Scalable Algorithms for Graph Matching and Edge Cover Computations

Arif Khan[1], Alex Pothen[1], Mostofa Patwary[2] & Pradeep Dubey[2]

[1]Computer Science, Purdue
[2]Intel Parallel Computing Labs

March 1, 2017

PURDUE
UNIVERSITY™

Overview: *b*-Matching

The Matching problem in graphs is well-studied, but this is not true of *b*-Matching:

- We discuss approximation algorithms that have high concurrency.
- We design the most efficient $1/2$-approximation algorithm, *b*-Suitor.
- We parallelize *b*-Suitor for shared memory and distributed memory machines.

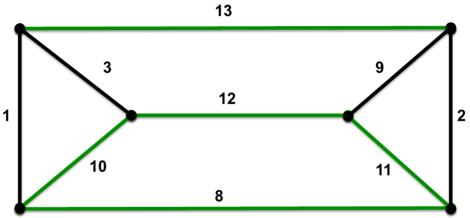
Overview: b-Edge Cover

Other than the Greedy algorithm, there is little work on approximation algorithms for b-Edge Cover.

- We design two new approximation algorithms: 3/2-approximate Locally Subdominant Edge (LSE) and 2-approximate Static-LSE (S-LSE).
- **We establish the relationship between b-Edge Cover and b-Matching in the context of approximation algorithms.**
- Using b-Matching, we design the most efficient algorithm MCE, a 2-approximation algorithm.

b-Matching

- A b-Matching is a set of edges M such that *at most* $b(v)$ edges in M are incident on each vertex $v \in V$.
- The weight of a b-Matching is the sum of the weights of the matched edges.
- Max. weight b-Matching : a matching with maximum weight.
- Standard Matching is a special case of b-Matching with $b = 1$.



Applications of b-Matchings

- Mesh refinement. [Hannemann et al, JEA, 1999]
- Spectral clustering. [Jebara et al, ECML, 2006]
- Semi supervised learning. [Jebara et al, ICML, 2009]
- Overlay network. [Georgiadis et al, IPDPS, 2010]
- Data Privacy. [Choromanski et al, NIPS, 2013]
- b-Edge Cover. [Khan et al, CSC, 2016]

Algorithms for b-MATCHING

$G = (V, E, w, b)$, $n = |V|$, $m = |E|$,
$\beta = \max_{v \in V} b(v)$, and $B = \sum_{v \in V} b(v)$.

- **Exact Algorithms**
  - $O(Bm \log n)$ [Gabow, 1983]
  - Finds the solution of maximum weight b-MATCHING.
  - Hard to implement, not amenable to parallelize and not suitable for solving large problems.

Exact Algorithms

Graph	Vertices	Edges	Exact time	Exact weight	1/2-Approx. time	1/2-Approx. % opt. wt./
IG5-16	37K	588K	10 s	1.4 e4	1.6e-2 s	98.7 %
Image-interp	360K	712K	1.2 s	1.5 e8	3.5e-2 s	96.5 %
LargeRegFile	2.9M	4.9M	6.9 s	9.7 e8	0.2 s	98.9 %
Rucci1	2.1M	7.8M	4 h 36 m	1.6 e8	1.3 s	99.7 %
GL7d16	1.5M	14.5M	9 h 50 m	5.8 e8	1.3 s	94.5 %
GL7d20	3.3M	29.9M	$>$ 100 h	NA	4.8 s	NA
GL7d18	3.5M	35.6M	$>$ 100 h	NA	5.5 s	NA
GL7d19	3.9M	37.3M	$>$ 100 h	NA	6.3 s	NA

*Ahmed Al-Herz (CS, Purdue)

Algorithms for b-Matching

$G = (V, E, w, b)$, $n = |V|$, $m = |E|$,
$\beta = \max_{v \in V} b(v)$, and $B = \sum_{v \in V} b(v)$.

- **Heuristic Algorithms**:
  - Heavy Edge Matching (HEM), $O(m \log \Delta)$
  - Easy to implement and parallelize.
  - Does not have any solution quality guarantee.
  - Solution depends on vertex processing order.

Algorithms for b-Matching

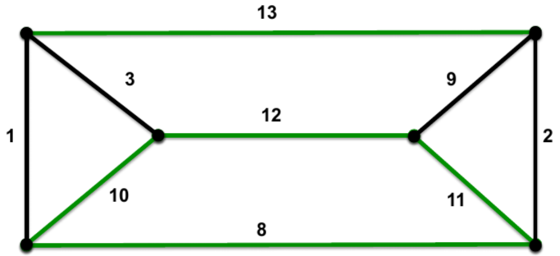
$G = (V, E, w, b)$, $n = |V|$, $m = |E|$,
$\beta = \max_{v \in V} b(v)$, and $B = \sum_{v \in V} b(v)$.

- **Approximation Algorithms**:
  - b-Suitor, $O(m \log \beta)$
  - 1/2-approximation algorithms: Solution weight is guaranteed to be 1/2 of the optimal weight.
  - Approximation guarantee is independent of vertex processing order.

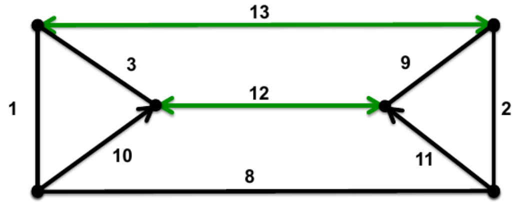
Approximation Algorithms for b-Matching

Strategy	Ratio	Matching	b-Matching
Greedy	1/2	Avis	Mestre
Path growing	1/2	Drake et al: PGA, PGA' Maue et al: GPA	Mestre
Local. Dom.	1/2	Preis, Manne et al : LD Birn et al: Local Max	Georgiadis et al: LD
Suitor	1/2	Manne & Halappanavar	**Khan et al.**
Aug Path	2/3 - ϵ	Pettie & Sanders	Mestre

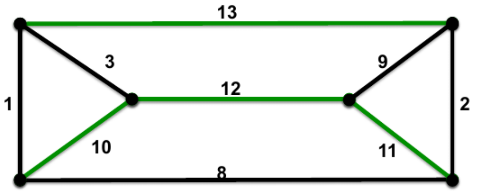
Greedy



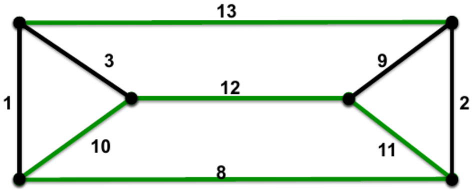
Locally Dominant Edges (LD)

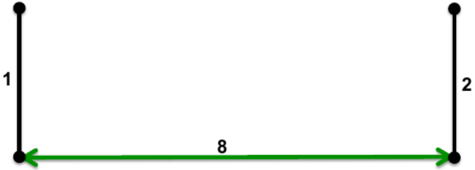


Locally Dominant Edges (LD)



Locally Dominant Edges (LD)





Core concept:

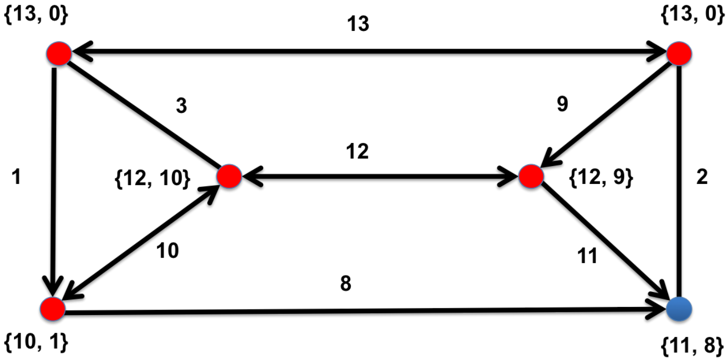
- Each unmatched vertex, u, **proposes** to its heaviest remaining neighbor v if v does not have **better offer already**.

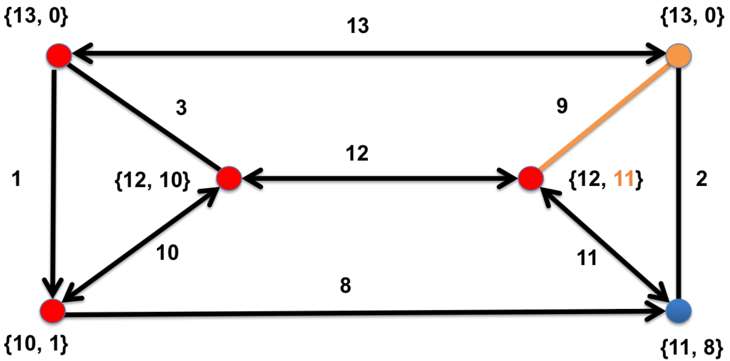
Data structure:

- A min priority heap, $S(v)$ of size $b(v)$ for each vertex v.
- If u proposes to v then $u \in S(v)$.

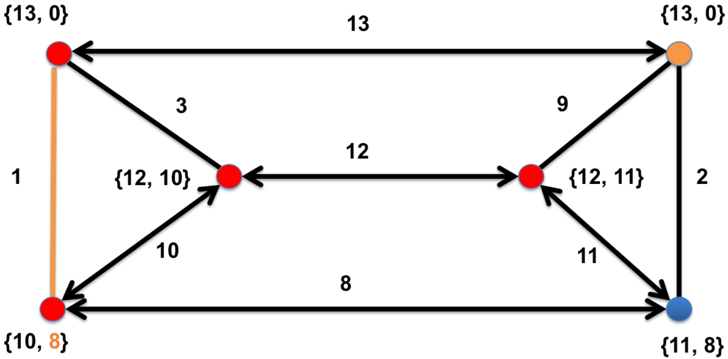
At termination:

$$v \in S(u) \iff u \in S(v)$$

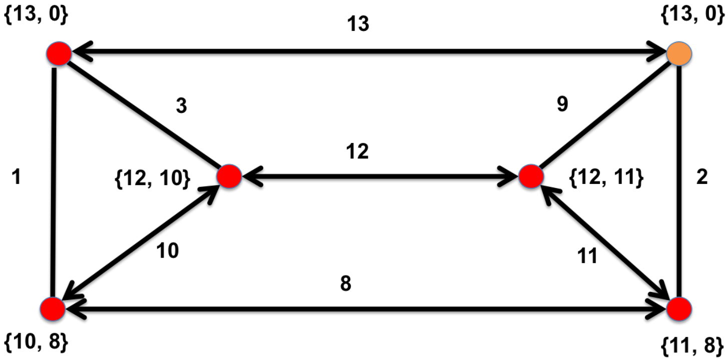

{13, 0}
13
{13, 0}
3
9
12
1
{12, 10}
{12, 9}
2
10
11
8
{10, 1}
{11, 8}


{13, 0}
13
{13, 0}
3
9
1
{12, 10}
12
{12, 11}
2
10
11
8
{10, 1}
{11, 8}

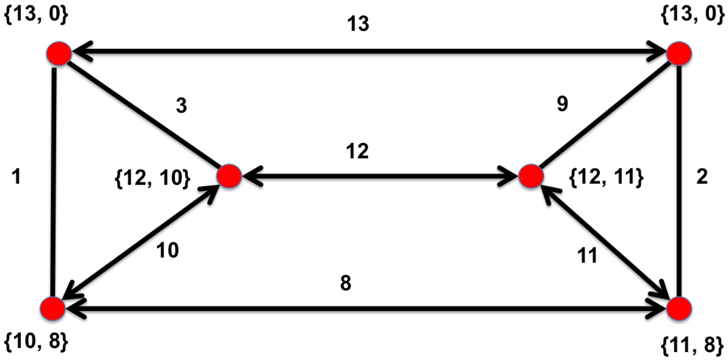
b-SUITOR



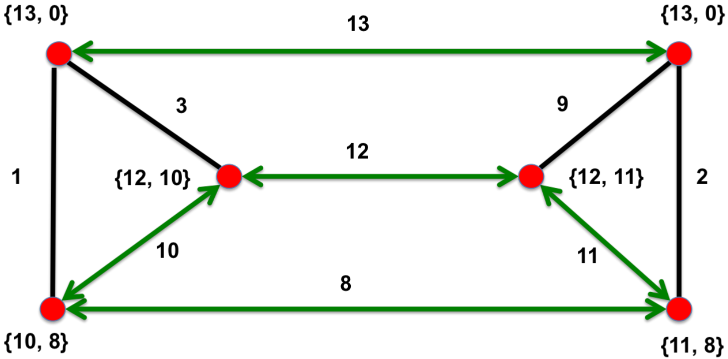
b-SUITOR



b-SUITOR



b-SUITOR



Characterisitcs of b-Suitor algorithm

- Greedy, LD and b-Suitor all compute exactly same matching!!
- Employing a global, local and *no ordering* respectively.
- For Greedy and LD: Once an edge is chosen, it enters in to the final solution.
- For b-Suitor: Proposals are made only based on **local information** and can be **annulled**. That is, b-Suitor is suitable for dynamic graphs.

Theory: b-SUITOR vs Other Approximation Algorithms

- b-SUITOR is the fastest known serial algorithm: ($\beta << \Delta << n$)
  - GREEDY: $O(m \log n)$, LD: $O(m \log \Delta)$ and b-SUITOR: $O(m \log \beta)$
- b-SUITOR has more concurrency than LD.
- The number of proposals is bounded by $O(B \log n)$ if the weights are randomly distributed.
  - This is obtained from the relationship of the b-MATCHING problem to the "Stable Fixtures" problem (generalization of Stable Matching).

Practice: *b*-SUITOR vs Other Approximation Algorithms

[Khan et. al, SISC'15]: Intel Xeon, 2.6 GHz, 16 Cores, 256 GB memory

- Serial Performance w.r.t *b*-SUITOR.
  - GREEDY: 16× slower.
  - PGA: 14× slower
  - LD: 6× slower
- Shared Memory Performance:
  - LD (16 cores): only 1.1× faster than *b*-SUITOR (serial).
  - *b*-SUITOR scales up to 13× with 16 Xeon cores.
  - *b*-SUITOR scales up to 50× with 60 Xeon Phi (KNC) cores.
- *b*-SUITOR requires 7× fewer edge traversals than LD.

Practice: *b*-SUITOR vs Other Approximation Algorithms

[Khan et. al, SISC'15]: Intel Xeon, 2.6 GHz, 16 Cores, 256 GB memory

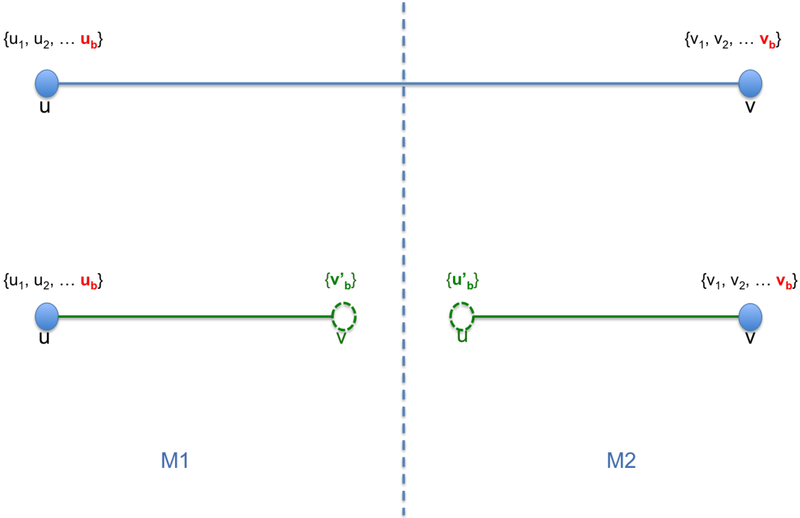
- Serial Performance w.r.t *b*-SUITOR.
  - GREEDY: 16× slower.
  - PGA: 14× slower
  - LD: 6× slower
- Shared Memory Performance:
  - LD (16 cores): only 1.1× faster than *b*-SUITOR (serial).
  - *b*-SUITOR scales up to 13× with 16 Xeon cores.
  - *b*-SUITOR scales up to 50× with 60 Xeon Phi (KNC) cores.
- *b*-SUITOR requires 7× fewer edge traversals than LD.

Practice: *b*-Suitor vs Other Approximation Algorithms

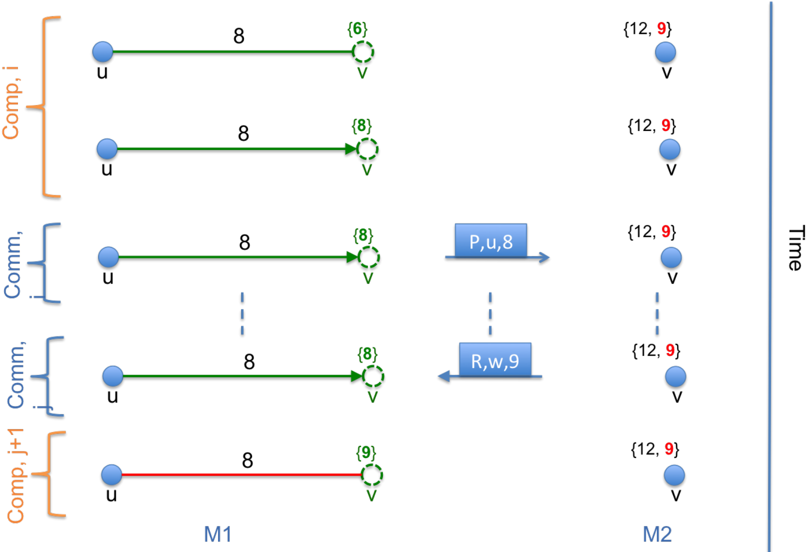
[Khan et. al, SISC'15]: Intel Xeon, 2.6 GHz, 16 Cores, 256 GB memory

- Serial Performance w.r.t *b*-Suitor.
  - Greedy: 16$\times$ slower.
  - PGA: 14$\times$ slower
  - LD: 6$\times$ slower
- Shared Memory Performance:
  - LD (16 cores): only 1.1$\times$ faster than *b*-Suitor (serial).
  - *b*-Suitor scales up to 13$\times$ with 16 Xeon cores.
  - *b*-Suitor scales up to 50$\times$ with 60 Xeon Phi (KNC) cores.
- *b*-Suitor requires 7$\times$ fewer edge traversals than LD.

Distributed Memory b-Suitor



Distributed b-SUITOR



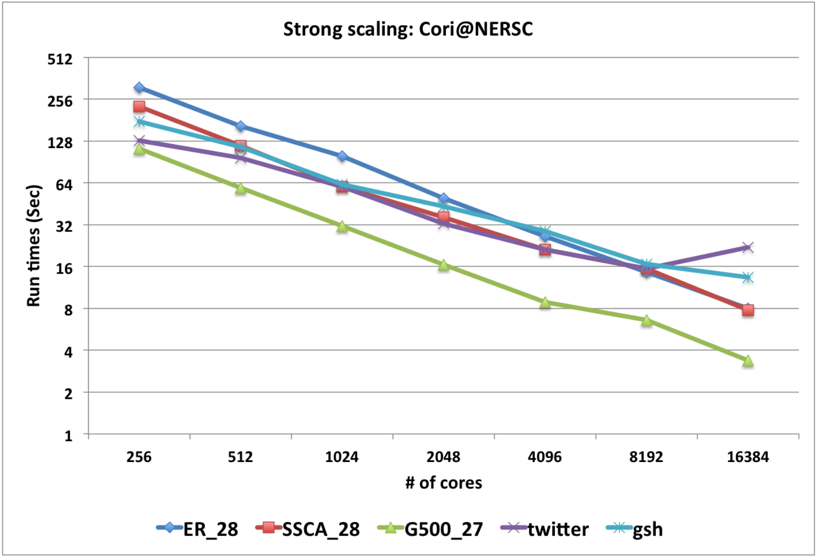
Strategies for Reducing Communication

- Subsetting the $b(v)$ values: $b' = \{1, 2, \ldots, 1/2b(v), \ldots, b(v)\}$.
- Subsetting the vertices on a compute node: $\{1, 2, \ldots\}$-way subsetting.
- Sorting the vertices on a compute node, based on their heaviest weight edges.

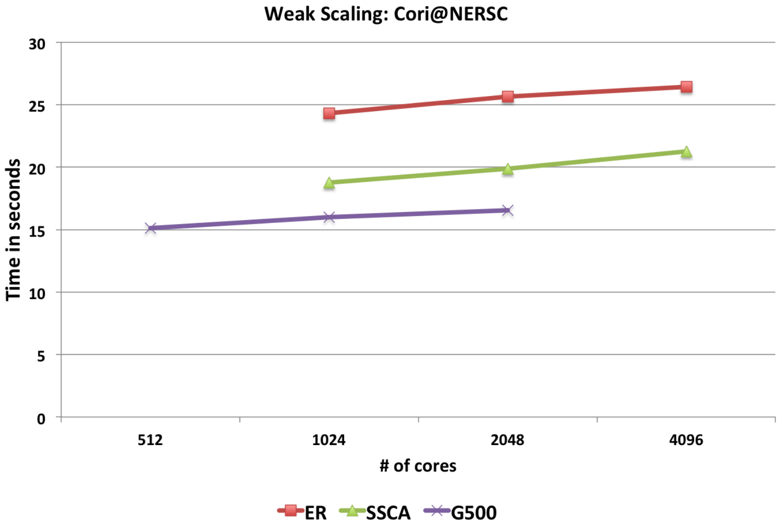
Problem Sets

Problems	Vertices	Edges	Avg. deg
ER_28	268,434,430	2,147,483,648	8
ER_27	134,217,028	1,073,741,824	8
ER_26	67,107,760	530,160,025	8
SSCA_28	268,435,154	2,136,323,325	8
SSCA_27	134,217,728	1,066,851,217	8
SSCA_26	67,107,987	534,179,576	8
G500_27	134,217,726	2,111,641,641	16
G500_26	67,108,089	1,073,058,343	16
G500_25	33,554,330	532,507,217	16
twitter	41,652,230	1,468,365,182	36
gsh-2015-host	68,680,142	1,802,747,600	27

Strong Scaling: *b*-SUITOR



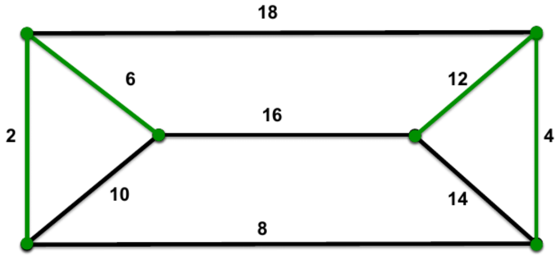
Weak Scaling: *b*-SUITOR



Conclusions: *b*-MATCHING

- A new 1/2- approximate *b*-MATCHING algorithm: *b*-SUITOR.
- *b*-SUITOR computes weights that are $>$ 97% of the optimal weights, for the (smaller) problems for which we can compute optimal weights.
- *b*-SUITOR outperforms the GREEDY and the LD algorithm w.r.t. to run time, and they all compute the same matching.
- The *b*-SUITOR algorithm scales on shared memory machines as well as on distributed memory machines with ten-thousands of processors.

- A min. weight b-EDGE COVER is a set of edges C such that **at least** $b(v)$ edges in C are incident on each vertex $v \in V$ and sum of the edge weights is minimized. For example, 1-Edge Cover:



Approx *b*-Edge Cover algorithms

Strategy	**Approx. Ratio**	Complexity	**Parallelizable**	**Algorithm**
Lightest Edge	Δ	$O(\beta m)$	Yes	* Hall & Hochbaum: Delta
Effective Weight	3/2	$O(m \log n)$	No	* Dobson: Greedy
Effective Weight & Local Sub Dom	3/2	$O(\beta m)$	Yes	**Khan et al: LSE**
Local Sub Dom	2	$O(\beta m)$	Yes	**Khan et al: S-LSE**
b-Matching	2	$O(m \log \beta')$	Yes	**Khan et al: MCE**

* Proposed for Set Multi-cover problem.

Relationship between b-Matching and b-Edge Cover

- **Optimal** b-Edge Cover using b-Matching [Schrijver]
  - Compute $b'(v) = \delta(v) - b(v)$, for each $v \in V$
  - Optimally solve *Max. Weight* b'-Matching, $M_{opt} \in E$.
  - Optimal *Min. Weight* b-Edge Cover, $C_{opt} = E \setminus M_{opt}$

Relationship between b-MATCHING and b-EDGE COVER

- What happens with approximate b-MATCHING ?
  - Compute $b'(v) = \delta(v) - b(v)$, for each $v \in V$
  - Approximately solve *Max. Weight* b'-Matching, $M' \in E$
  - ?? *Min. Weight* b-EDGE COVER, $C' = E \setminus M'$

Relationship between *b*-Matching and *b*-Edge Cover

- If approximate *b*-Matching solution edges have *locally dominant* property then the complemented *b*-Edge Cover solution will have approximation guarantee.
- *b*-Suitor (a 1/2- approximate b'-Matching) will give a 2-approximate *b*-Edge Cover we call it MCE algorithm.

Results

Problems	**b=1**	**b=5**
Fault_639	3.56%	1.13%
mouse_gene	12.12%	6.55%
Serena	4.65%	1.51%
bone010	2.00%	0.96%
dielFilterV3real	1.88%	0.11%
Flan_1565	9.33%	4.41%
kron_g500-logn21	**16.42%**	**13.53%**
hollywood-2011	5.52%	1.74%
G500_21	8.88%	3.26%
SSA21	12.30%	4.89%
eu-2015	6.78%	2.33%
Geo. Mean	**6.15%**	**2.14%**

Table: Solution quality of 2-approximation algorithms w.r.t 3/2-approximation algorithms.

Run times: Approximation algorithms for b-EDGE COVER

Intel Xeon (Haswell), 2.4 GHz, 36 Cores, 128 GB memory

- Serial Performance: w.r.t. MCE.
  - GREEDY: $21\times$ slower,
  - LSE: $9\times$ slower
  - S-LSE: $5\times$.
- Shared Memory Performance, w.r.t. serial MCE:
  - LSE (36 cores): only $3.7\times$ faster than MCE (serial)
  - MCE scales up to $30\times$ with 36 Intel Xeon (Haswell).
  - MCE scales up to $49\times$ with 68 Intel Xeon Phi (KNL) cores.

Contributions

- A new 3/2-approximate b-EDGE COVER algorithm: LSE.
- Showed that approximate b-MATCHING could be used to compute approximate b-EDGE COVER. This leads to the fastest and scalable approximation algorithm, called MCE.

Ongoing & Future Research

- Adaptive anonymity. (Google Research, NY)
- Graph sparsification and Community Detection. (PNNL)
- Recommender system and k-partite matching. (Netflix, Columbia)
- Resource allocation in Data Centers. (Microsoft Research)

Publications

- **Arif Khan**, Alex Pothen, Mostofa Patwary, Mahantesh Halappanavar, Nadathur Satish, Narayanan Sundaram, Pradeep Dubey. *Computing b-Matchings to Scale on Distributed Memory Multiprocessors by Approximation.* Supercomputing, 2016.
- **Arif Khan**, Alex Pothen. *A new 3/2-Approximation Algorithm for the b-Edge Cover Problem.* SIAM CSC, 2016.
- **Arif Khan**, Alex Pothen, Mostofa Patwary, Nadathur Satish, Narayanan Sundaram, Fredrik Manne, Mahantesh Halappanavar, Pradeep Dubey. *Efficient approximation algorithms for weighted b-Matching*. SIAM SISC, 2016.
- **Arif Khan**, David Gleich, Mahantesh Halappanavar & Alex Pothen. *A Multithreaded Algorithm for Network Alignment via Approximate Matching*. The International Conference for High Performance computing, Network, Storage and Analysis (Supercomputing), 2012.
- Mahantesh Halappanavar, Alex Pothen, Fredrik Manne, Ariful Azad, Johannes Langguth & **Arif Khan**, *Codesign Lessons Learned from Implementing Graph Matching Algorithms on Multithreaded Architectures*, IEEE Computer, pp. 46-55, August 2015.
- Ariful Azad, Mahantesh Halappanavar, Sivasankaran Rajamanickam, Erik G. Boman, **Arif Khan** & Alex Pothen. *Multithreaded Algorithms for Maximum Matching in Bipartite Graphs*. 26th IEEE International Parallel & Distributed Processing Symposium (IPDPS), 2012.