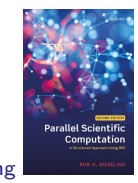
Sequential Graph Matching

Sections 5.1-5.2 of Parallel Scientific Computation, 2nd edition

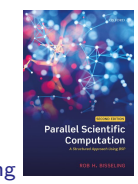
Rob H. Bisseling

Utrecht University



Motivation of graph matching

- Graph matching is the pairing of neighbouring vertices in a graph.
- It has applications in finding
  - suitable partners in online dating services;
  - suitable organ donors in medicine;
  - similar proteins in Protein-Protein Interaction networks from bioinformatics;
  - large pivot elements in matrix computations;
  - similar vertices to be merged in graph coarsening.



Matching can win you a Nobel memorial prize

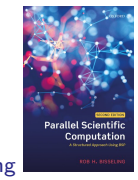
Marriage as an Economic Problem

Lloyd Shapley and Alvin Roth win the Nobel Prize for showing the best way to match people with what they really want.

By Matthew Yglesias | Posted Monday, Oct. 15, 2012, at 1:51 PM ET

The Nobel Prize in economics went to Alvin E. Roth and Lloyd S. Shapley "for the theory of stable allocations and the practice of market design"

Alvin Roth (1951) and Lloyd Shapley (1923–2016).
Source: Slate magazine October 15, 2012.

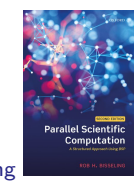


Graph terminology

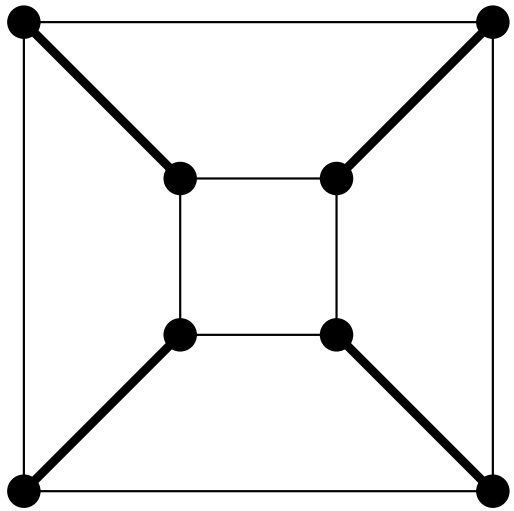
- A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a set $\mathcal{V}$ of vertices (nodes) and a set $\mathcal{E}$ of edges (connections).
- An edge is a pair $e = (u, v)$ with $u, v \in \mathcal{V}$.
- We assume the graph is undirected so that we identify

$$(u, v) \equiv (v, u).$$

- We also assume that the graph is simple, i.e., it has no self-edges (u, u) and there exists at most one edge between the same pair of vertices.
- Furthermore, we assume that every edge $e = (u, v)$ has a weight $\omega(e) = \omega(u, v) > 0$.
- The number of vertices of the graph is $n = |\mathcal{V}|$ and the number of edges is $m = |\mathcal{E}|$.



A matching

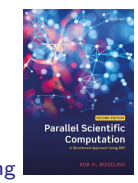


$n = 8,\ m = 12$
4 matched edges
(thick lines)

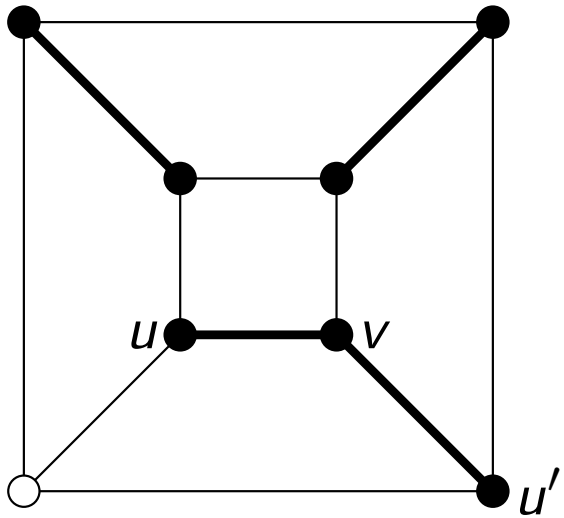
- A matching is a subset $\mathcal{M} \subseteq \mathcal{E}$ such that

$$(u, v), (u', v) \in \mathcal{M} \quad \Rightarrow \quad u = u'.$$

- Thus, no two edges in the matching are incident to the same vertex v.

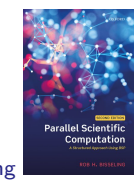


Not a matching

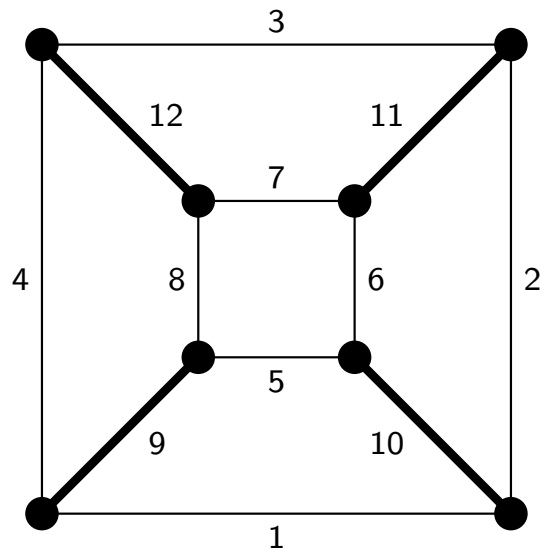


$n = 8,\ m = 12$

- Here, two marked edges $(u, v), (u', v)$ are incident to the same vertex v, so this is not a matching.



Edge-weighted matching

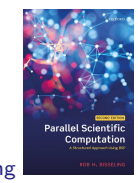


$n = 8,\ m = 12$
$|\mathcal{M}| = 4$
$\omega(\mathcal{M}) = 42$

- The cardinality of a matching $\mathcal{M}$ is $|\mathcal{M}|$.
- The weight of a matching $\mathcal{M}$ is

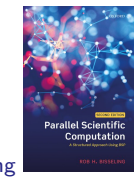
$$\omega(\mathcal{M}) = \sum_{e \in \mathcal{M}} \omega(e).$$

- The edge-weighted maximum matching problem: find an $\mathcal{M}$ with maximum possible weight $\omega(\mathcal{M})$.



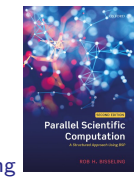
Graph algorithms for social networks

- Facebook has an Open Graph application, which enables clicking a button if you like a webpage.
- Comment of Time Magazine: "Graph: It's a nerdy name for something that's surprisingly simple."



Polynomial time is not good enough

- Finding a maximum-weight matching is possible in polynomial time $\mathcal{O}(mn + n^2 \log n)$ (Gabow 1990).
- The One-World matching problem has 10 billion vertices (people) with 1000 edges (friends) per vertex, i.e., $n = 10^{10}$ and $m = 10^{13}$.
- It takes $\mathcal{O}(10^{23}) = 100\,000\,000$ Pflops to solve this problem to optimality.
- Fugaku, ranked first for the HPCG benchmark on the TOP500 list of supercomputers in June 2022, runs at a speed of 16 Pflop/s solving a sparse linear system.
- Fugaku would take 6 250 000 s or about 72 days to solve the One-World matching problem. That's a lot!
- We need linear-time greedy or approximation algorithms instead of cubic-time exact algorithms.

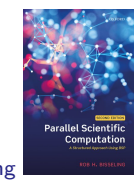


Approximation algorithm

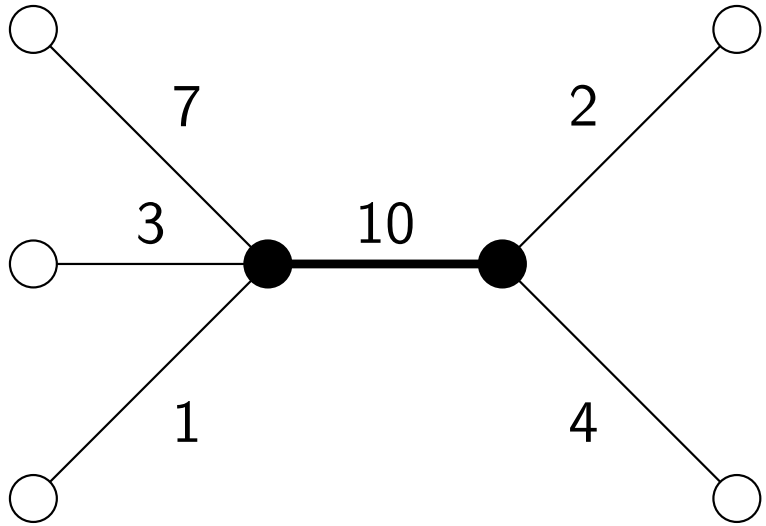
- An approximation algorithm provides a problem solution within reasonable time that differs at most by a guaranteed factor from the optimal solution.
- An α-approximation algorithm for edge-weighted matching gives a matching $\mathcal{M}$ with

$$\omega(\mathcal{M}) \geq \alpha \cdot \omega(\mathcal{M}^*),$$

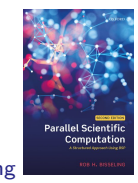
where $\mathcal{M}^*$ denotes a maximum matching.



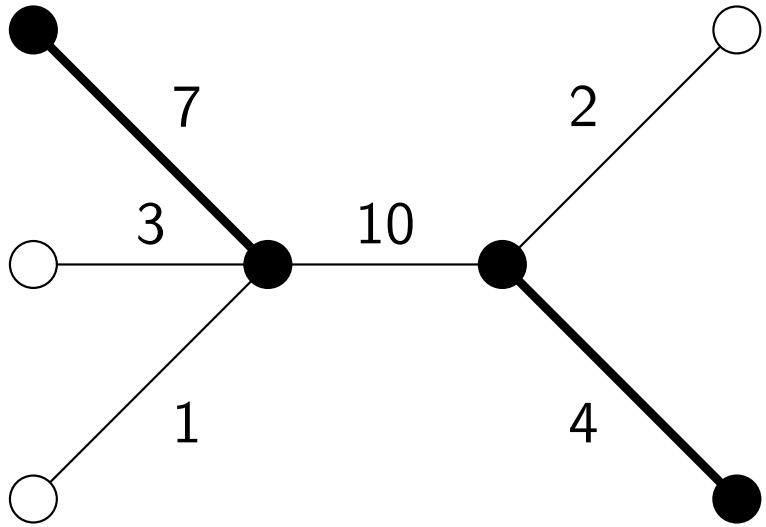
Dominant edge



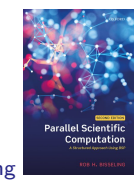
- An edge (u, v) is dominant if for all edges e incident to u or v, we have
$$\omega(u, v) \geq \omega(e).$$
- Here, the edge with weight 10 is dominant.
- Furthermore, the matching $\mathcal{M}$ containing this single edge is maximal, since it cannot be extended.



The maximum matching



- This matching $\mathcal{M}$ with $|\mathcal{M}| = 2$ and $\omega(\mathcal{M}) = 11$ has maximum weight.
- Note the difference between a maximal matching and a maximum matching.

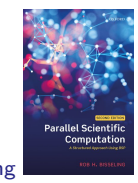


Basic dominant-edge algorithm (Preis 1999)

input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$: graph with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$.
output: $\mathcal{M}$: matching, $\mathcal{M} \subseteq \mathcal{E}$.

$\mathcal{M} := \emptyset$;
while $\mathcal{E} \neq \emptyset$ **do**
  pick a dominant edge $(u, v) \in \mathcal{E}$;
  $\mathcal{M} := \mathcal{M} \cup \{(u, v)\}$;
  $\mathcal{E} := \mathcal{E} \setminus \{(x, y) \in \mathcal{E} : x = u \vee x = v\}$;
  $\mathcal{V} := \mathcal{V} \setminus \{u, v\}$;
return $\mathcal{M}$;

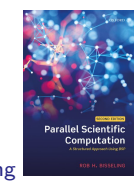
R. Preis, in *Proceedings STACS 1999*, Lecture Notes in Computer Science, Vol. 1563, pp. 259–269. Springer.



Lemma 5.1: the basic dominant-edge algorithm yields a maximal matching

Proof:

- Let $\mathcal{M}$ be the matching produced by the basic dominant-edge algorithm and $\mathcal{E}$ the original edge set.
- An edge $e \in \mathcal{E}$ to be added to $\mathcal{M}$ in an extension is not in $\mathcal{M}$. Hence it must have been removed by an edge $(u, v) \in \mathcal{M}$ sometime during the algorithm.
- Then e must be incident to u or v and hence cannot be in the same matching as (u, v).
- Thus, the matching $\mathcal{M}$ cannot be extended. □

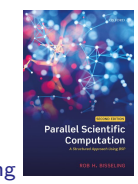


Lemma 5.2: the algorithm yields a $\frac{1}{2}$-approximation

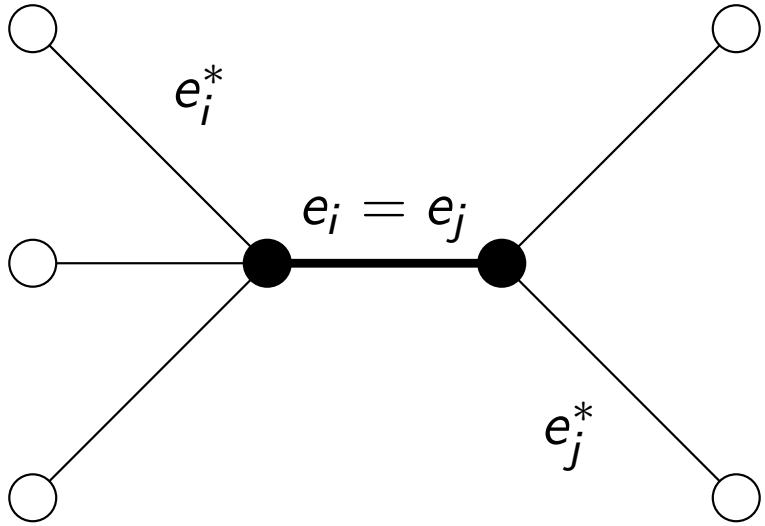
Proof:

- Let $\mathcal{M}$ be the matching produced by the basic dominant-edge algorithm.
- Let $\mathcal{M}^* = \{e_0^*, \ldots, e_{k-1}^*\}$ be a maximum matching.
- For each edge $e_i^* \in \mathcal{M}^*$:
  - if $e_i^* \in \mathcal{M}$, we define $e_i = e_i^*$;
  - if $e_i^* \notin \mathcal{M}$, we define $e_i =$ the edge that removed e_i^* during the algorithm.
- This creates a list $e_0, \ldots, e_{k-1}$ of edges from $\mathcal{M}$, with

$$\omega(e_i) \geq \omega(e_i^*), \quad \text{for } 0 \leq i < k.$$

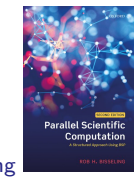


Proof (cont'd)



- Note that $e_i = e_j$ for $i \neq j$ is possible, since e_i^* and e_j^* may have been removed by the same edge in $\mathcal{M}$.
- However, at most 2 edges from $\mathcal{M}^*$ may have been removed by the same edge and these must be at opposite ends of the removing edge.
- Therefore, an edge from $\mathcal{M}$ occurs at most twice in the list of the e_i, so that

$$\sum_{i=0}^{k-1} \omega(e_i) \leq 2\omega(\mathcal{M}).$$



Proof (cont'd)

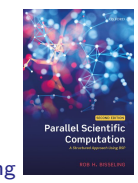
- Combining our inequalities gives

$$\omega(\mathcal{M}^*) = \sum_{i=0}^{k-1} \omega(e_i^*) \leq \sum_{i=0}^{k-1} \omega(e_i) \leq 2\omega(\mathcal{M}).$$

  which is equivalent to

$$\omega(\mathcal{M}) \geq \frac{1}{2}\omega(\mathcal{M}^*).$$

□



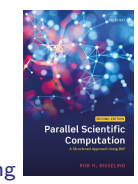
Local domination algorithm: initialization

$D := \emptyset$; ▷ dominant vertices
$\mathcal{M} := \emptyset$; ▷ matched edges
for all $v \in \mathcal{V}$ **do**
    $pref(v) :=$ **nil**; ▷ preference of v

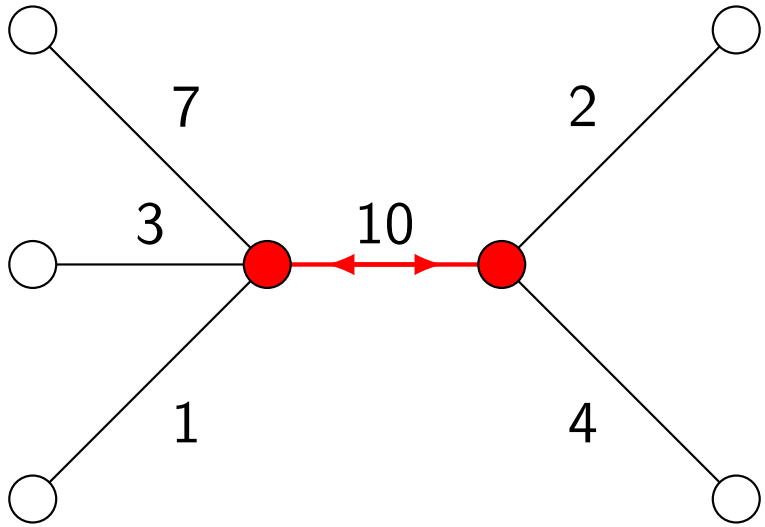
{ Find initial dominant edges }
for all $v \in \mathcal{V}$ **do**
    $Adj_v := \{u \in \mathcal{V} : (u, v) \in \mathcal{E}\}$; ▷ adjacent vertices of v
    if $Adj_v \neq \emptyset$ **then**
        $pref(v) := \text{argmax}\{\omega(u, v) \ : \ u \in Adj_v\}$;
        if $pref(pref(v)) = v$ **then**
            $D := D \cup \{v, pref(v)\}$;
            $\mathcal{M} := \mathcal{M} \cup \{(v, pref(v))\}$;

...

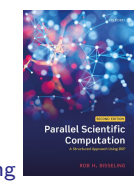
F. Manne and R. H. Bisseling, in *Proceedings PPAM 2007*, Lecture Notes in Computer Science, Vol. 4967, pp. 708–717. Springer.



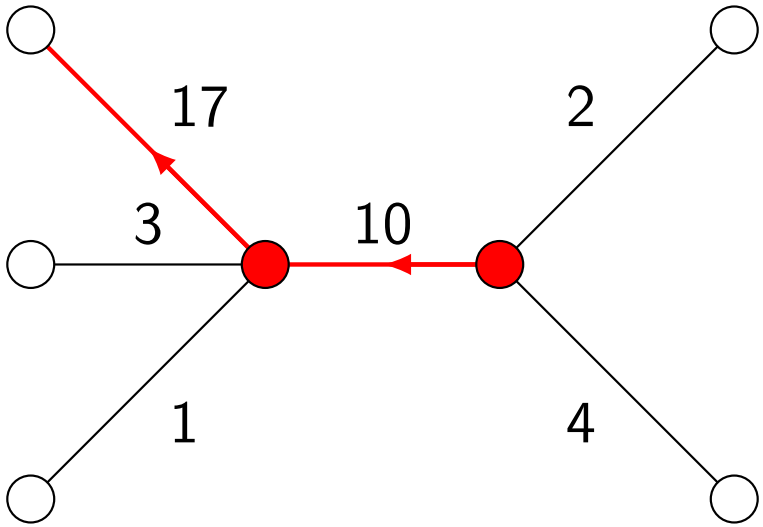
Mutual preferences

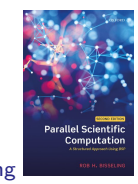


- A mutual preference corresponds to a dominant edge.
- For simplicity, we assume that there are no equal preferences (ties).



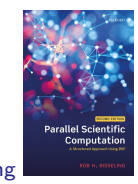
Nonmutual preferences





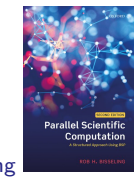
Local domination algorithm: main loop

...
{ Process matched vertices }
while $D \neq \emptyset$ **do**
    pick a vertex $v \in D$;
    $D := D \setminus \{v\}$;
    for all $x \in Adj_v \ : \ (x, pref(x)) \notin \mathcal{M}$ **do**
        $Adj_x := Adj_x \setminus \{v\}$;
        if $Adj_x \neq \emptyset$ **then**
            $pref(x) := \mathrm{argmax}\{\omega(x, y) \ : \ y \in Adj_x\}$;
            if $pref(pref(x)) = x$ **then**
                $D := D \cup \{x, pref(x)\}$;
                $\mathcal{M} := \mathcal{M} \cup \{(x, pref(x))\}$;
        else
            $pref(x) :=$ **nil**;



Properties of the local domination algorithm

- As long as we can add an edge to the matching, there exists a dominant edge, the heaviest remaining edge.
- The algorithm keeps going until the set of dominant vertices D becomes empty. Then the matching $\mathcal{M}$ is maximal.
- For now, we assume without loss of generality that the weights are unique. This guarantees that there are no ties.
- Dominance is a local property: the algorithm is easy to parallelize.



Summary

- A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a set $\mathcal{V}$ of vertices and a set $\mathcal{E}$ of edges.
- A matching in an undirected graph $\mathcal{G}$ is a subset $\mathcal{M} \subseteq \mathcal{E}$ such that no two edges in the subset share a vertex.
- An α-approximation algorithm for edge-weighted matching gives a matching $\mathcal{M}$ with

$$\omega(\mathcal{M}) \geq \alpha \cdot \omega(\mathcal{M}^*),$$

  where $\mathcal{M}^*$ denotes a maximum matching.
- The local domination algorithm is a parallelizable $\frac{1}{2}$-approximation algorithm for edge-weighted matching.