Computing Approximate Weighted Matchings in Parallel

Fredrik Manne, University of Bergen

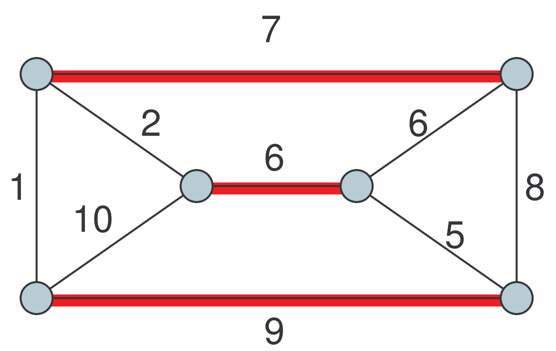
with

Rob Bisseling, Utrecht University
Alicia Permell, Michigan Tech. University

The Edge Weighted Matching Problem

Given an edge weighted graph *G(V,E).*

Select an independent set S of edges of maximum weight.



W(opt) = 18

Best known algorithm has running time $O(|V||E|+|V|^2 log|V|)$

Often too expensive for real applications

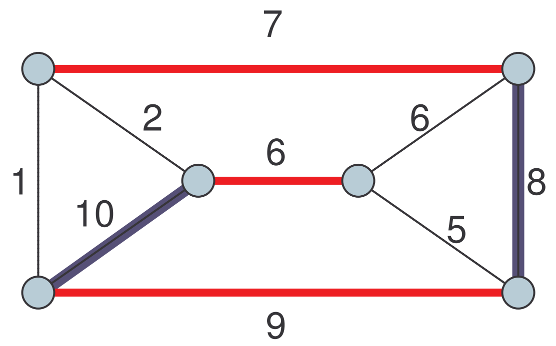
Fast Approximation Algorithms

The Greedy Approach

While there are edges left

Add most expensive remaining edge *(v,w)* to *S*

Remove *(v,w)* and all edges incident on *v* and *w*.



$W(S) \geq \frac{1}{2} W(opt)$

Running time $O(|E| \log |V|)$ (due to sorting)

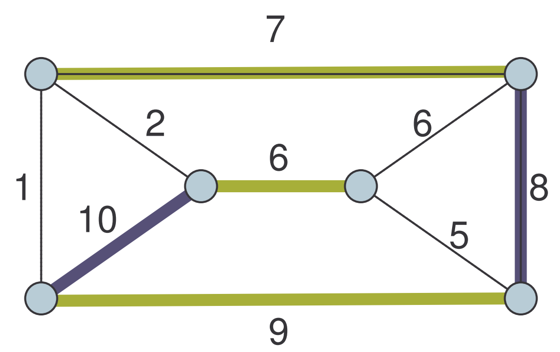
Parallelization: Difficult to sort distributed weights

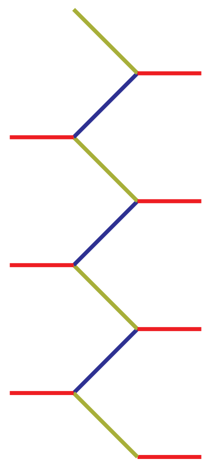
Fast Approximation Algorithms

Path growing [Drake and Hougardy]

Grow non-intersecting path along heaviest edge
while coloring edges alternatively **green** or **blue**.

Return heaviest of **green** and **blue**.





W(G) + W(B) ≥ W(opt)

Max{W(G),W(B)} ≥ ½W(opt)

Running time $O(|V| + |E|)$

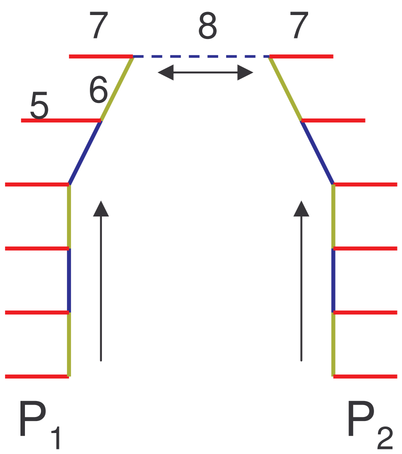
Also inherently sequential

A Parallel Path Growing Algorithm

Idea: Grow multiple paths at the same time

Assumes shared memory model

A potential problem



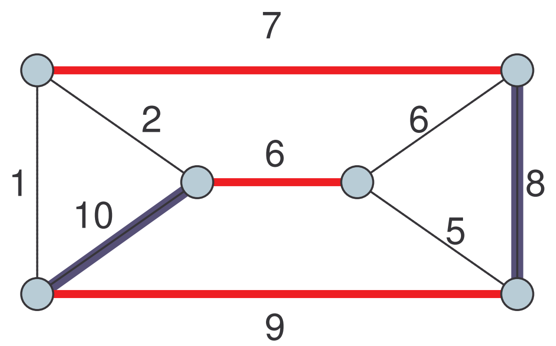
Cannot guarantee that we get 0.5 approximation

Experiments using UPC on random graphs indicate that this does not affect quality too much.

A New Sequential Algorithm

From the analysis of the Greedy algorithm:

A chosen edge must dominate its remaining neighborhood



Local Domination Algorithm

D = all dominating edges in *G*

While *D* is not empty

Remove *(v,w)* from *D* and add to *S*

Remove all edges incident on *v* and *w* from *G*

Add any new dominating edges to *D*

Observation

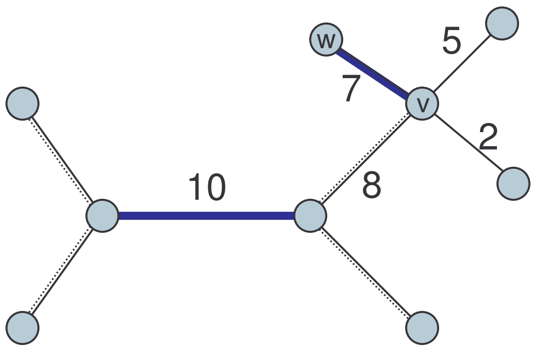
If the weights are unique, then the algorithm will produce the same matching as the Greedy algorithm.

Implementation Details

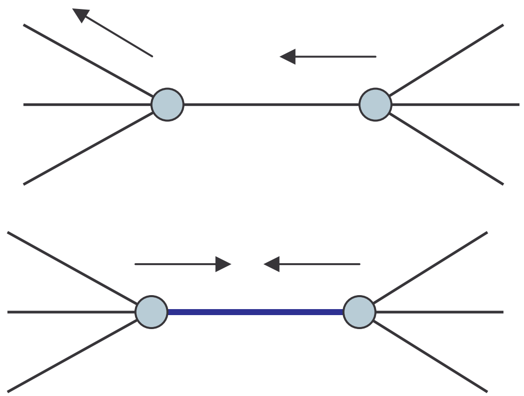
Selecting initial dominating edges take time $O(|V| + |E|)$

When a dominating edge is removed only its distance-2 neighbors can become dominating

Only the heaviest edge incident on v is a candidate



Maintain a pointer for each vertex to the remaining heaviest edge



How to find candidates:

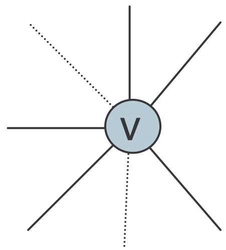
1. Presort edges incident on each vertex, can test candidate in $O(1)$ time,
2. Perform linear search for new candidate incident on v

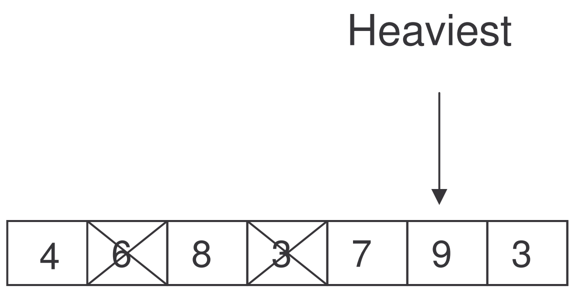
Total time

$O(|V|\, d \log d + |V| + |E|)$

$O(|V|d^2 + |V| + |E|)$

Linear Search for Candidates

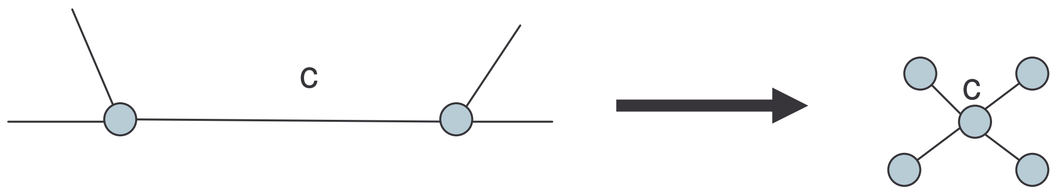




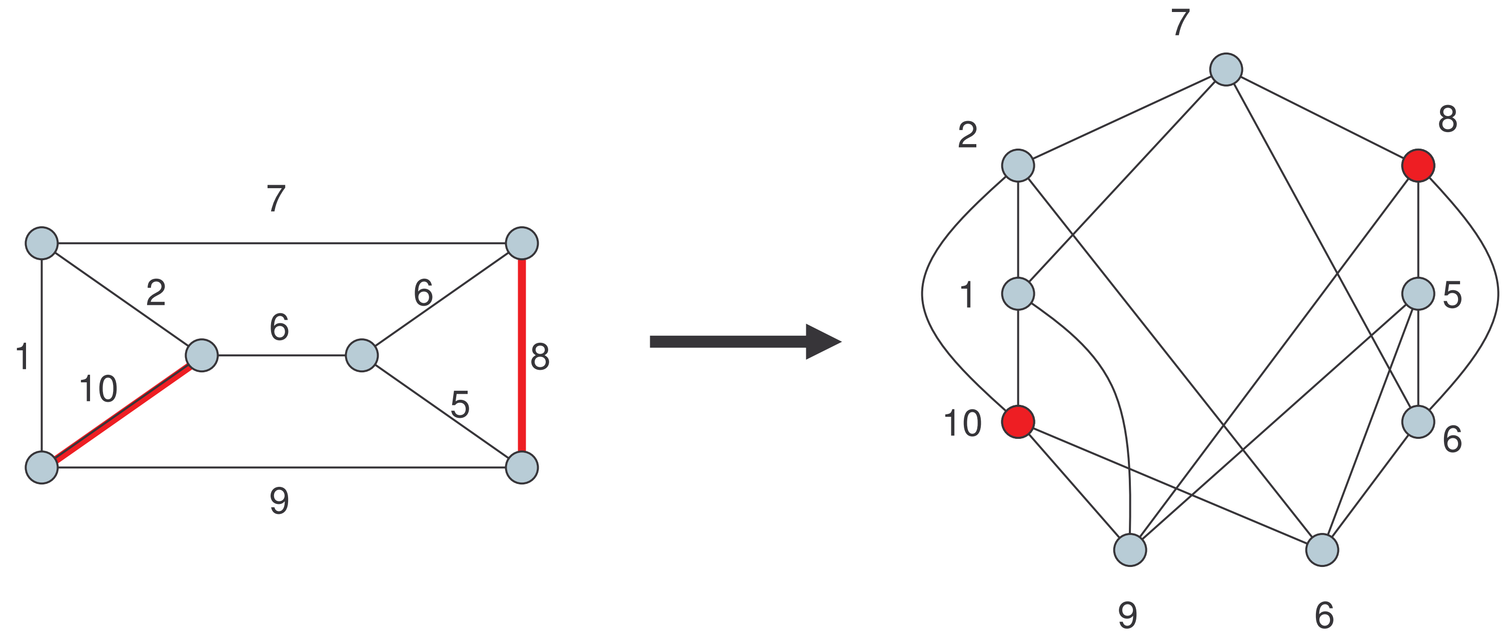
Only compress list when heaviest edge is removed

Observation
If every edge has the same probability of being removed, then the expected time to maintain the pointer for v is $O(d_v)$ (and not $O(d^2)$).

Another View of the Algorithm



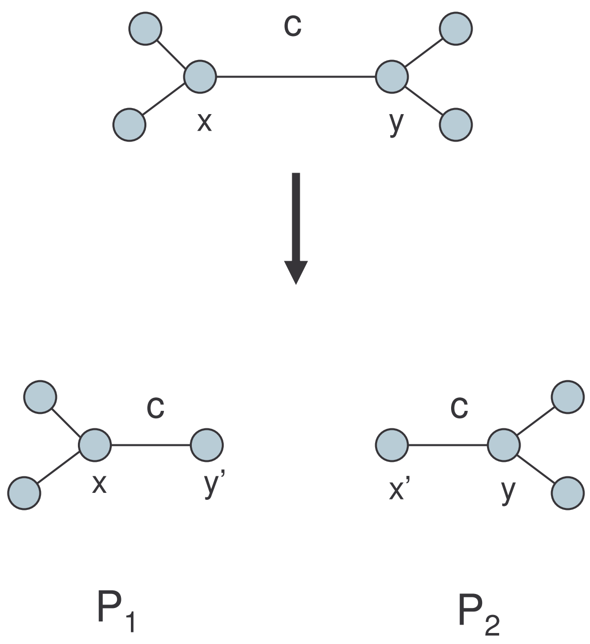
Replace edges with vertices



But this is just Luby's algorithm for finding a maximal independent set of vertices but now run on the edges!

The Parallel Local Domination Algorithm

Data distribution

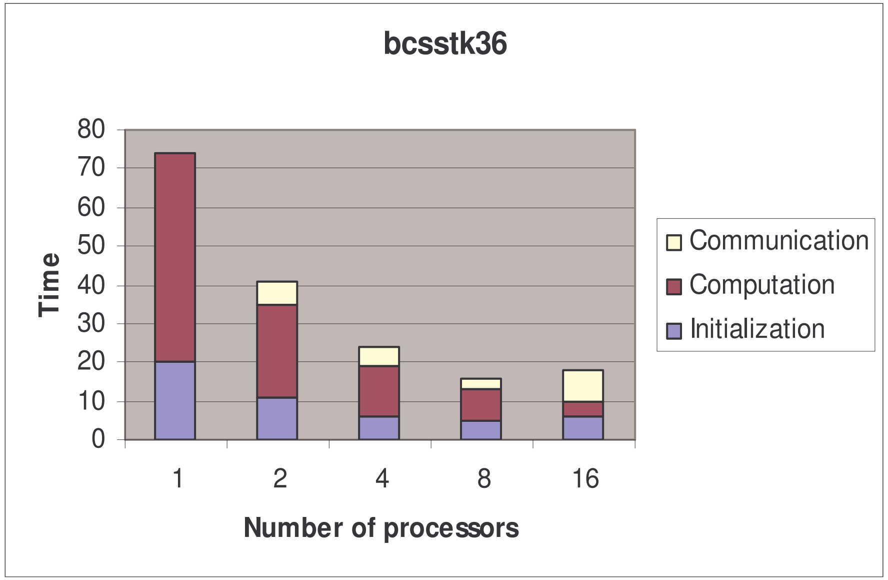


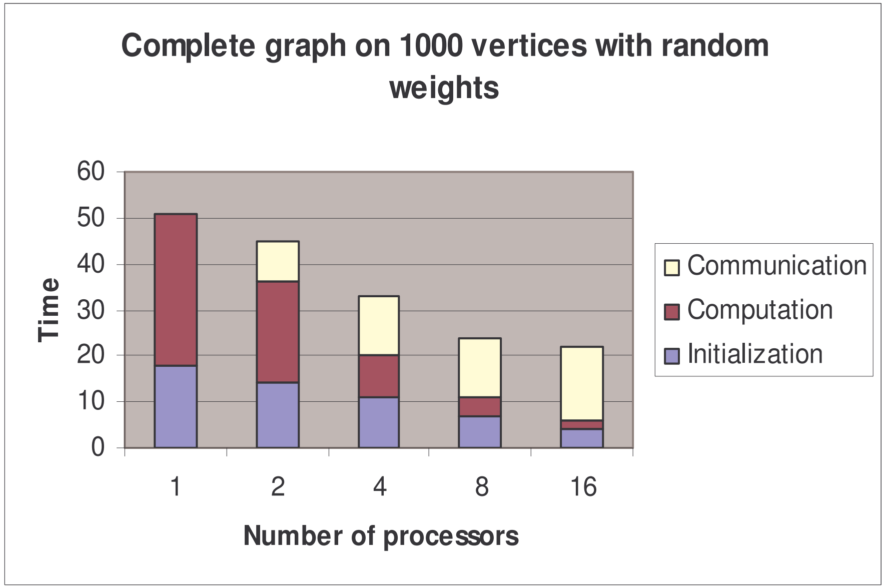
The Algorithm

Partition vertices (and edges) into p subsets
Find initial dominant edges
While some processor has edges left
    Run algorithm locally
    Update neighbor processors

Initial Experiments

Experiments using MPI on IBM Regatta computer





Conclusion

Contributions

- New fast sequential 0.5 approximation algorithm suitable for parallelization
- Parallel implementation showing that it actually scales

Still to do

- Optimize the code
- More experiments
- More rigorous analysis
- Extend algorithm with short augmenting paths

Data

Bcsstk35, n=30237, m = 1450163

P	Total time	Initialization	Computation	Communication
1	74	20	54	0
2	43	11	25	6
4	24	6	13	5
8	16	5	8	3
16	19	6	4	8

Complete random graph on 1000 vertices and 1000000 vertices

1	52	18	33	0
2	45	14	22	9
4	35	11	9	13
8	25	7	4	13
16	24	4	2	16