Distributed Approximate Matching

Zvi Lotker, Boaz Patt-Shamir, Adi Rosēn

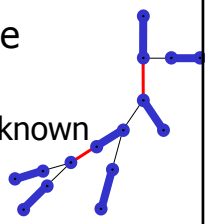
Problem Statement

- Input: a Graph G(V,E)
- The system is synchronous
  - In each round, each node sends O(log n) bits to each neighbor
  - time complexity = number of rounds
- We consider two models:
  - Static
    - Goal: compute a matching from scratch
  - Dynamic
    - Nodes (with their edges) are inserted and deleted one at a time

2

Matching

- A matching M(G) of a graph G: A set of non-intersecting edges
- A maximum matching M(G) is a matching whose size is the maximum among all matching of G
- Maximum **weighted** matching: edges have weights…
  - $w:E \rightarrow \Re$ is a weight. We assume the weight is known to both endpoints of the edge e=(i,j)



3

Previous work - general graphs (sequential)

- Unweighted: Micali and Vazirani [80] $O(|V|^{1/2}|E|)$ time algorithm.
- Weighted graphs: Gabow [90] $O(|V||E|+|V|^2 \log |V|)$ time algorithm.

4

Previous work - distributed algorithms

- Israeli & Itai [86]: 2-approximation of unweighted matching (maximal matching) in time O(log|V|).
- Wattenhofer[2] [04]: 5-approximation in $O(\log^2 n)$ time.
- Kuhn, Moscibroda & Wattenhofer [05]: no constant approximation (even randomized) in time smaller than $\Omega\left(\min\left(\sqrt{\frac{\log n}{\log\log n}}\ ,\ \frac{\log\Delta}{\log\log\Delta}\right)\right)$

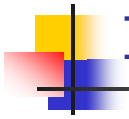
Δ is the maximal degree in the graph

5

Our Results

- Static case:
  - $4+\varepsilon$ approximation for weighted matching of an arbitrary weighted graph in O(log n) time
- Dynamic case:
  - Maintain $1+\varepsilon$ approximation of maximum unweighted matching in $O(1/\varepsilon)$ time for each topological change.
  - Maintain O(1) approximation of maximum weighted matching in constant time for each topological change.

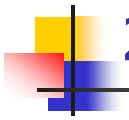
6



In the remainder of the talk…

- Approximate maximum weighted matching in static graphs
  - Basic ideas
  - Algorithm overview
- Maintenance of approximate maximum matching in unweighted graphs

7



2+ε -approx in O(log^2(n)) time

Idea #1:

- divide the edges into classes of roughly the same weight
- Run algorithm for unweighted matching in each class, sequentially starting with heaviest

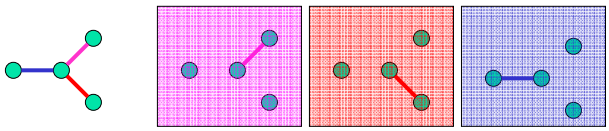
- If weight classes are powers of 1+ε: get approximation factor 2+2ε
- Time complexity: O(log^2(n)) rounds

8

O(1)-approx in O(log(n)) time

Idea #2: divide the edges into classes of roughly the same weight, but now…

- Run algorithm for unweighted matching in each class in parallel
- Time complexity: O(log(n)) rounds
- Problem now: result is not a matching!



9

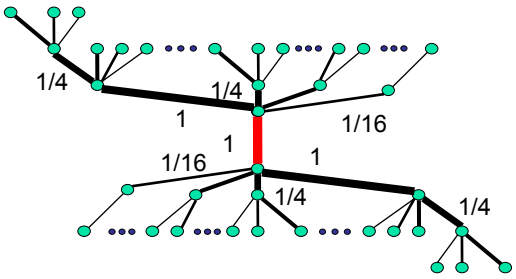
O(1)-approx in O(log(n)) time

- Need to resolve conflicts at nodes with many "matching" edges!
- Idea:
  - Each node selects the heaviest edge output by an "unweighted" matching
  - An edge is output by the algorithm only if selected by both endpoints
- O(1) additional time, O(1) approx

10

Resolve conflicts between different sub classes

- Each unit weight in the final output may have rejected about 16 units of OPT



11

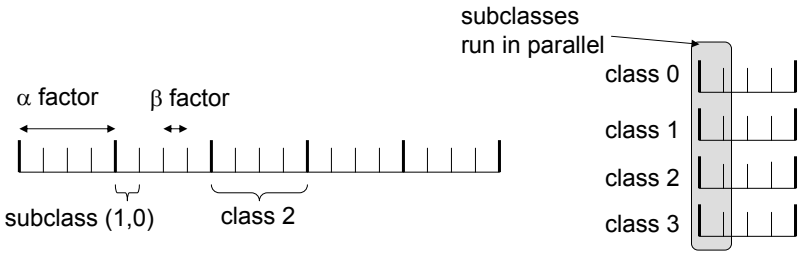
4+ε approx in O(log(n)) time

- Idea: careful combination of parallel and serial executions of the weighted matching algorithm
- We run a sequence of iterations
- In each iteration, many classes are run in parallel

12

4+ε approx in O(log(n)) time

- Partition weight according to edge weights by a two-level hierarchy.
  - $\alpha=1+1/\varepsilon$, $\beta=1+\varepsilon$
- class i includes all edges e with $w(e)\in[\alpha^i, \alpha^{i+1})$.
  - Each class is further divided into $k=\log_\beta\alpha$ subclasses.

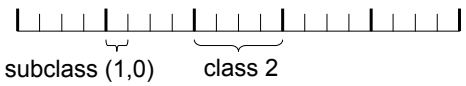


13

4+ε approx in O(log(n)) time

In iteration j, run in parallel an independent instance of unweighted matching for each subclass (*,j)

- If (u,v) is selected in subclass (i,j), we remove nodes v,u from all classes of weights smaller than (i,j).



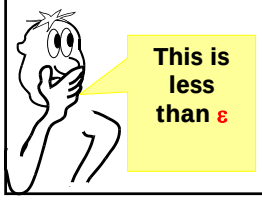
The result is not a matching!

- Subclasses in the same (i,*) class are sequential
- But subclasses from different classes (*,j) run in parallel
- We also improve the conflict resolution

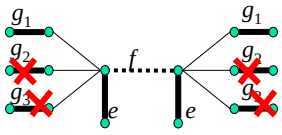
14

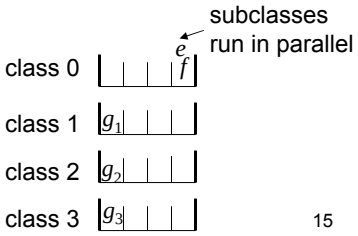
Resolve conflicts between different subclasses

- For i=1 to log(n)
  - Each node v pick the heaviest edge in $\mathcal{A}$
  - If edge (v,u) is selected by both v,u, then output (v,u), and remove (v,*),(u,*) from $\mathcal{A}$



The worst case is

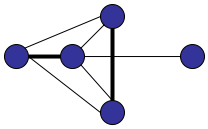




15

Dynamic case: The problem

- Input: a topological change
  - Insertion of a single node
  - Deletion of a single node



- Response: changing the matching edges
  - a distributed computation. Output: a local indication for each edge (in matching or not?)

16

Some basic facts form matching theory (1)

Let M be a matching.

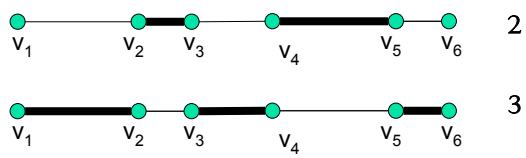
- A path is **alternating** if it consists of edges drawn alternately from M and E-M.
- An **augmenting** path is an alternating path whose endpoints are unmatched nodes
- Key property:
  If P is an augmenting path then $M \oplus P$ is also a matching, and $|M \oplus P| = |M|+1$

17

Some basic facts form matching theory (2)

Theorem: If there is no augmenting path of length 2k −1 or less w.r.t. M, then the size of the largest matching in G is at most

$$|M| \cdot \frac{k}{k+1}$$



18

The main idea of algorithm

Implication: Sufficient to eliminate all augmenting paths whose length is at most $2/\varepsilon - 1$.

Easy to see: After insertion, such a path must start with the new node

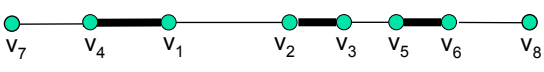
Harder to see: Sufficient to augment once, along the **shortest** path

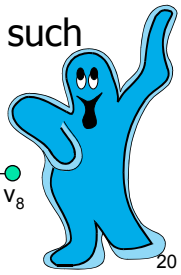
19

The main idea of algorithm

Algorithm: When a new node v' is inserted…

- Search for all augmenting paths that start with v' and whose length is at most $2/\varepsilon - 1$.
  - If no such path exists, we are done.
  - Otherwise, augment along the shortest such path.

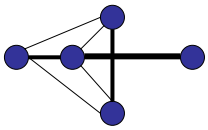




20

Deleting an node

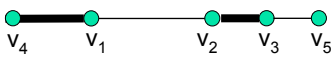
- We can emulate deleting a node v by inserting nodes.
  - If node v is not matched then we can removing all the edges (v,*).
  - If node v is matched to node u then we remove node v,u and insert the node u.



21

Lower bounds

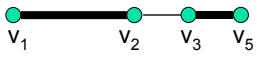
- Theorem
  - $\Omega(1/\varepsilon)$-time is necessary to maintain $(1+\varepsilon)$ approximation of maximum matching in dynamic unweighted graphs.
  - Assume $0<\varepsilon= 1/k\in N$. Consider a line of $2(k-1)$ edges. The size of the maximum matching in this graph is $k - 1$, so a $(1+\varepsilon)$-approximation must be a maximum matching



22

Lower bounds

- Theorem
  - $\Omega(1/\varepsilon)$-time is necessary to maintain $(1+\varepsilon)$ approximation of maximum matching in dynamic unweighted graphs.
  - Assume $0<\varepsilon= 1/k\in N$. Consider a line of $2(k-1)$ edges. The size of the maximum matching in this graph is $k - 1$, so a $(1+\varepsilon)$-approximation must be a maximum matching



23

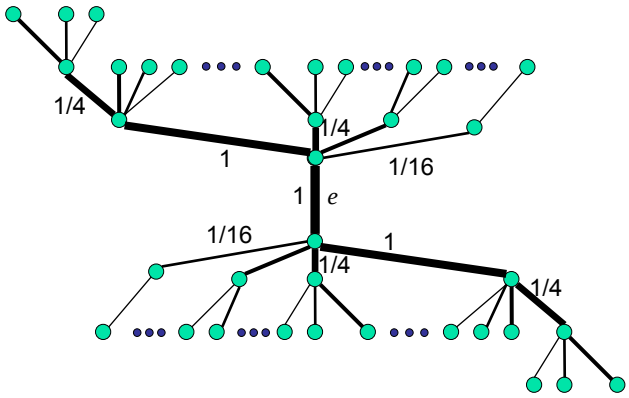
Dynamic Weight

- Our algorithm is based on the idea to reduce the weighted case to the unweighted case.
  - We partition the edges into disjoint classes, where all edges in class $i \geq 0$ have weight in $[4^i, 4^{i+1})$.
  - When a node is inserted, it initiates the unweighted algorithm for each weight class
  - Each node then picks, as a candidate for the output, the matched incident edge in the highest weight class.

24

Dynamic Weight worst case

- Each edge in the matching can remove the following tree in the worst case



25

From Dynamic to Static The idea of the algorithm

- Our approach is to reduce the weighted case to multiple instances of the unweighted case.
- The main question is how to merger the multiple instances

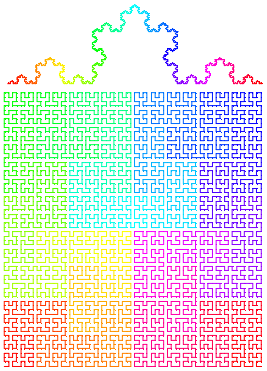
26

Open problem

- Can you have an 1+ε approx

27

End



28