Scalable Algorithms for Graph Matching and Edge Cover Computations

Arif Khan\textsuperscript{1}, Alex Pothen\textsuperscript{1}, Mostofa Patwary\textsuperscript{2} & Pradeep Dubey\textsuperscript{2}

\textsuperscript{1}Computer Science, Purdue
\textsuperscript{2}Intel Parallel Computing Labs

March 1, 2017
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: \textit{b-Edge Cover}

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

- We design two new approximation algorithms: $3/2$-approximate Locally Subdominant Edge (LSE) and 2-approximate Static-LSE (S-LSE).

- \textbf{We establish the relationship between \textit{b-Edge Cover} and \textit{b-Matching} in the context of approximation algorithms.}

- Using \textit{b-Matching}, we design the most efficient algorithm MCE, a 2-approximation algorithm.
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]
$G = (V, E, w, b)$, $n = |V|$, $m = |E|$,  
$\beta = \max_{v \in V} b(v)$, 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.
<table>
<thead>
<tr>
<th>Graph</th>
<th>Vertices</th>
<th>Edges</th>
<th>Exact time</th>
<th>Exact weight</th>
<th>1/2-Approx. time</th>
<th>% opt. wt./</th>
</tr>
</thead>
<tbody>
<tr>
<td>IG5-16</td>
<td>37K</td>
<td>588K</td>
<td>10 s</td>
<td>1.4 e4</td>
<td>1.6e-2 s</td>
<td>98.7 %</td>
</tr>
<tr>
<td>Image-interp</td>
<td>360K</td>
<td>712K</td>
<td>1.2 s</td>
<td>1.5 e8</td>
<td>3.5e-2 s</td>
<td>96.5 %</td>
</tr>
<tr>
<td>LargeRegFile</td>
<td>2.9M</td>
<td>4.9M</td>
<td>6.9 s</td>
<td>9.7 e8</td>
<td>0.2 s</td>
<td>98.9 %</td>
</tr>
<tr>
<td>Rucci1</td>
<td>2.1M</td>
<td>7.8M</td>
<td>4 h 36 m</td>
<td>1.6 e8</td>
<td>1.3 s</td>
<td>99.7 %</td>
</tr>
<tr>
<td>GL7d16</td>
<td>1.5M</td>
<td>14.5M</td>
<td>9 h 50 m</td>
<td>5.8 e8</td>
<td>1.3 s</td>
<td>94.5 %</td>
</tr>
<tr>
<td>GL7d20</td>
<td>3.3M</td>
<td>29.9M</td>
<td>&gt; 100 h</td>
<td>NA</td>
<td>4.8 s</td>
<td>NA</td>
</tr>
<tr>
<td>GL7d18</td>
<td>3.5M</td>
<td>35.6M</td>
<td>&gt; 100 h</td>
<td>NA</td>
<td>5.5 s</td>
<td>NA</td>
</tr>
<tr>
<td>GL7d19</td>
<td>3.9M</td>
<td>37.3M</td>
<td>&gt; 100 h</td>
<td>NA</td>
<td>6.3 s</td>
<td>NA</td>
</tr>
</tbody>
</table>

*Ahmed Al-Herz (CS, Purdue)
$G = (V, E, w, b)$, $n = |V|$, $m = |E|$, 
$\beta = \max_{v \in V} b(v)$, 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.
$G = (V, E, w, b)$, $n = |V|$, $m = |E|$, 
$\beta = \max_{v \in V} b(v)$, 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

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Ratio</th>
<th>Matching</th>
<th>$b$-Matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>1/2</td>
<td>Avis</td>
<td>Mestre</td>
</tr>
<tr>
<td>Path growing</td>
<td>1/2</td>
<td>Drake et al: PGA, PGA' Maue et al: GPA</td>
<td>Mestre</td>
</tr>
<tr>
<td>Suitor</td>
<td>1/2</td>
<td>Manne &amp; Halappanavar</td>
<td>Khan et al.</td>
</tr>
<tr>
<td>Aug Path</td>
<td>2/3 - $\epsilon$</td>
<td>Pettie &amp; Sanders</td>
<td>Mestre</td>
</tr>
</tbody>
</table>
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:
$$v \in S(u) \iff u \in S(v)$$
Characteristics of \textit{b-Suitor} algorithm

- \textbf{Greedy}, LD and \textit{b-Suitor} all compute exactly same matching!!

- Employing a global, local and \textit{no ordering} respectively.

- For \textbf{Greedy} and LD: Once an edge is chosen, it enters in to the final solution.

- For \textit{b-Suitor}: Proposals are made only based on \textit{local information} and can be \textbf{annulled}. That is, \textit{b-Suitor} is suitable for dynamic graphs.
Theory: \textit{b-Suitor} vs Other Approximation Algorithms

- \textit{b-Suitor} is the fastest known serial algorithm: $(\beta \ll \Delta \ll n)$
  - \textbf{Greedy}: $O(m \log n)$, \textbf{LD}: $O(m \log \Delta)$ and \textbf{b-Suitor}: $O(m \log \beta)$
- \textit{b-Suitor} has more concurrency than \textit{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 \textit{b-Matching} problem to the ”Stable Fixtures” problem (generalization of Stable Matching).
[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.
[Khan et. al, SISC’15]: Intel Xeon, 2.6 GHz, 16 Cores, 256 GB memory

- Serial Performance w.r.t \textit{b-SUITOR}.
  - \textbf{Greedy}: 16\times slower.
  - PGA: 14\times slower
  - LD: 6\times slower

- Shared Memory Performance:
  - LD (16 cores): only 1.1\times faster than \textit{b-SUITOR} (serial).
  - \textit{b-SUITOR} scales up to 13\times with 16 Xeon cores.
  - \textit{b-SUITOR} scales up to 50\times with 60 Xeon Phi (KNC) cores.

- \textit{b-SUITOR} requires 7\times fewer edge traversals than LD.
[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.
Distributed Memory \( b\)-SUITOR

\[ \{u_1, u_2, \ldots, u_b\} \]

\[ \{v_1, v_2, \ldots, v_b\} \]

\[ \{u_1, u_2, \ldots, u_b\} \quad \{v'_b\} \quad \{u'_b\} \quad \{v_1, v_2, \ldots, v_b\} \]

M1

M2
Distributed \textit{b-Suitor}

\begin{itemize}
\item \textbf{Comp, i} \hfill \textbf{Comm, i}
\begin{itemize}
\item \textbf{Comp, j+1} \hfill \textbf{Comm, i}
\end{itemize}
\end{itemize}
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.
<table>
<thead>
<tr>
<th>Problems</th>
<th>Vertices</th>
<th>Edges</th>
<th>Avg. deg</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER_28</td>
<td>268,434,430</td>
<td>2,147,483,648</td>
<td>8</td>
</tr>
<tr>
<td>ER_27</td>
<td>134,217,028</td>
<td>1,073,741,824</td>
<td>8</td>
</tr>
<tr>
<td>ER_26</td>
<td>67,107,760</td>
<td>530,160,025</td>
<td>8</td>
</tr>
<tr>
<td>SSCA_28</td>
<td>268,435,154</td>
<td>2,136,323,325</td>
<td>8</td>
</tr>
<tr>
<td>SSCA_27</td>
<td>134,217,728</td>
<td>1,066,851,217</td>
<td>8</td>
</tr>
<tr>
<td>SSCA_26</td>
<td>67,107,987</td>
<td>534,179,576</td>
<td>8</td>
</tr>
<tr>
<td>G500_27</td>
<td>134,217,726</td>
<td>2,111,641,641</td>
<td>16</td>
</tr>
<tr>
<td>G500_26</td>
<td>67,108,089</td>
<td>1,073,058,343</td>
<td>16</td>
</tr>
<tr>
<td>G500_25</td>
<td>33,554,330</td>
<td>532,507,217</td>
<td>16</td>
</tr>
<tr>
<td>twitter</td>
<td>41,652,230</td>
<td>1,468,365,182</td>
<td>36</td>
</tr>
<tr>
<td>gsh-2015-host</td>
<td>68,680,142</td>
<td>1,802,747,600</td>
<td>27</td>
</tr>
</tbody>
</table>
Strong Scaling: \textit{b-SUITOR}

![Graph showing strong scaling results for Cori@NERSC](image)
Weak Scaling: \textit{b-SUITOR}

\begin{center}
\textbf{Weak Scaling: Cori@NERSC}
\end{center}

\begin{figure}
\centering
\begin{tikzpicture}
\begin{axis}[
    title={Weak Scaling: Cori@NERSC},
    xlabel={\# of cores},
    ylabel={Time in seconds},
    xmin=512, xmax=4096,
    ymin=0, ymax=30,
    xtick={512,1024,2048,4096},
    ytick={0,5,10,15,20,25,30},
    legend style={at={(0.5,0.2)},anchor=north},
    legend entries={ER, SSCA, G500},
]
\addplot [red,mark=x] coordinates {
    (512,15)
    (1024,18)
    (2048,21)
    (4096,24)
};
\addplot [green,mark=triangle] coordinates {
    (512,12)
    (1024,14)
    (2048,16)
    (4096,18)
};
\addplot [blue,mark=x] coordinates {
    (512,10)
    (1024,12)
    (2048,14)
    (4096,16)
};
\end{axis}
\end{tikzpicture}
\end{figure}
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 \textbf{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

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

* Proposed for Set Multi-cover problem.


- **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, \( M_{\text{opt}} \in E \).
  - Optimal Min. Weight \( b\)-Edge Cover, \( C_{\text{opt}} = E \setminus M_{\text{opt}} \)
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, $C' = E \setminus M'$
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

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

**Table**: Solution quality of 2-approximation algorithms w.r.t 3/2-approximation algorithms.
Intel Xeon (Haswell), 2.4 GHz, 36 Cores, 128 GB memory

- Serial Performance: w.r.t. MCE.
  - Greedy: 21× slower,
  - LSE: 9× slower
  - S-LSE: 5×.

- Shared Memory Performance, w.r.t. serial MCE:
  - LSE (36 cores): only 3.7× faster than MCE (serial)
  - MCE scales up to 30× with 36 Intel Xeon (Haswell).
  - MCE scales up to 49× 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)


