weighted label propagation (LP)
for k_1 iterations **do**
  for k_2 iterations **do**
    Balance partitions with LP to satisfy vertex constraint
    Refine partitions with FM to minimize edge cut
  for k_3 iterations **do**
    Balance partitions with LP to satisfy edge constraint and minimize max per-part cut
    Refine partitions with FM to minimize edge cut

Part assignment after random initialization.



Network shown is the Infectious network dataset from KONECT (http://konect.uni-koblenz.de/)

Part assignment after degree-weighted label propagation.



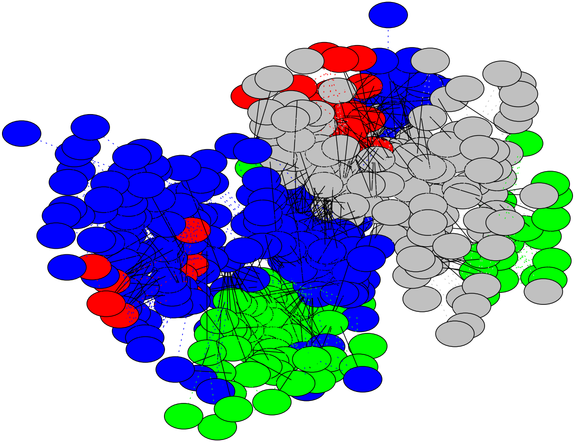
Network shown is the Infectious network dataset from KONECT (http://konect.uni-koblenz.de/)

Algorithms
Primary Algorithm Overview

- After label propagation, we balance vertices among partitions and minimize edge cut (baseline PuLP ends here)
- **Observations**:
  - Partitions are still unbalanced in terms of edges
  - Edge cut is good, max per-part cut isn't necessarily
- **PuLP-M and PuLP-MM Solutions**:
  - Maintain vertex balance while explicitly balancing edges
  - Alternate between minimizing total edge cut and max per-part cut (for PuLP-MM, PuLP-M only minimizes total edge cut)

Initialize p random partitions
Execute degree-weighted label propagation (LP)
for k_1 iterations **do**
  for k_2 iterations **do**
    Balance partitions with LP to satisfy vertex constraint
    Refine partitions with FM to minimize edge cut
  for k_3 iterations **do**
    Balance partitions with LP to satisfy edge constraint and minimize max per-part cut
    Refine partitions with FM to minimize edge cut

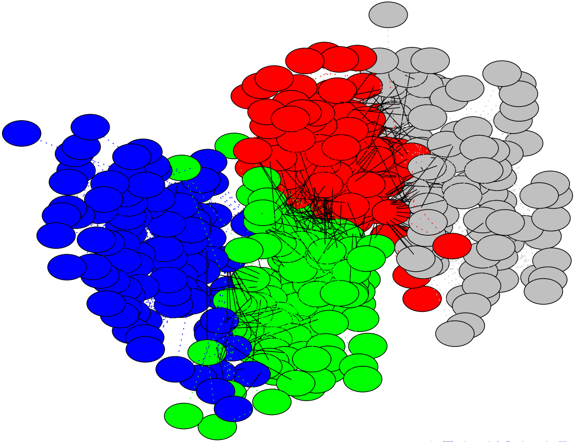
Part assignment after degree-weighted label propagation.



Network shown is the Infectious network dataset from KONECT (http://konect.uni-koblenz.de/)

Algorithms
Primary Algorithm Overview

Part assignment after balancing for vertices and minimizing edge cut.

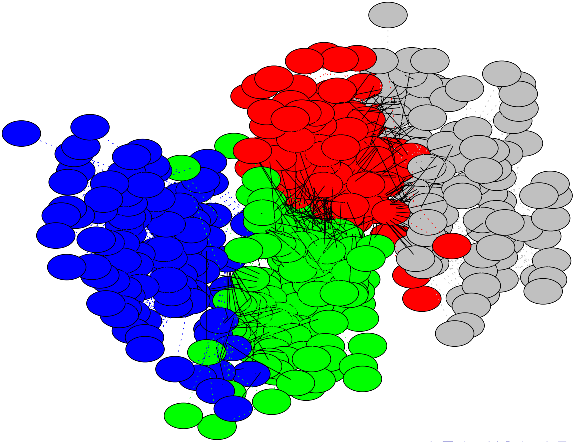


Network shown is the Infectious network dataset from KONECT (http://konect.uni-koblenz.de/)

Initialize p random partitions
Execute degree-weighted label propagation (LP)
for k_1 iterations **do**
  for k_2 iterations **do**
    Balance partitions with LP to satisfy vertex constraint
    Refine partitions with FM to minimize edge cut
  for k_3 iterations **do**
    Balance partitions with LP to satisfy edge constraint and minimize max per-part cut
    Refine partitions with FM to minimize edge cut

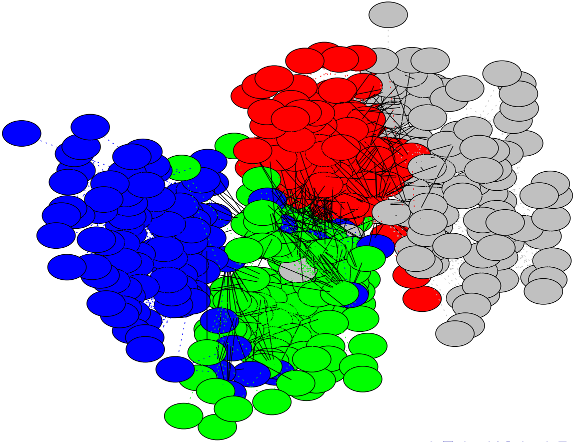
Algorithms
Primary Algorithm Overview

Part assignment after balancing for vertices and minimizing edge cut.



Network shown is the Infectious network dataset from KONECT (http://konect.uni-koblenz.de/)

Part assignment after balancing for edges and minimizing total edge cut and max per-part edge cut



Network shown is the Infectious network dataset from KONECT (http://konect.uni-koblenz.de/)

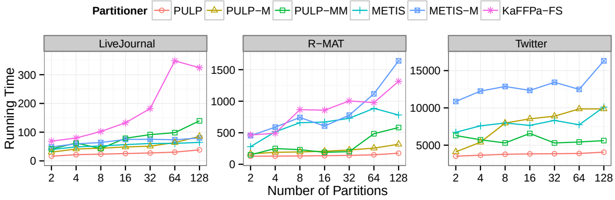
Results
Test Environment and Graphs

- Test system: *Compton*
  - Intel Xeon E5-2670 (Sandy Bridge), dual-socket, 16 cores, 64 GB memory.
- Test graphs:
  - LAW graphs from UF Sparse Matrix, SNAP, MPI, Koblenz
  - Real (one R-MAT), small-world, 60 K–70 M vertices, 275 K–2 B edges
- Test Algorithms:
  - **METIS** - single constraint single objective
  - **METIS-M** - multi constraint single objective
  - **ParMETIS** - METIS-M running in parallel
  - **KaFFPa** - single constraint single objective
  - **PuLP** - single constraint single objective
  - **PuLP-M** - multi constraint single objective
  - **PuLP-MM** - multi constraint multi objective
- Metrics: 2–128 partitions, serial and parallel running times, memory utilization, edge cut, max per-partition edge cut

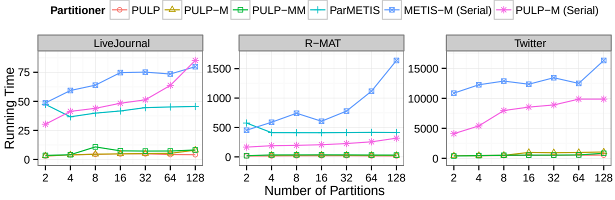
Results

Running Times - Serial (top), Parallel (bottom)

- In serial, PULP-MM runs 1.7× faster (geometric mean) than next fastest



- In parallel, PULP-MM runs 14.5× faster (geometric mean) than next fastest (ParMETIS times are fastest of 1 to 256 cores)



Results

Memory utilization for 128 partitions

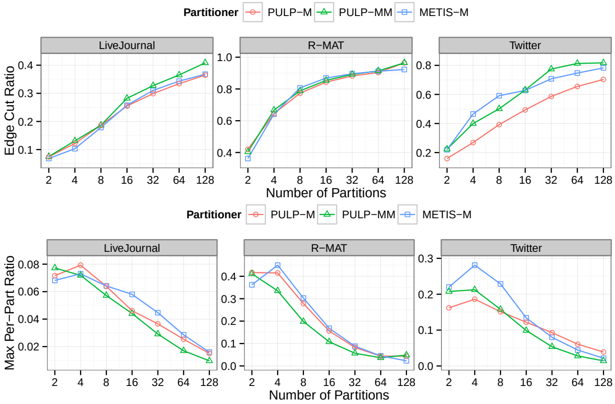
- PuLP utilizes minimal memory, $O(n)$, 8-39× less than other partitioners
- Savings are mostly from avoiding a multilevel approach

Network	Memory Utilization				**Improv.**
	METIS-M	KaFFPa	PuLP-MM	Graph Size	
LiveJournal	7.2 GB	5.0 GB	0.44 GB	0.33 GB	21×
Orkut	21 GB	13 GB	0.99 GB	0.88 GB	23×
R-MAT	42 GB	-	1.2 GB	1.02 GB	35×
DBpedia	46 GB	-	2.8 GB	1.6 GB	28×
WikiLinks	103 GB	42 GB	5.3 GB	4.1 GB	25×
sk-2005	121 GB	-	16 GB	13.7 GB	8×
Twitter	487 GB	-	14 GB	12.2 GB	39×

Results

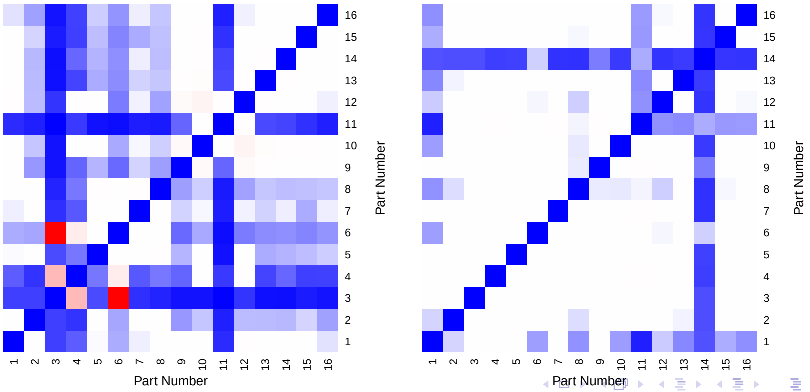
Performance - Edge Cut and Edge Cut Max

- PuLP-M produces better edge cut than METIS-M over most graphs
- PuLP-MM produces better max edge cut than METIS-M over most graphs



Results
Balanced communication

- uk-2005 graph from LAW, METIS-M (left) vs. PuLP-MM (right)
- Blue: low comm; White: avg comm; Red: High comm
- PuLP reduces max inter-part communication requirements and balances total communication load through all tasks



Future Work

- Explore techniques for avoiding local minima, such as simulated annealing, etc.
- Further parallelization in distributed environment for massive-scale graphs
- Demonstrate performance of PuLP partitions with graph analytics
- Explore tradeoff and interactions in various parameters and iteration counts

Conclusions

- We presented PULP, a multi-constraint multi-objective partitioner designed for small-world graphs
- Shared-memory parallelism
- PULP demonstrates an average speedup of 14.5$\times$ relative to state-of-the-art partitioners
- PULP requires 8-39$\times$ less memory than state-of-the-art partitioners
- PULP produces partitions with comparable or better quality than METIS/ParMETIS for small-world graphs

Conclusions

- We presented PuLP, a multi-constraint multi-objective partitioner designed for small-world graphs
- Shared-memory parallelism
- PuLP demonstrates an average speedup of 14.5$\times$ relative to state-of-the-art partitioners
- PuLP requires 8-39$\times$ less memory than state-of-the-art partitioners
- PuLP produces partitions with comparable or better quality than METIS/ParMETIS for small-world graphs
- **Questions?**

Acknowledgments

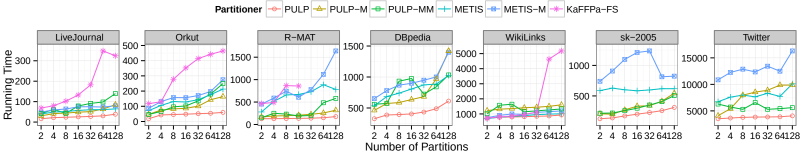
- DOE Office of Science through the FASTMath SciDAC Institute
  - Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.
- NSF grant ACI-1253881, OCI-0821527
- Used NERSC hardware for generating partitionings - supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231

- Backup slides

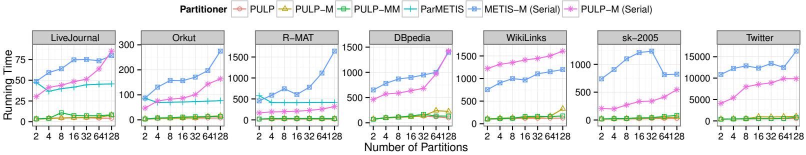
Results

Running Times - Serial (top), Parallel (bottom)

- PuLP faster than others over most tests in serial
- In parallel, PuLP always faster than other



- In parallel, PuLP runs 14.5× faster (geometric mean)



Results

Memory utilization for 128 partitions

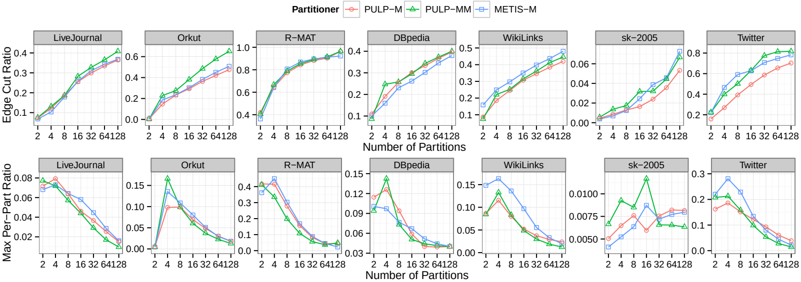
- PuLP utilizes minimal memory - $O(n)$
- Savings are mostly from avoiding a multilevel approach

Network	Memory Utilization			Graph Size	**Improv.**
	METIS-M	KaFFPa	PuLP-MM		
LiveJournal	7.2 GB	5.0 GB	0.44 GB	0.33 GB	21×
Orkut	21 GB	13 GB	0.99 GB	0.88 GB	23×
R-MAT	42 GB	-	1.2 GB	1.02 GB	35×
DBpedia	46 GB	-	2.8 GB	1.6 GB	28×
WikiLinks	103 GB	42 GB	5.3 GB	4.1 GB	25×
sk-2005	121 GB	-	16 GB	13.7 GB	8×
Twitter	487 GB	-	14 GB	12.2 GB	39×

Results

Performance - Edge Cut and Edge Cut Max

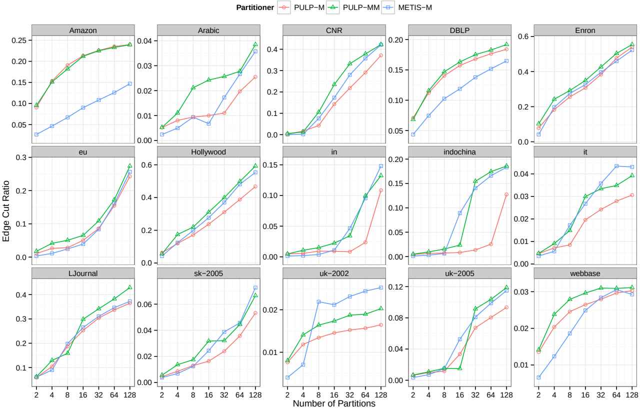
- PuLP-M produces better edge cut than METIS-M over most graphs
- PuLP-MM produces better max edge cut than METIS-M over most graphs
- Taken together, these demonstrate the tradeoff for multi objective



Results

Performance - Edge Cut and Edge Cut Max

- PuLP-M produces better edge cut than METIS-M over most graphs
- PuLP-MM produces better max edge cut than METIS-M over most graphs
- Taken together, these demonstrate the tradeoff for multi objective
- Across all Lab for Web Algorithmics graphs



Results

Performance - Edge Cut and Edge Cut Max

- PuLP-M produces better edge cut than METIS-M over most graphs
- PuLP-MM produces better max edge cut than METIS-M over most graphs
- Taken together, these demonstrate the tradeoff for multi objective
- Across all Lab for Web Algorithmics graphs