Vertex Weighted Matching: Parallel Approximation Algorithms

Ahmed Al-Herz and Alex Pothen
Department of Computer Science
Purdue University, USA

July 11, 2019
Outline

- Matching concepts
- Serial 2/3-approximation algorithms
- Parallel 2/3-approx. algorithm and Synchronization
- Experimental results
- Conclusions and References
Definitions

- Matching: a set of vertex-disjoint edges; hence at most one edge incident on each vertex.
- Maximum cardinality, edge-weighted matching
- We consider maximum vertex weighted matching. Arises in internet advertising as well as in computing sparse bases for the null space and column space of matrices, crew scheduling, etc.
There is an $O(1 - 1/e) \approx 0.6$-approx. algorithm for the online matching problem.
Online Vertex-Weighted Matching

- We report the first parallel $2/3$-approx. algorithm (offline) for maximum vertex-weighted matching.
Matching concepts

- Alternating path: a path that has alternating matching and non-matching edges.
- Augmenting path: an odd length alternating path that begins and ends with unmatched edges.

By swapping matching and non-matching edges, we increase the weight and the cardinality of the matching.
Matching concepts

- (Weight)-Increasing path: an even length alternating path whose unmatched start vertex $u$ has higher weight than its matched end vertex $v$; hence $\phi(u) > \phi(v)$.

- By swapping matching and non-matching edges, we increase the weight (but not the cardinality) of the matching.
A More General Definition

- $k$-augmentation: an augmenting path or increasing path with at most $k$ non-matching edges.
- 2-augmentation: augmenting paths of lengths one and three; increasing paths of length two and four.

**Theorem**

If a matching does not admit a $k$-augmentation, then it is a $k/(k+1)$-approximation to a maximum weight matching.

- Theorem is true for maximum cardinality and maximum edge-weighted matchings as well.
2/3-Direct Approximation Algorithm

1: procedure 2/3-DIRECT($G = (V, E, \phi)$)  
2: Initialize the matching $M$ to be empty;  
3: Order the vertices in non-increasing order of weights;  
4: for each unmatched vertex $u$ in order do  
5: Search for an aug. path $P$ of length at most 3 that reaches a heaviest unmatched vertex $v$;  
6: If $P$ is found, augment the matching $M$ with $M \oplus P$;  
7: end for  
8: end procedure

Dobrian, Halappanavar, Pothen, Al-Herz, SISC (2019)
2/3-Iterative Approximation Algorithm

1: **procedure** 2/3-ITER($G = (V, E, \phi)$)
2: 
3:   [Initialize with a 2/3-approx. cardinality matching;]
4: **while** a 2-augmentation exists **do**
5:   Search for a 2-augmentation $P$ from $u$;
6:   If $P$ is found, augment the matching with $M \oplus P$;
7:   Update the set of unmatched vertices $U$;
8: **end while**
9: **end procedure**
Direct vs. Iterative Algorithms for MVM

- The Direct algorithm employs augmenting paths only, and no increasing paths; Iterative algorithm employs both.
- The Direct algorithm sorts vertices and matches vertices in non-decreasing order of weights, while the Iterative algorithm processes unmatched vertices in any order, and hence can be implemented in parallel.
- The Iterative algorithm can be initialized with another matching; the Direct algorithm cannot be initialized.
- Worst-case time complexity ($\Delta$ is maximum degree, $n$ no. of vertices, $m$ no. of edges):
  - Direct: $O(m \log \Delta + n \log n)$
  - Iterative: $O(m\Delta^2)$
Parallel 2/3-approximate cardinality matching algorithm

1:  procedure PAR-2/3-ITER($G = (V, E, \phi)$)
2:  Initialize with 2/3-approx. cardinality matching computed in parallel;
3:  while a 2-augmentation exists in parallel do
4:    Search for a 2-augmentation $P$ from an unmatched vertex $u$;
5:    If $P$ is found, try to lock vertices on $P$;
6:    If locks obtained, augment $M$ with $M \leftarrow M \oplus P$;
7:    Release all locks;
8:    Update set of unmatched vertices;
9:  end while
10: end procedure
Threads $\{T_i\}$ computes an increasing path beginning at $u_i$ that includes matched edges $(v_{2i-1}, v_{2i})$ and $(v_{2i+1}, v_{2i+2})$. Thread $T_k$ includes matched edges $(v_{2k-1}, v_k)$ and $(v_1, v_2)$. 
If each thread $T_i$ locks $v_{2i-1}$, we have a cyclic wait!
Cyclic wait happens also for augmenting paths of length 3.
Locking procedure

- If $u, v$ are the endpoints of an augmenting path, then we permit a thread to augment from $u$ to $v$ only if $u < v$.
- We lock the lower numbered endpoint of a matching edge.

$\begin{align*}
\text{Lock (in order) } v_1, v_4, \min\{v_2, v_3\}
\end{align*}$

$\begin{align*}
\text{Let } m_1 = \min\{v_1, v_2\}, m_2 = \min\{v_3, v_4\}
\text{Lock (in order) } u, \min\{m_1, m_2\} \text{ and } \max\{m_1, m_2\}
\end{align*}$
Parallel Iterative Algorithm and Locks

- If a thread fails to acquire a lock, it releases all locks and processes the next unmatched vertex or moves to the next iteration.

- **Bad things that could happen:**
  - Deadlock: some thread waiting for ever to acquire a lock.
  - Livelock: some threads cannot make any progress in matching their vertices.
  - Starvation: some thread cannot match any vertices at all.

**No bad news Theorem**
While there is a 2-augmentation in the graph, in every iteration of the parallel iterative algorithm at least one thread will succeed in matching a vertex.
The set of test problems

| Graph          | $|V|$     | Degree | $|E|$      |
|----------------|---------|--------|-----------|
|                |         | Mean   | SD/ Mean  |
| kron_g500      | 2 097 152 | 117.92 | 7.47      | 91 040 932 |
| M6             | 3 501 776 | 5.99   | 0.14      | 10 501 936 |
| hugetric       | 6 592 765 | 2.99   | 0.01      | 9 885 854  |
| rgg_n_2_23_s0  | 8 388 608 | 15.14  | 0.26      | 63 501 393 |
| hugetrace      | 12 057 441 | 2.99 | 0.01      | 18 082 179 |
| nlpkkt200      | 16 240 000 | 26.60  | 0.09      | 215 992 816 |
| hugebubbles    | 19 458 087 | 2.99   | 0.01      | 29 179 764 |
| road_usa       | 23 947 347 | 2.41   | 0.39      | 28 854 312 |
| europe_osm     | 50 912 018 | 2.12   | 0.23      | 54 054 660  |
| rmat-G500      | 48 877 747 | 85.28  | 15.48     | 2 084 251 521 |
| rmat-SSCA      | 93 488 461 | 45.29  | 9.96      | 2 117 212 258 |
| rmat-ER        | 134 217 728 | 32.00  | 0.29      | 2 147 483 625 |
Running Time of Algorithms

- Running time of a Maximum vertex weighted matching algorithm on these problems ranges from 10 to 6000 seconds.

- Relative performance of 6 approximation algorithms are computed by the ratio of the time for the Exact algorithm and the time for the approximation algorithm.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$1 - \epsilon$-Scal</th>
<th>$2/3 - \epsilon$-GPA-ROMA</th>
<th>$2/3$-DIR</th>
<th>$2/3$-ITER</th>
<th>$1/2$-ITER</th>
<th>Suitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon = 1/3$</td>
<td>$\epsilon = 0.01$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geom. Mean</td>
<td>0.63</td>
<td>1.3</td>
<td>23</td>
<td>40</td>
<td>110</td>
<td>43</td>
</tr>
</tbody>
</table>
Run times vs. Edges Searched

- 2/3-IITER
- Linear (2/3-IITER)
- Suitor
- Linear (Suitor)
- GPA-ROMA
- Linear (GPA-ROMA)
Computed weights

Gap to optimality (%) is

\[
(1 - \frac{\text{Weight of approx. matching}}{\text{Weight of maximum matching}}) \times 100.
\]

<table>
<thead>
<tr>
<th>Graph</th>
<th>1 - (\epsilon)-Scal.</th>
<th>2/3 - (\epsilon)</th>
<th>2/3-(\epsilon)-GPA-ROMA</th>
<th>2/3-(\epsilon)-DIR</th>
<th>1/2-(\epsilon)-ITER</th>
<th>1/2-(\epsilon)-ITER</th>
<th>Suitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\epsilon = 1/3)</td>
<td>(\epsilon = 0.01)</td>
<td>Geom. Mean</td>
<td>1.25</td>
<td>0.23</td>
<td>0.32</td>
<td><strong>0.084</strong></td>
<td>0.76</td>
</tr>
</tbody>
</table>

The 2/3-approx. algorithms compute nearly optimal weights!
Speedup on 20 Xeon threads
Conclusions

- We have described a new serial $2/3$-approximation algorithm for the maximum vertex-weighted matching problem.
- This algorithm can be adapted to provide the first parallel algorithm with approximation ratio better than $1/2$ for any matching problem.
- Vertex-weighted matching problems provide hard test cases for edge-weighted matching problems.
## A Puzzle

**Exact Edge-weighted Matching on GL7D20**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Random Weights</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original</td>
<td>edge [0, 1]</td>
<td>vertex [0, 0.5] summed</td>
</tr>
<tr>
<td>Time (s)</td>
<td>4.61 E0</td>
<td>1.330 E1</td>
<td>1.001 E5</td>
</tr>
<tr>
<td>Cardinality</td>
<td>1,437,546</td>
<td>1,437,546</td>
<td>1,437,546</td>
</tr>
<tr>
<td>No. augs.</td>
<td>1,437,546</td>
<td>1,437,546</td>
<td>1,437,546</td>
</tr>
<tr>
<td>Aug. path length</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum</td>
<td>9</td>
<td>61</td>
<td>2383</td>
</tr>
<tr>
<td>No. distinct</td>
<td>5</td>
<td>30</td>
<td>938</td>
</tr>
<tr>
<td>Mean</td>
<td>1.009</td>
<td>1.896</td>
<td>72.55</td>
</tr>
<tr>
<td>No. dual updates</td>
<td>1.35 E7</td>
<td>7.49 E6</td>
<td>3.92 E10</td>
</tr>
<tr>
<td>Time aug. paths</td>
<td>3.38</td>
<td>9.72</td>
<td>4.31 E4</td>
</tr>
<tr>
<td>Time dual updates</td>
<td>0.42</td>
<td>0.27</td>
<td>4.44 E3</td>
</tr>
</tbody>
</table>
References


