Big graphs for big data: parallel matching and clustering on billion-vertex graphs

Rob H. Bisseling

Mathematical Institute, Utrecht University

Collaborators: Bas Fagginger Auer, Fredrik Manne, Albert-Jan Yzelman

Asia-trip A-Eskwadraat, July 2014

Universiteit Utrecht

Graph Matching

- Introduction
- Greedy algorithm
- Parallelisable 1/2-approximation algorithm
- BSP algorithm
- GPU algorithm
- Results

Clustering

- Introduction
- Sequential algorithm
- GPU algorithm
- Results

Conclusion

Matching can win you a Nobel 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"

Source: Slate magazine October 15, 2012

Universiteit Utrecht

Motivation of graph matching

- Graph matching is a pairing of neighbouring vertices.
- It has applications in
  - medicine: finding suitable donors for organs
  - social networks: finding partners
  - scientific computing: finding pivot elements in matrix computations
  - graph coarsening: making the graph smaller by merging similar vertices

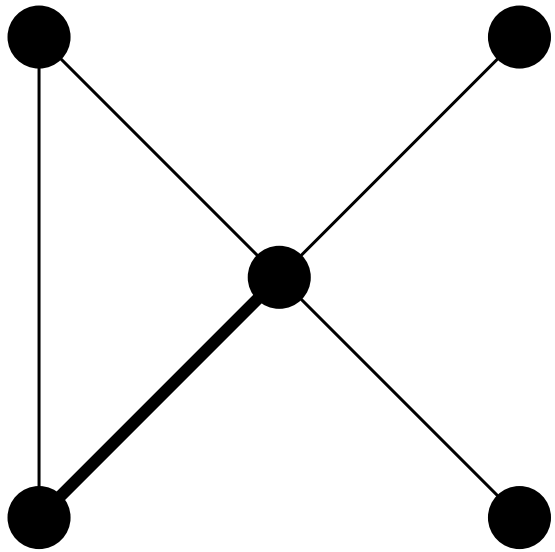
Motivation of greedy/approximation graph matching

- Optimal solution is possible in polynomial time.
- Time for weighted matching in graph $G = (V, E)$ is $\mathcal{O}(mn + n^2 \log n)$ with $n = |V|$ the number of vertices, and $m = |E|$ the number of edges (Gabow 1990).
- The aim is a billion vertices, $n = 10^9$, with 100 edges per vertex, i.e. $m = 10^{11}$.
- Thus, a time of $\mathcal{O}(10^{20}) = 100,000$ Petaflop units is far too long. Fastest supercomputer today, the Chinese Tianhe-2 (Milky-Way 2), performs 33.8 Petaflop/s.
- We need linear-time greedy or approximation algorithms.

Formal definition of graph matching

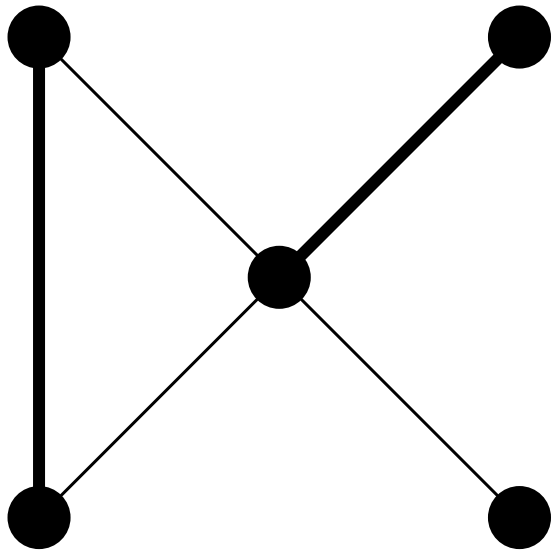
- A graph is a pair $G = (V, E)$ with vertices V and edges E.
- All edges $e \in E$ are of the form $e = (v, w)$ for vertices $v, w \in V$.
- A matching is a collection $M \subseteq E$ of disjoint edges.
- Here, the graph is undirected, so $(v, w) = (w, v)$.

Maximal matching



- A matching is maximal if we cannot enlarge it further by adding another edge to it.

Maximum matching



- A matching is maximum if it possesses the largest possible number of edges, compared to all other matchings.

Edge-weighted matching

- If the edges are provided with weights $\omega : E \to \mathbb{R}_{>0}$, finding a matching M which maximises

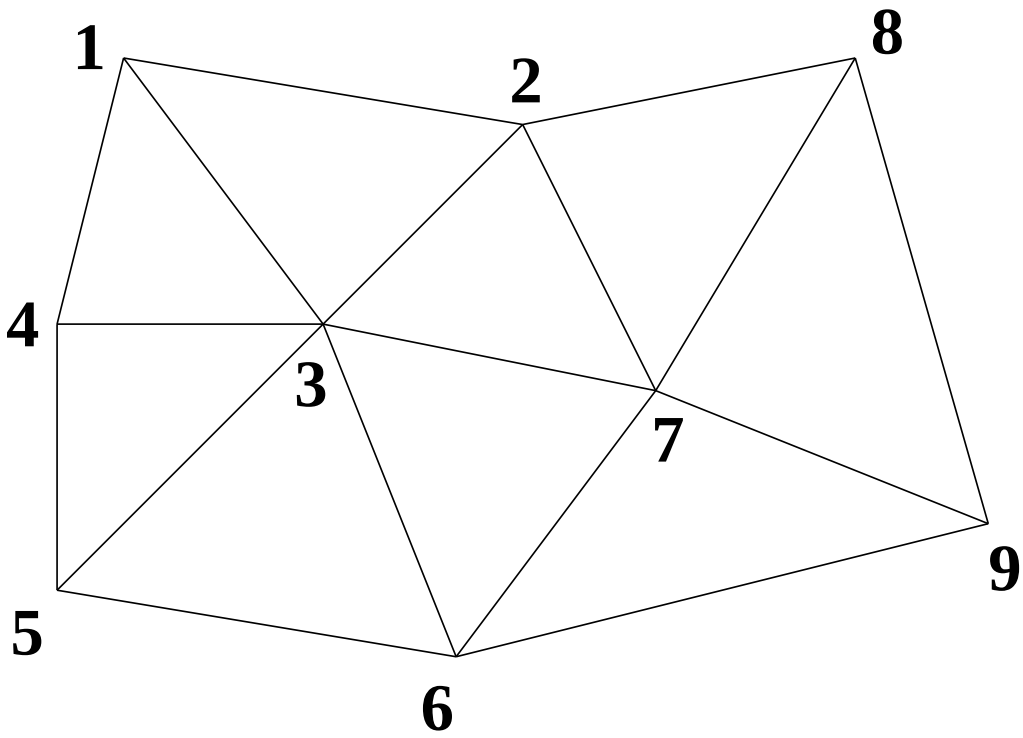
$$\omega(M) = \sum_{e \in M} \omega(e),$$

  is called edge-weighted matching.
- Greedy matching provides us with maximal matchings, but not necessarily with maximum possible weight.

Sequential greedy matching

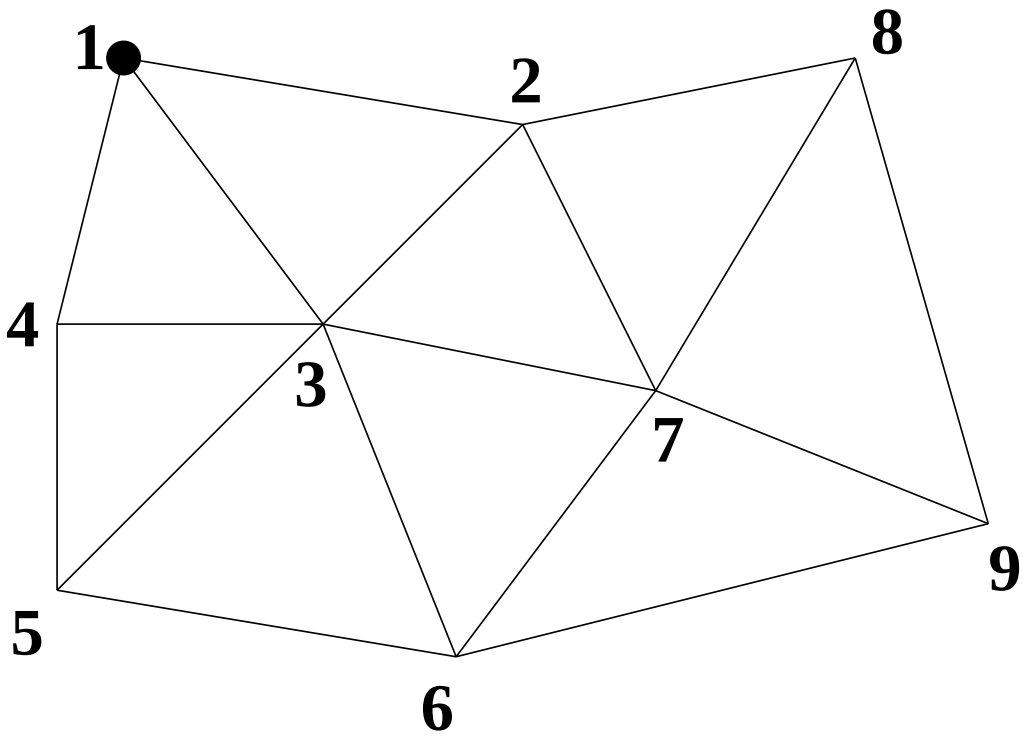
- In random order, vertices $v \in V$ select and match neighbours one-by-one.
- Here, we can pick
  - the first available neighbour w of v, greedy random matching
  - the neighbour w with maximum $\omega(v, w)$, greedy weighted matching
- Or: we sort all the edges by weight, and successively match the vertices v and w of the heaviest available edge (v, w). This is commonly called greedy matching.

Sequential greedy random matching



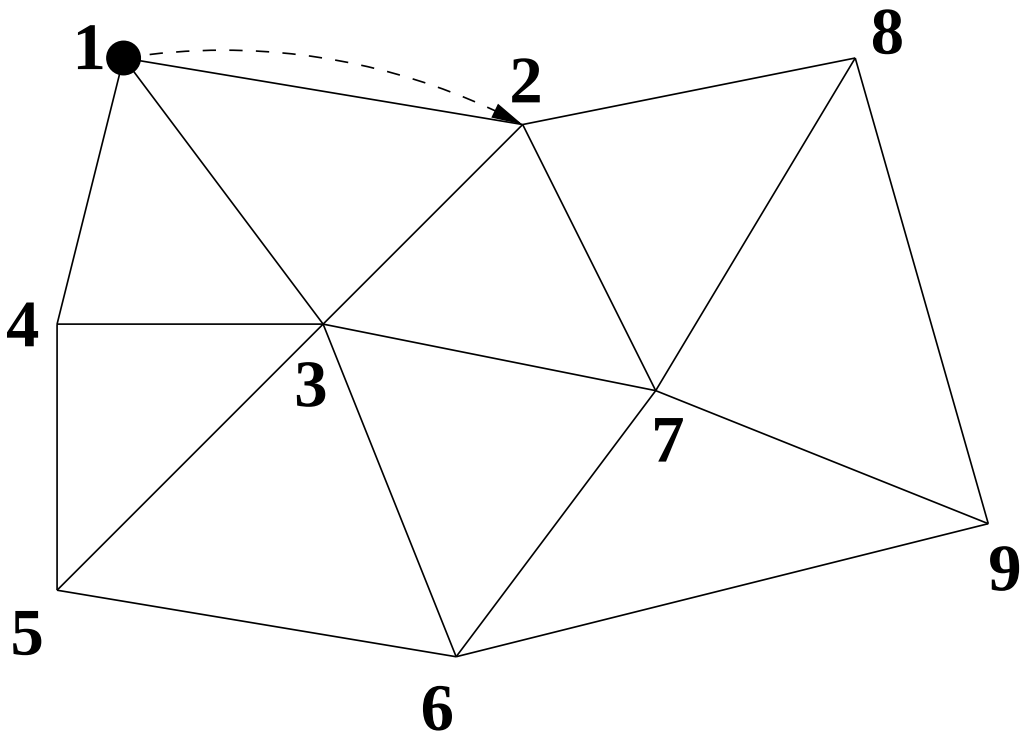
Universiteit Utrecht

Sequential greedy random matching



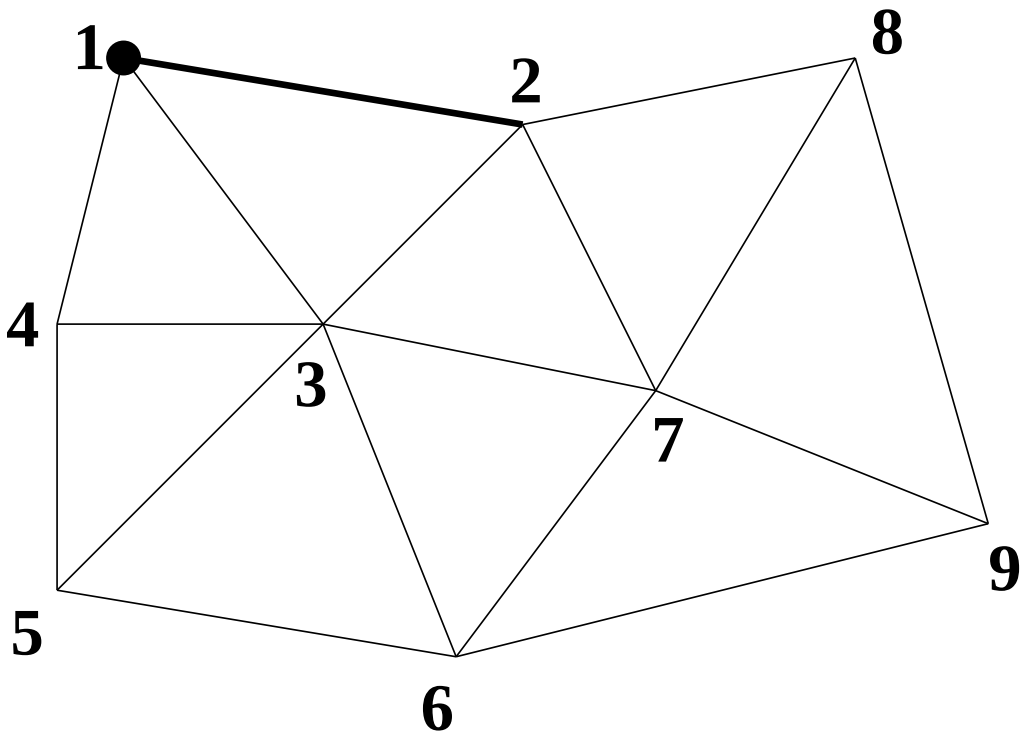
Universiteit Utrecht

Sequential greedy random matching



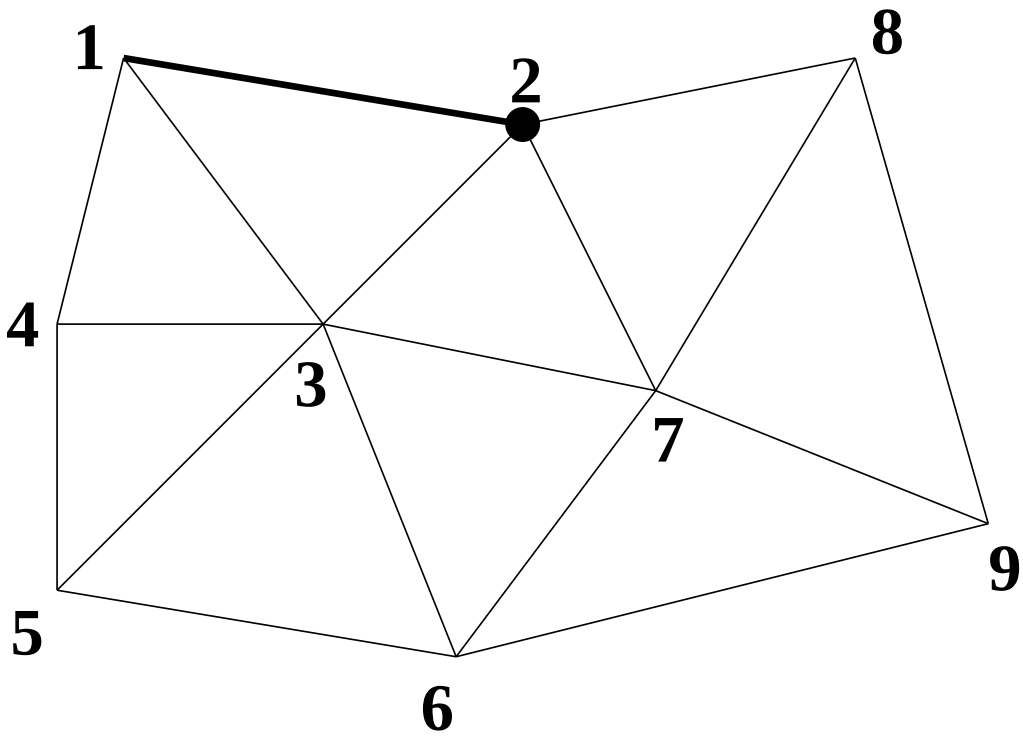
Universiteit Utrecht

Sequential greedy random matching



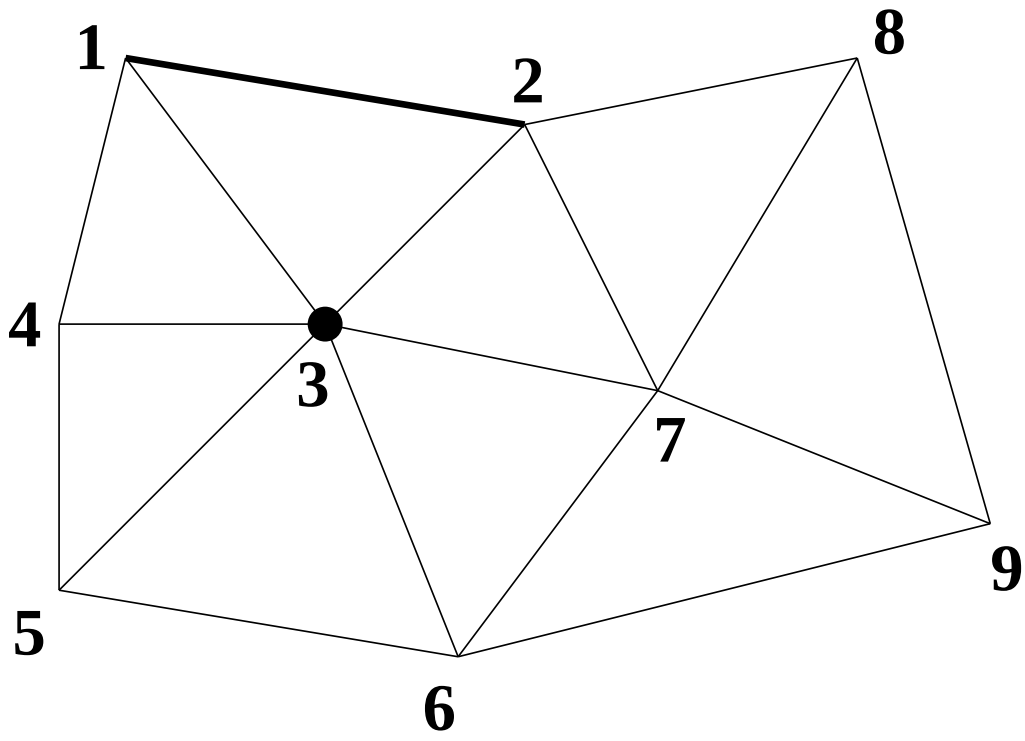
Universiteit Utrecht

Sequential greedy random matching



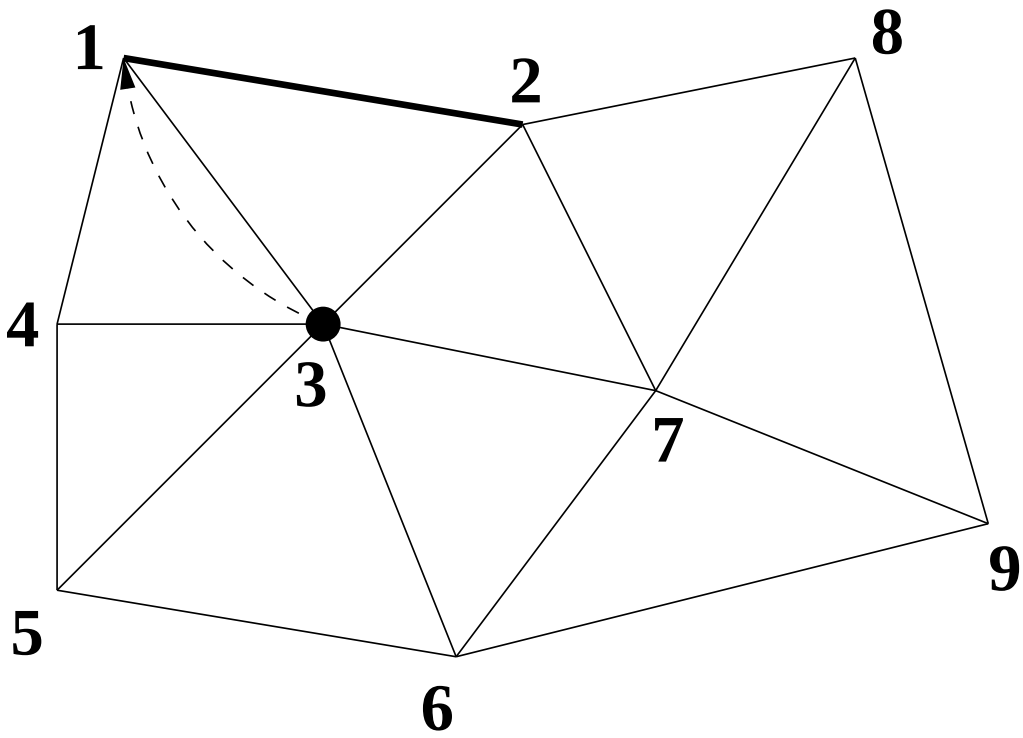
Universiteit Utrecht

Sequential greedy random matching



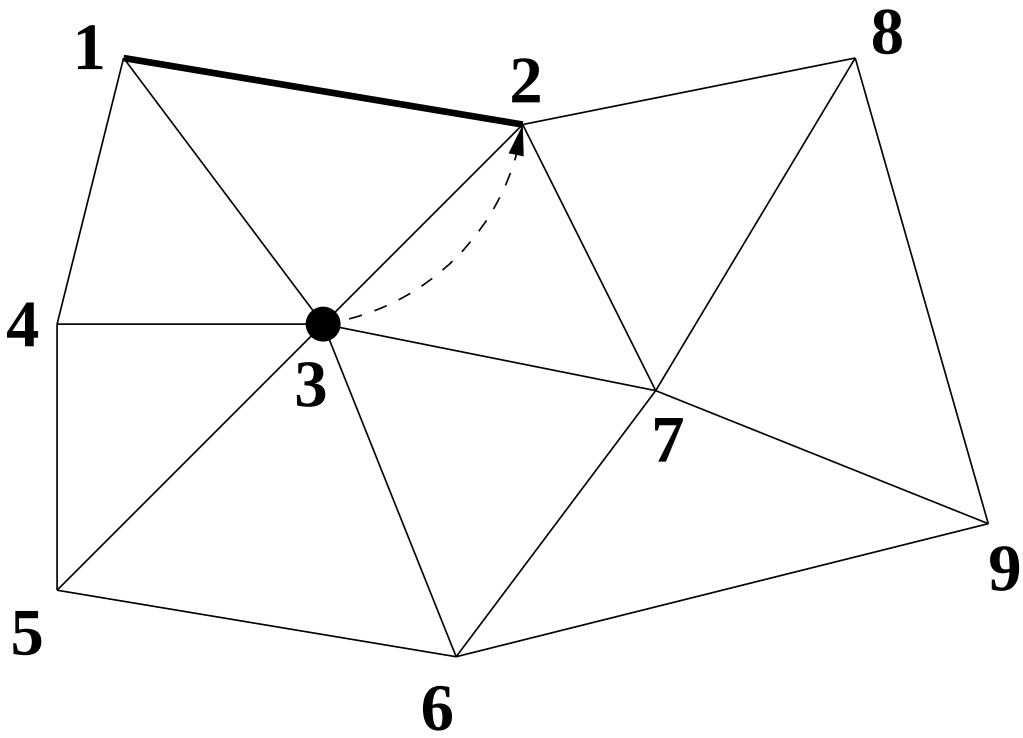
Universiteit Utrecht

Sequential greedy random matching



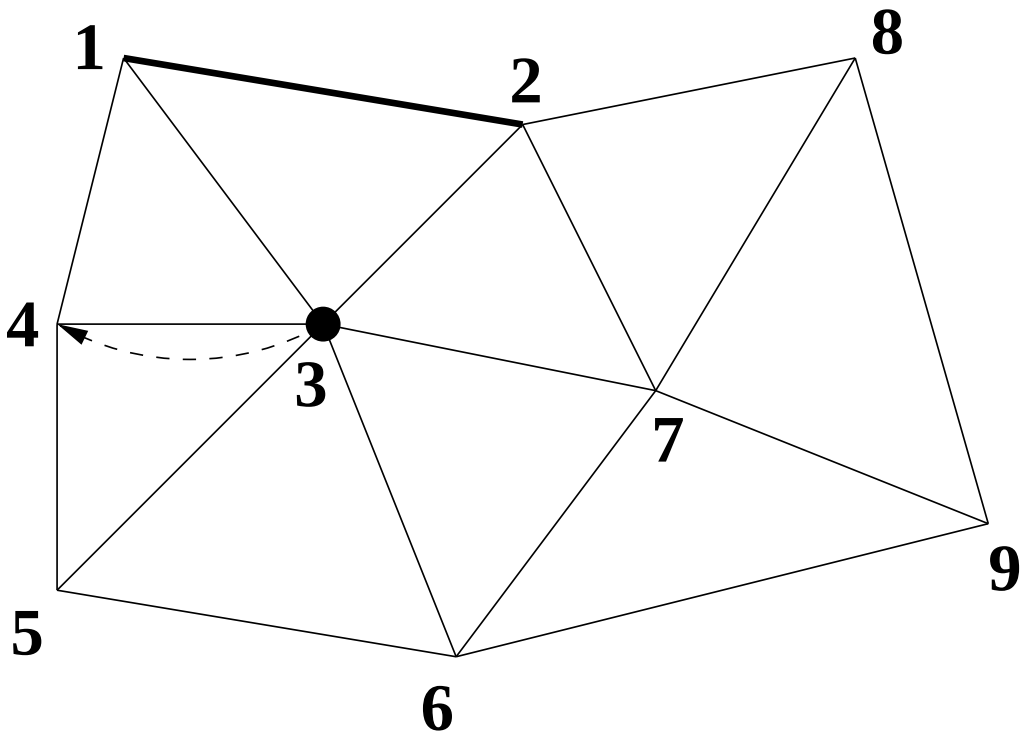
Universiteit Utrecht

Sequential greedy random matching



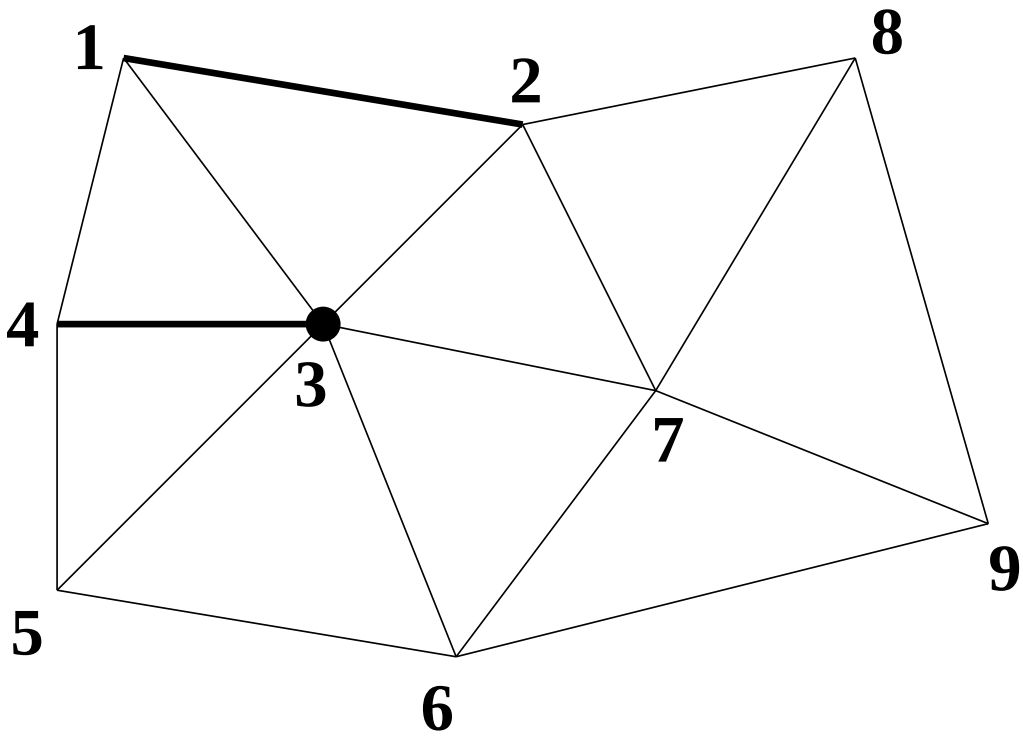
Universiteit Utrecht

Sequential greedy random matching



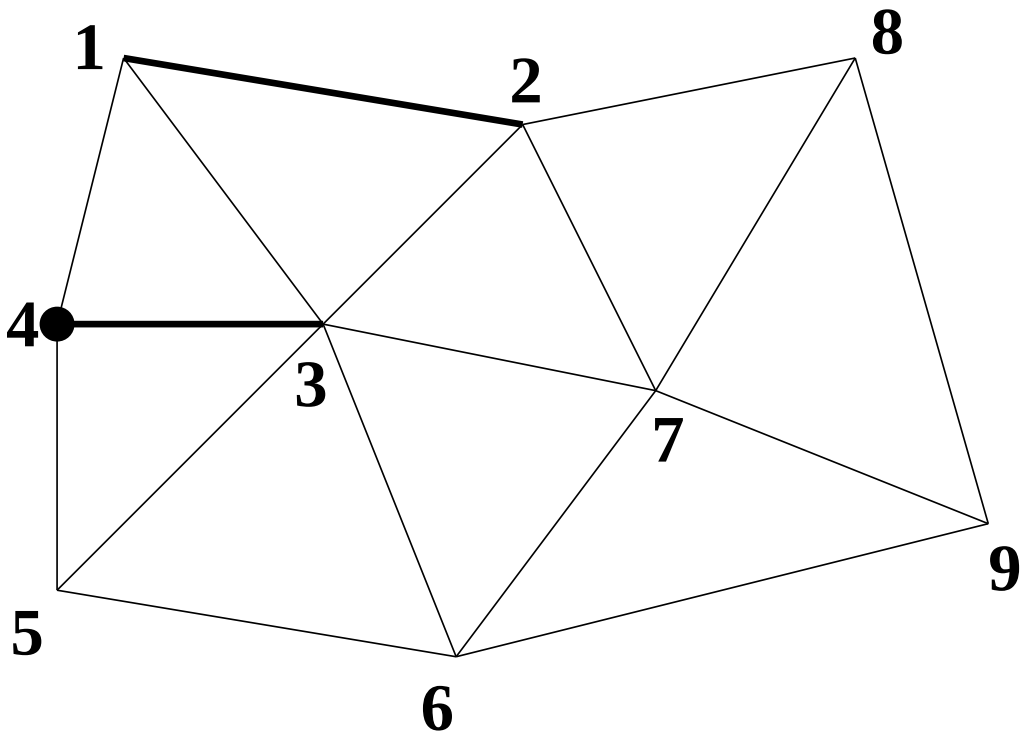
Universiteit Utrecht

Sequential greedy random matching



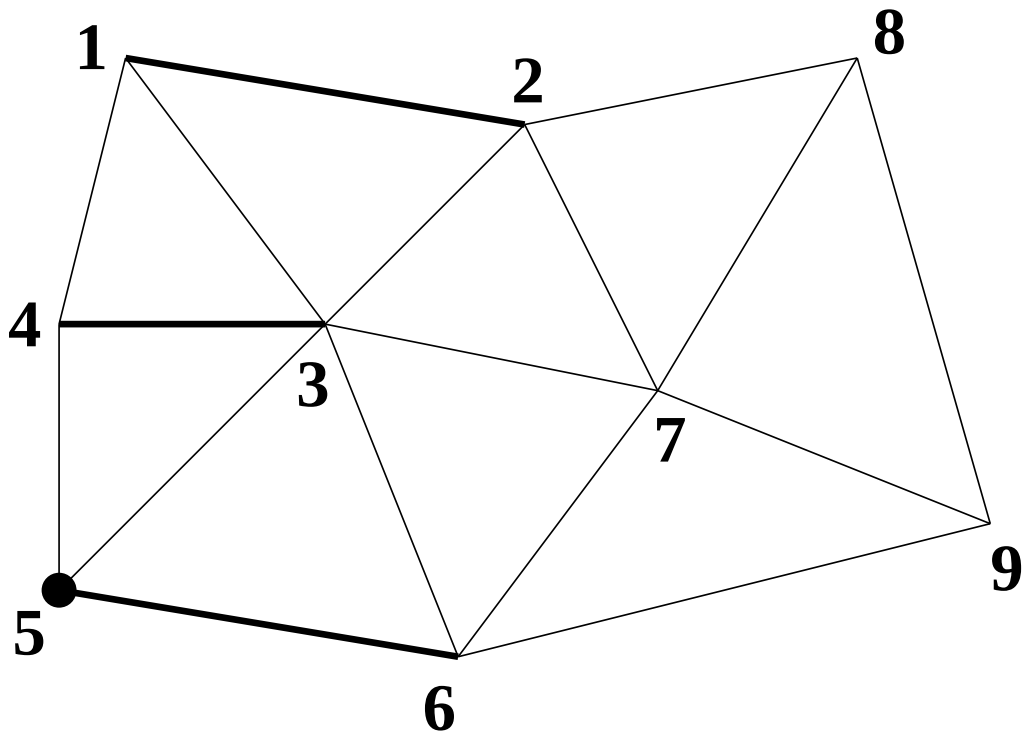
Universiteit Utrecht

Sequential greedy random matching



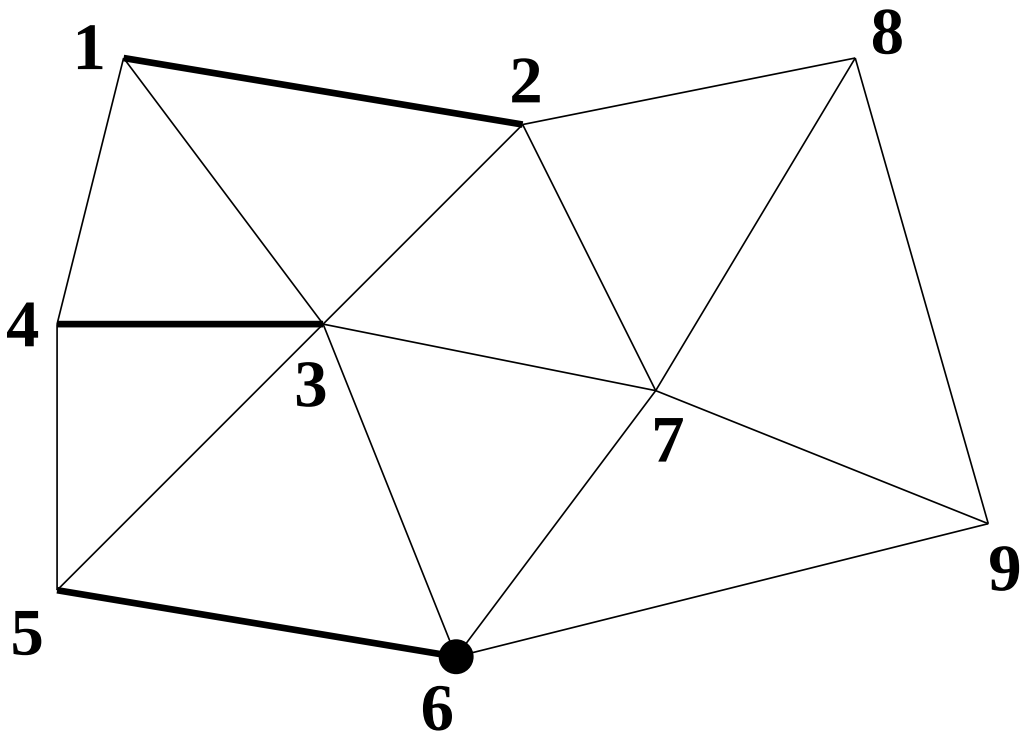
Universiteit Utrecht

Sequential greedy random matching



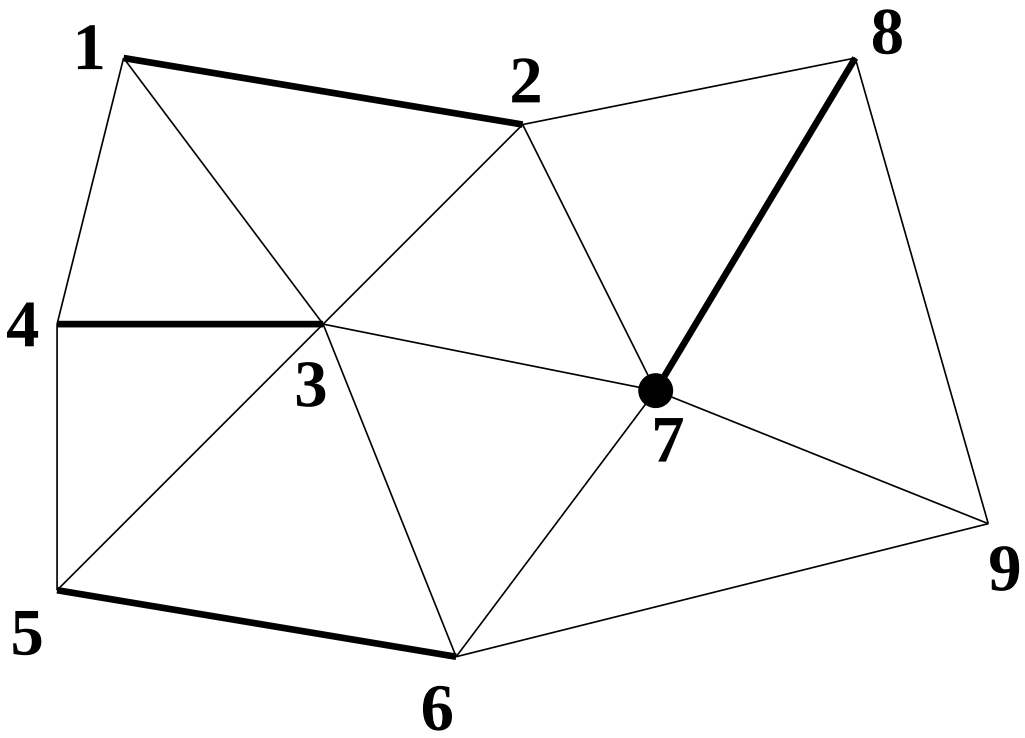
Universiteit Utrecht

Sequential greedy random matching



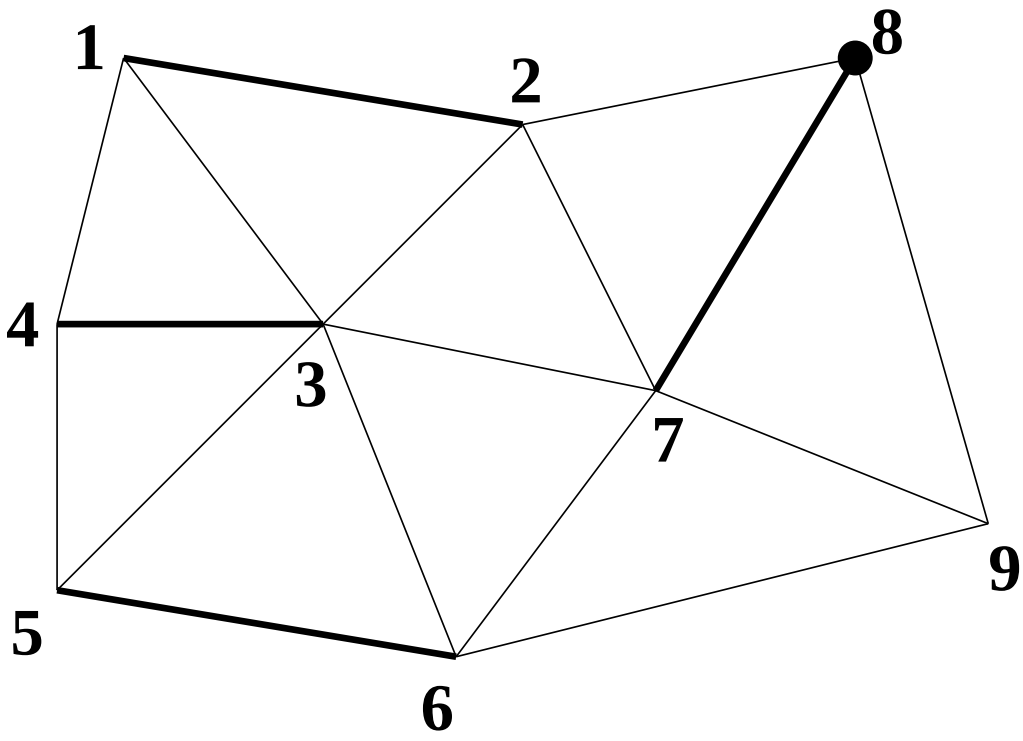
Universiteit Utrecht

Sequential greedy random matching



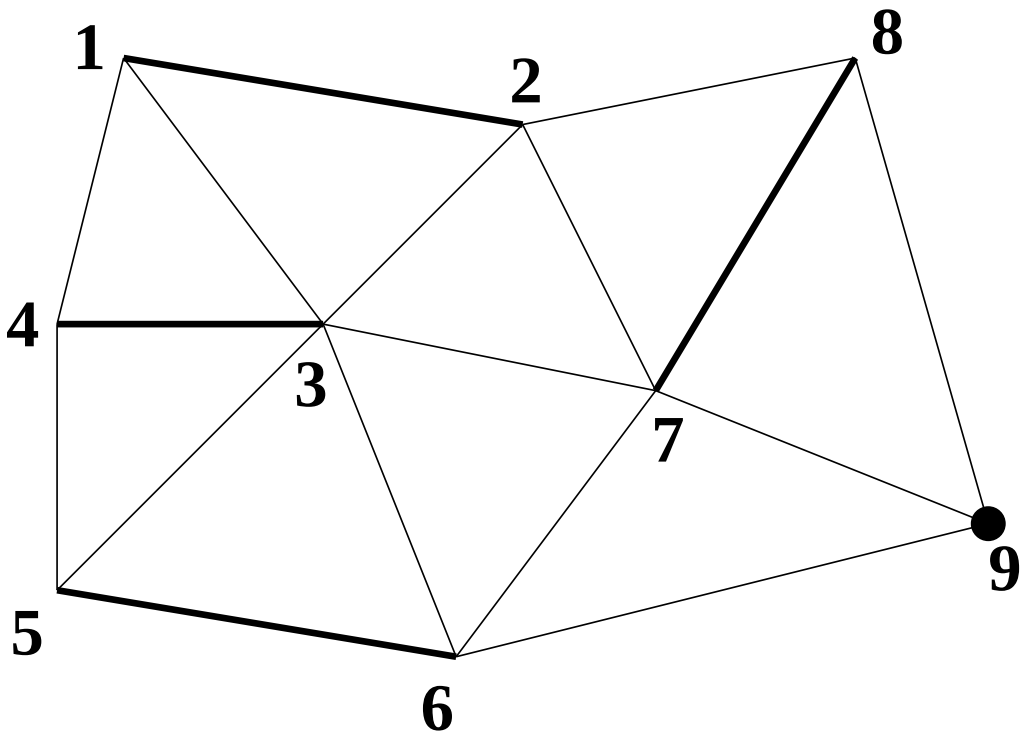
Universiteit Utrecht

Sequential greedy random matching



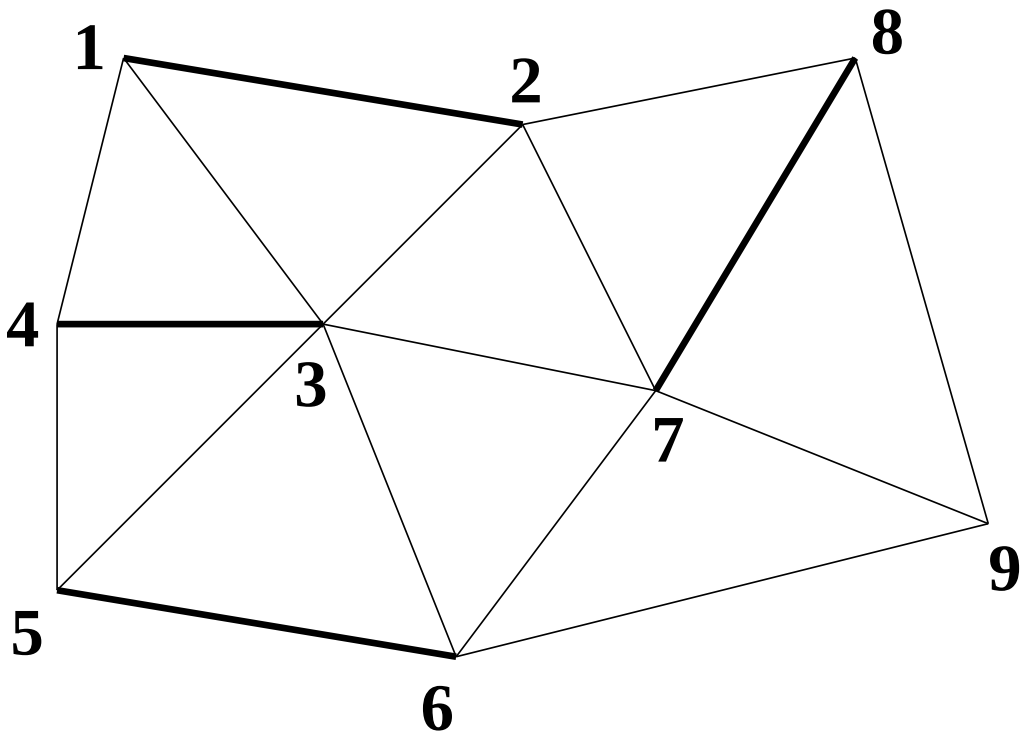
Universiteit Utrecht

Sequential greedy random matching



Universiteit Utrecht

Sequential greedy random matching

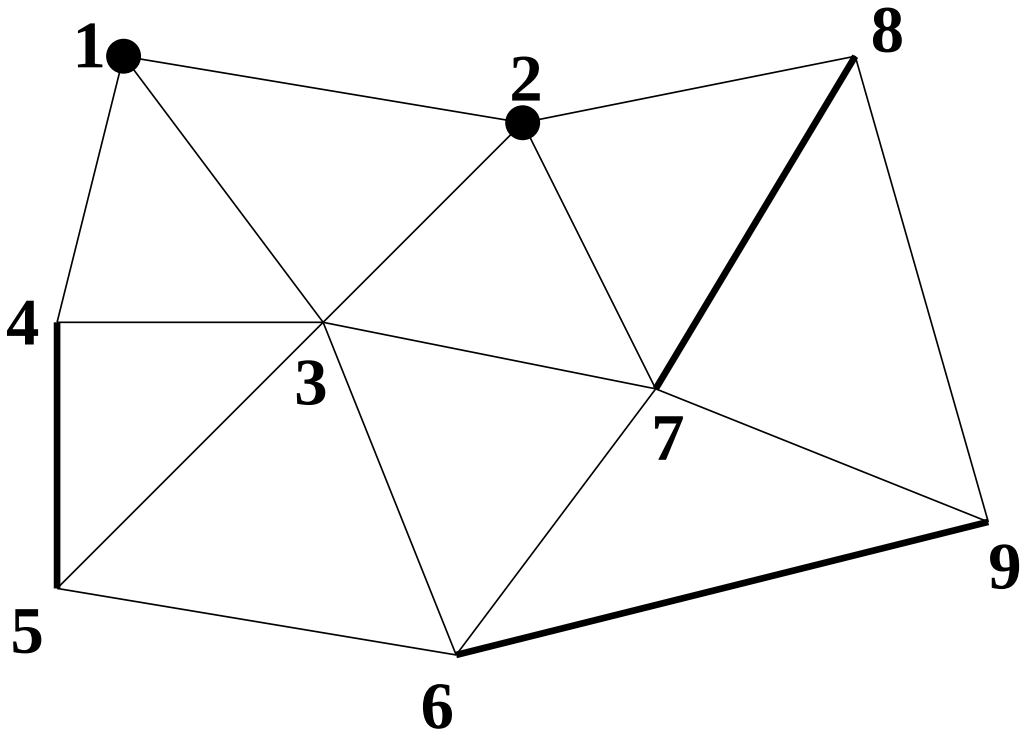


Universiteit Utrecht

Greedy matching is a 1/2-approximation algorithm

- Weight $\omega(M) \geq \omega_{\text{optimal}}/2$
- Cardinality $|M| \geq |M_{\text{card}-\max}|/2$, because M is maximal.
- Time complexity is $\mathcal{O}(m \log m)$, because all edges must be sorted.

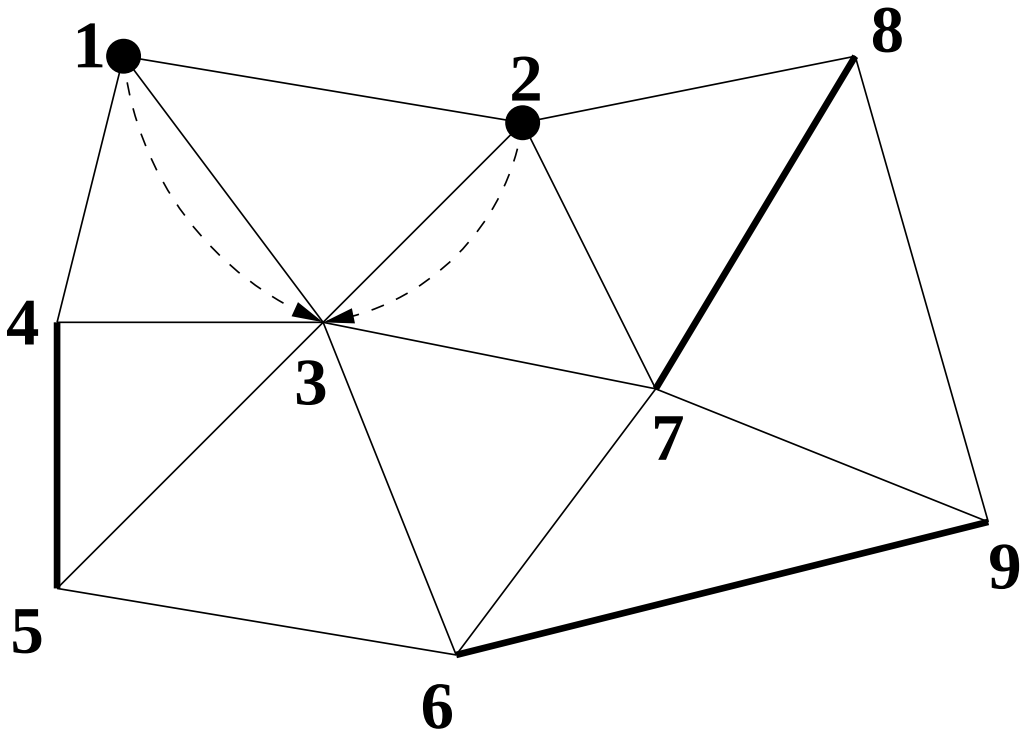
Parallel greedy matching: trouble



Suppose we match vertices simultaneously.

Universiteit Utrecht

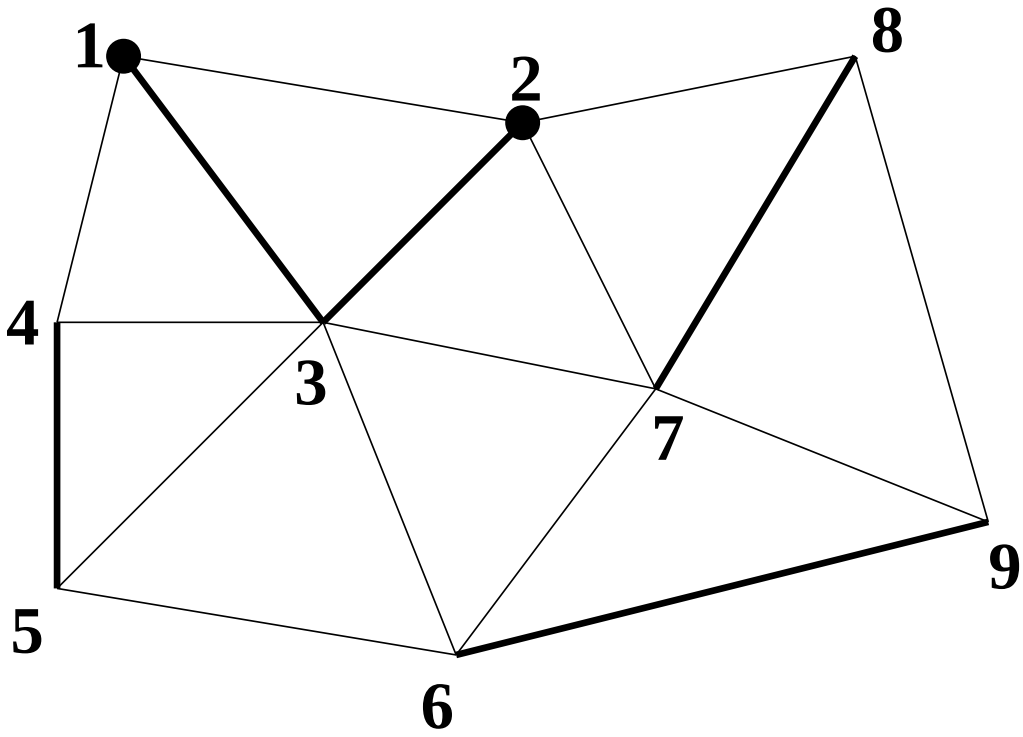
Parallel greedy matching: trouble



Two vertices each find an unmatched neighbour...

Universiteit Utrecht

Parallel greedy matching: trouble



. . . but generate an invalid matching.

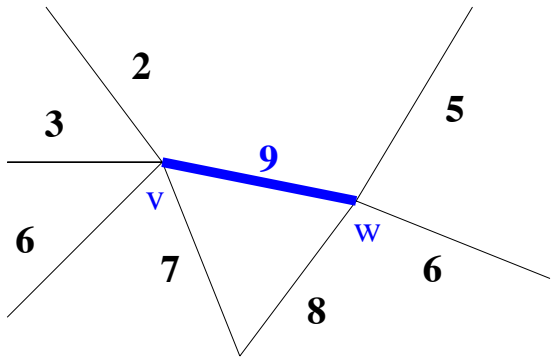
Universiteit Utrecht

Parallelisable dominant-edge algorithm

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

- An edge $(v, w) \in E$ is dominant if

$$\omega(v, w) = \max\{\omega(x, y) : (x, y) \in E \wedge (x = v \vee x = w)\}$$



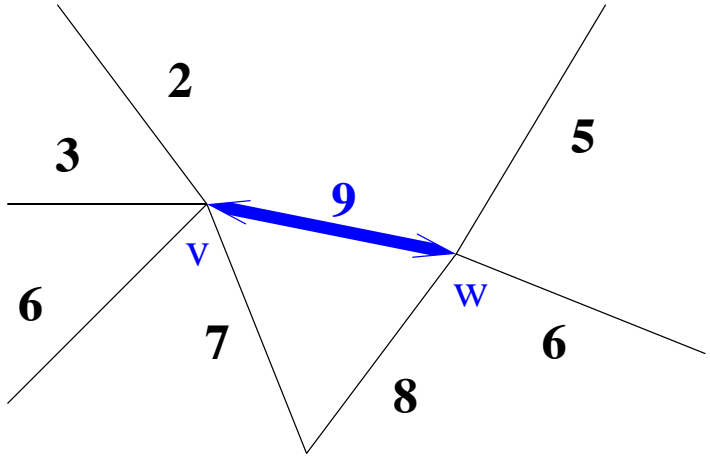
Universiteit Utrecht

Sequential approximation algorithm: initialisation

function SeqMatching(V, E)
  for all $v \in V$ **do**
    $pref(v) = null$
  $D := \emptyset$
  $M := \emptyset$

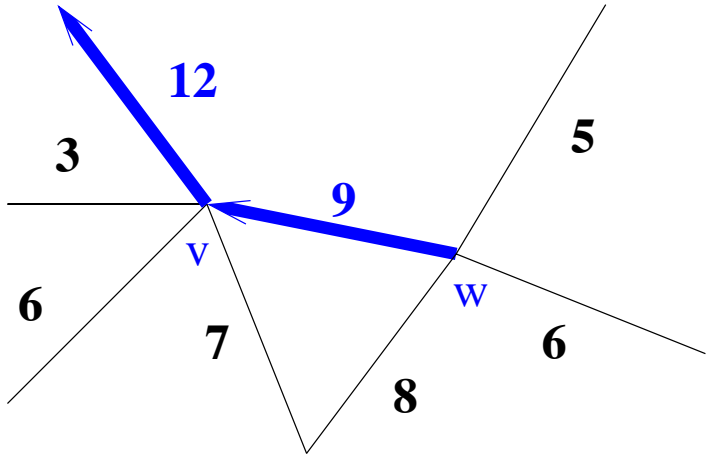
  { Find dominant edges }
  for all $v \in V$ **do**
    $Adj_v := \{w \in V : (v, w) \in E\}$
    $pref(v) := \text{argmax}\{\omega(v, w) \ : \ w \in Adj_v\}$
    if $pref(pref(v)) = v$ **then**
      $D := D \cup \{v, pref(v)\}$
      $M := M \cup \{(v, pref(v))\}$

Mutual preferences



Universiteit Utrecht

Non-mutual preferences



Universiteit Utrecht

Sequential approximation algorithm: main loop

while $D \neq \emptyset$ **do**
  pick $v \in D$
  $D := D \setminus \{v\}$
  for all $x \in Adj_v \setminus \{pref(v)\} \ : \ (x, pref(x)) \notin M$ **do**
    $Adj_x := Adj_x \setminus \{v\}$

    { Set new preference }
    $pref(x) := \mathrm{argmax}\{\omega(x, w) \ : \ w \in Adj_x\}$
    if $pref(pref(x)) = x$ **then**
      $D := D \cup \{x, pref(x)\}$
      $M := M \cup \{(x, pref(x))\}$
return M

Properties of the dominant-edge algorithm

- Dominant-edge algorithm is a 1/2-approximation:

$$\omega(M) \geq \omega_{\text{optimal}}/2$$

- Dominance is a local property: easy to parallelise.
- Algorithm keeps going until set of dominant vertices D is empty and matching M is maximal.
- Assumption without loss of generality: weights are unique. Otherwise, use vertex numbering to break ties.

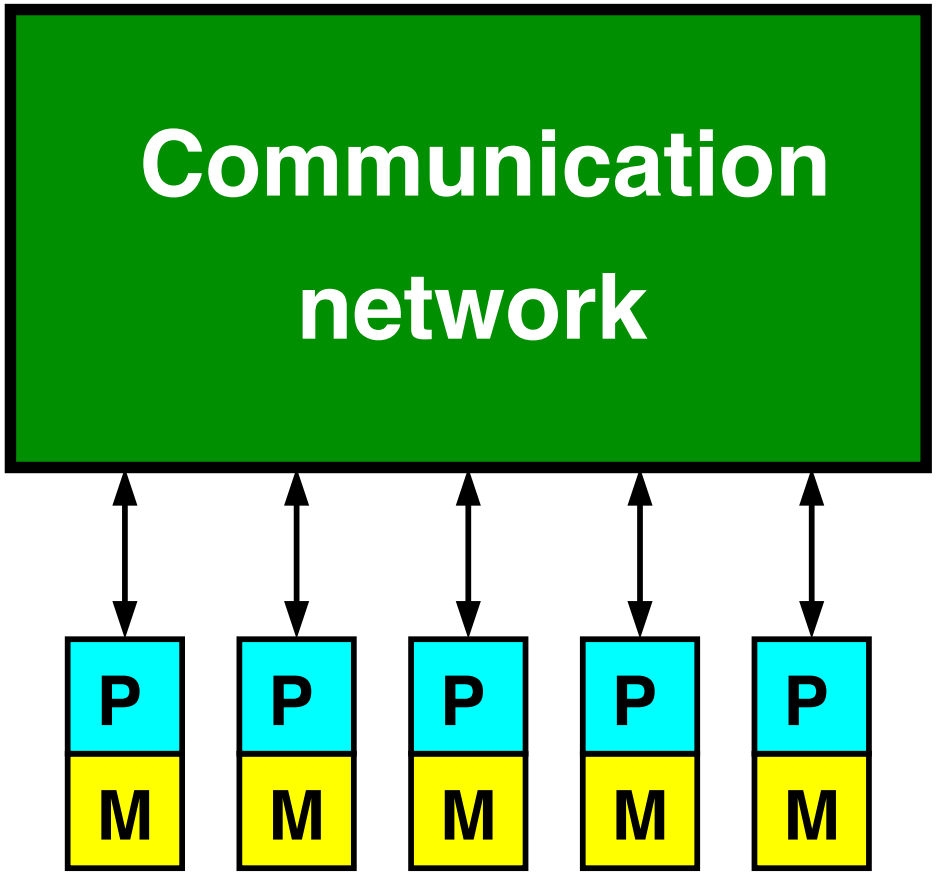
Time complexity

- Linear time complexity $\mathcal{O}(|E|)$ if edges of each vertex are sorted by weight.
- Sorting costs are

$$\sum_v deg(v) \log deg(v) \leq \sum_v deg(v) \log \Delta = 2|E| \log \Delta,$$

  where Δ is the maximum vertex degree.
- This algorithm is based on a dominant-edge algorithm by Preis (1999), called LAM, which is linear-time $\mathcal{O}(|E|)$, does not need sorting, and also is a $1/2$-approximation, but is hard to parallelise.

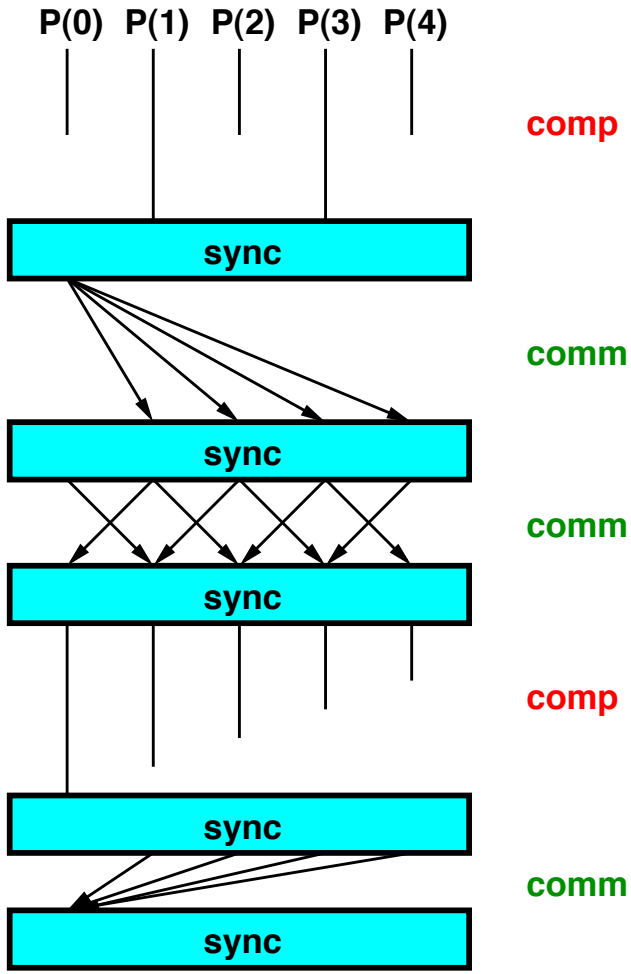
Parallel computer: abstract model



Bulk synchronous parallel (BSP) computer.
Proposed by Leslie Valiant, 1989.

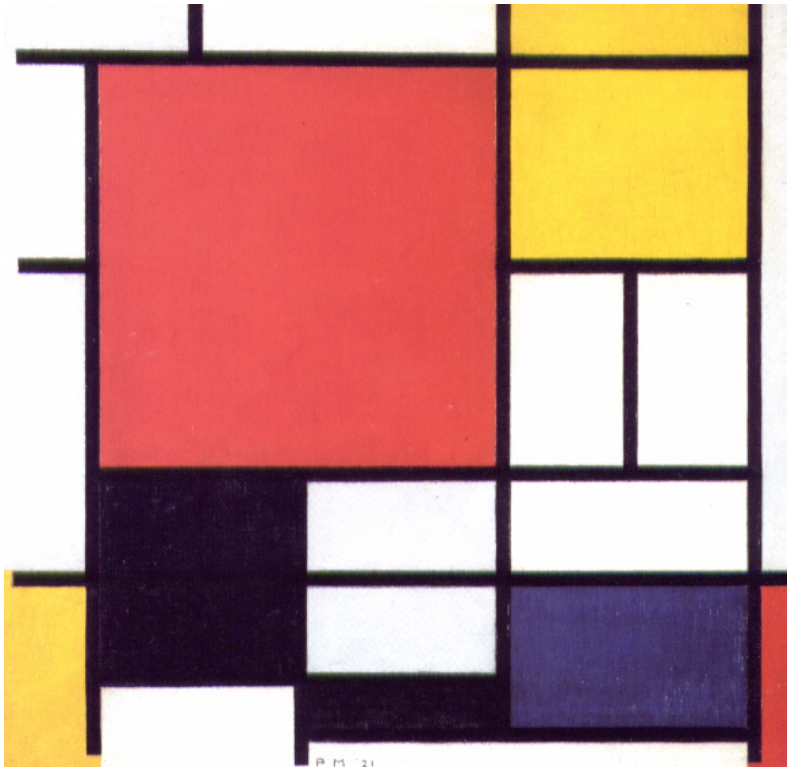
Universiteit Utrecht

Parallel algorithm: supersteps



Universiteit Utrecht

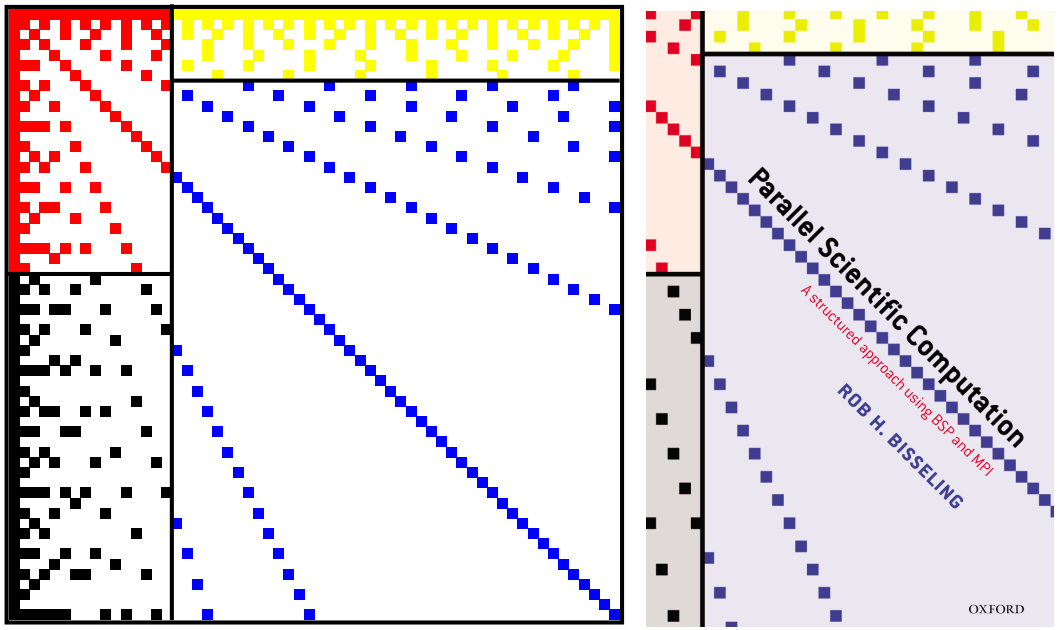
Composition with Red, Yellow, Blue and Black



Piet Mondriaan 1921

Universiteit Utrecht

Mondriaan data distribution for matrix prime60



- Non-Cartesian block distribution of 60×60 matrix prime60 with 462 nonzeros, for $p = 4$
- $a_{ij} \neq 0 \iff i|j \text{ or } j|i \qquad (1 \leq i, j \leq 60)$

Universiteit Utrecht

Parallel algorithm (Manne & Bisseling, 2007)

- Processor $P(s)$ has vertex set V_s, with

$$\bigcup_{s=0}^{p-1} V_s = V$$

  and $V_s \cap V_t = \emptyset$ if $s \neq t$.
- This is a p-way partitioning of the vertex set.

Halo vertices

- The adjacency set Adj_v of a vertex v may contain vertices w from another processor.
- We define the set of halo vertices

$$H_s = \bigcup_{v \in V_s} Adj_v \setminus V_s$$

- The weights $\omega(v, w)$ are stored with the edges, for all $v \in V_s$ and $w \in V_s \cup H_s$.
- $E_s = \{(v, w) \in E : v \in V_s\}$
  is the subset of all the edges connected to V_s.

Parallel algorithm for $P(s)$: initialisation

function ParMatching(V_s, H_s, E_s, distribution ϕ)
    for all $v \in V_s$ **do**
        $pref(v) = null$
    $D_s := \emptyset$
    $M_s := \emptyset$

    { Find dominant edges }
    for all $v \in V_s$ **do**
        $Adj_v := \{w \in V_s \cup H_s : (v, w) \in E_s\}$
        $SetNewPreference(v, Adj_v, pref, V_s, D_s, M_s, \phi)$
    Sync

Setting a vertex preference

function SetNewPreference(v, Adj, V, D, M, ϕ)
  $pref(v) := \text{argmax}\{\omega(v, w) \ : \ w \in Adj\}$
  if $pref(v) \in V$ **then**
    if $pref(pref(v)) = v$ **then**
      $D := D \cup \{v, pref(v)\}$
      $M := M \cup \{(v, pref(v))\}$
  else
    put $proposal(v, pref(v))$ in $P(\phi(pref(v)))$

How to propose

Source: www.theguardian.com

proposal(v, w): v proposes to w

Universiteit Utrecht

Parallel algorithm for $P(s)$: main loop

process received messages

while $D_s \neq \emptyset$ **do**
  pick $v \in D_s$
  $D_s := D_s \setminus \{v\}$
  for all $x \in Adj_v \setminus \{pref(v)\} \ : \ (x, pref(x)) \notin M_s$ **do**
    if $x \in V_s$ **then**
      $Adj_x := Adj_x \setminus \{v\}$
      $SetNewPreference(x, Adj_x, pref, V_s, D_s, M_s, \phi)$
    else $\{x \in H_s\}$
      put $unavailable(v, x)$ in $P(\phi(x))$
Sync

Parallel algorithm for $P(s)$: process received messages

for all messages m received **do**
  if $m = proposal(x, y)$ **then**
    if $pref(y) = x$ **then**
      $D_s := D_s \cup \{y\}$
      $M_s := M_s \cup \{(x, y)\}$
      put $accepted(x, y)$ in $P(\phi(x))$
  if $m = accepted(x, y)$ **then**
    $D_s := D_s \cup \{x\}$
    $M_s := M_s \cup \{(x, y)\}$
  if $m = unavailable(v, x)$ **then**
    if $(x, pref(x)) \notin M_s$ **then**
      $Adj_x := Adj_x \setminus \{v\}$
      $SetNewPreference(x, Adj_x, pref, V_s, D_s, M_s, \phi)$

Termination

- The algorithm alternates supersteps of computation running the main loop and communication handling the received messages.
- The whole algorithm terminates when no messages have been received by processor $P(s)$ and the local set D_s is empty, for all s.
- This can be checked at every synchronisation point.

Load balance

- Processors can have different amounts of work, even if they have the same number of vertices or edges.
- Use can be made of a global clock based on ticks, the unit of time needed to 'handle' a vertex x (in $\mathcal{O}(1)$).
- Here, 'handling' could mean setting a new preference.
- After every k ticks, everybody synchronises.

Synchronisation frequency

- Guidance for the choice of k is provided by the BSP parameter l, the cost of a global synchronisation.
- Choosing $k \geq l$ guarantees that at most 50% of the total time is spent in synchronisation.
- Choosing k sufficiently small will cause all processors to be busy during most supersteps.
- Good choice: $k = 2l$?

MulticoreBSP enables shared-memory BSP

KU Leuven | Flanders ExaScience Lab | Utrecht University

MulticoreBSP *efficient, easy & correct parallelisation*

HOME SOFTWARE COMMUNITY FORUM

- Introduction
- BSP model
- BSP library

Home → Introduction

Print this page

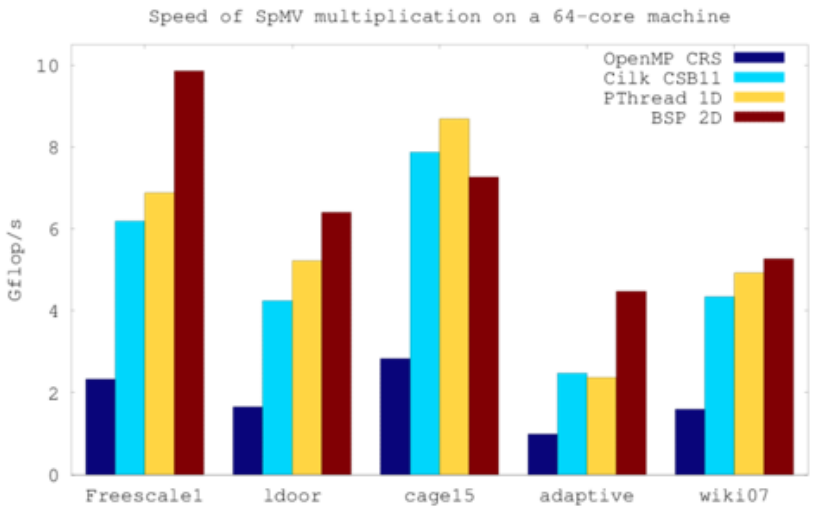
RECENT NEWS

- Version 1.2, and a roadmap for future releases.
- MulticoreBSP for C, version 1.1 released.

see all news items

Introduction

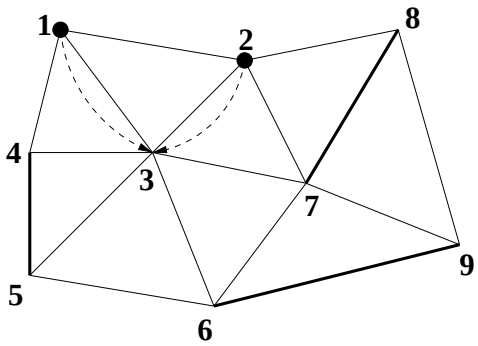
MulticoreBSP brings **Bulk Synchronous Parallel (BSP)** programming to modern multicore processors. BSP programming leads to high-performance codes:



Albert-Jan Yzelman 2014, www.multicorebsp.org

Universiteit Utrecht

GPU matching

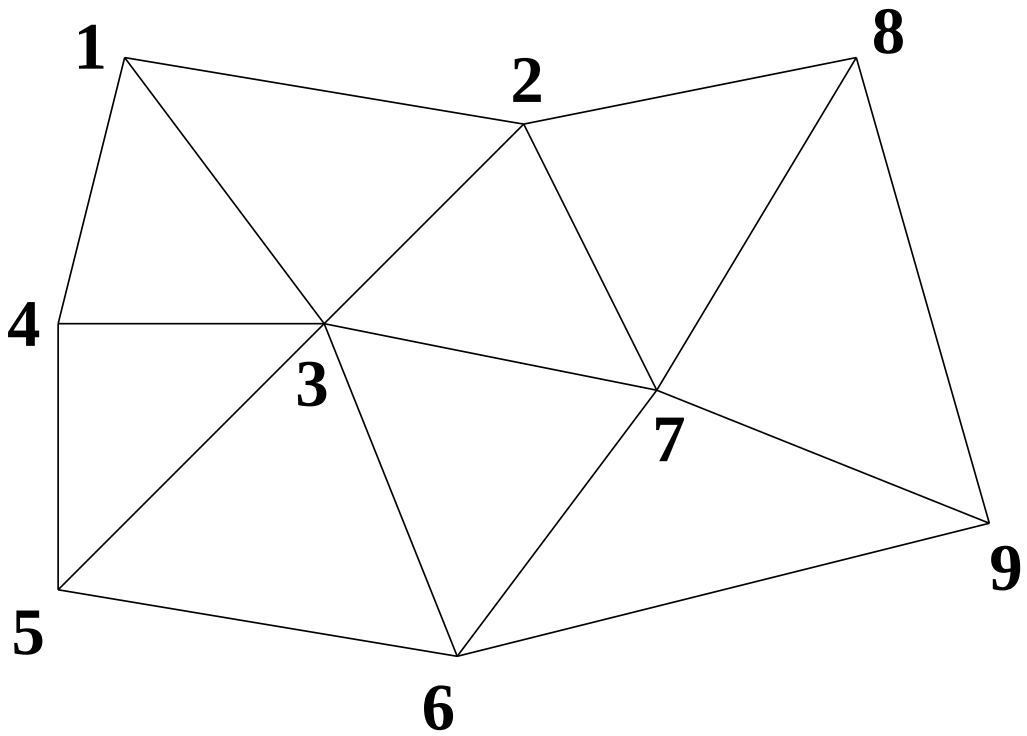


- A different approach, tightly coupled to the GPU architecture.
- To prevent matching conflicts, we create two groups of vertices:
  - **Blue** vertices propose.
  - **Red** vertices respond.
- Proposals that were responded to, are matched.

Universiteit Utrecht

GPU matching

Colour
Propose
Respond
Match

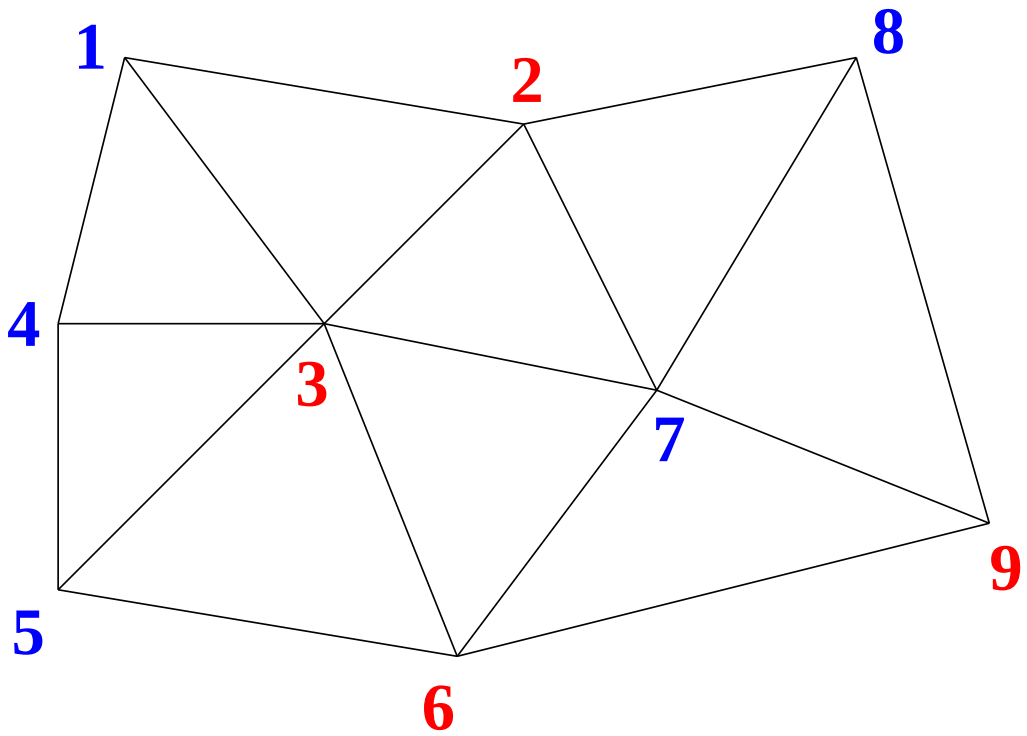


	1	2	3	4	5	6	7	8	9
π	-	-	-	-	-	-	-	-	-
σ	-	-	-	-	-	-	-	-	-

Universiteit Utrecht

GPU matching

Colour
Propose
Respond
Match

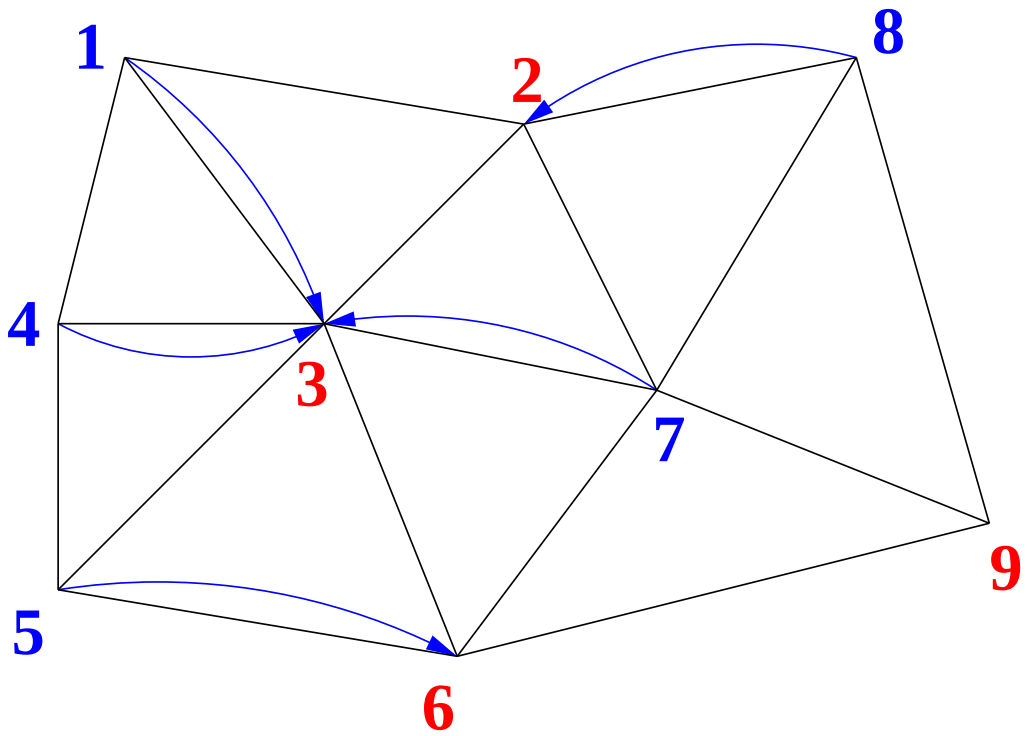


	1	2	3	4	5	6	7	8	9
π	b	r	r	b	b	r	b	b	r
σ	-	-	-	-	-	-	-	-	-

Universiteit Utrecht

GPU matching

Colour
Propose
Respond
Match

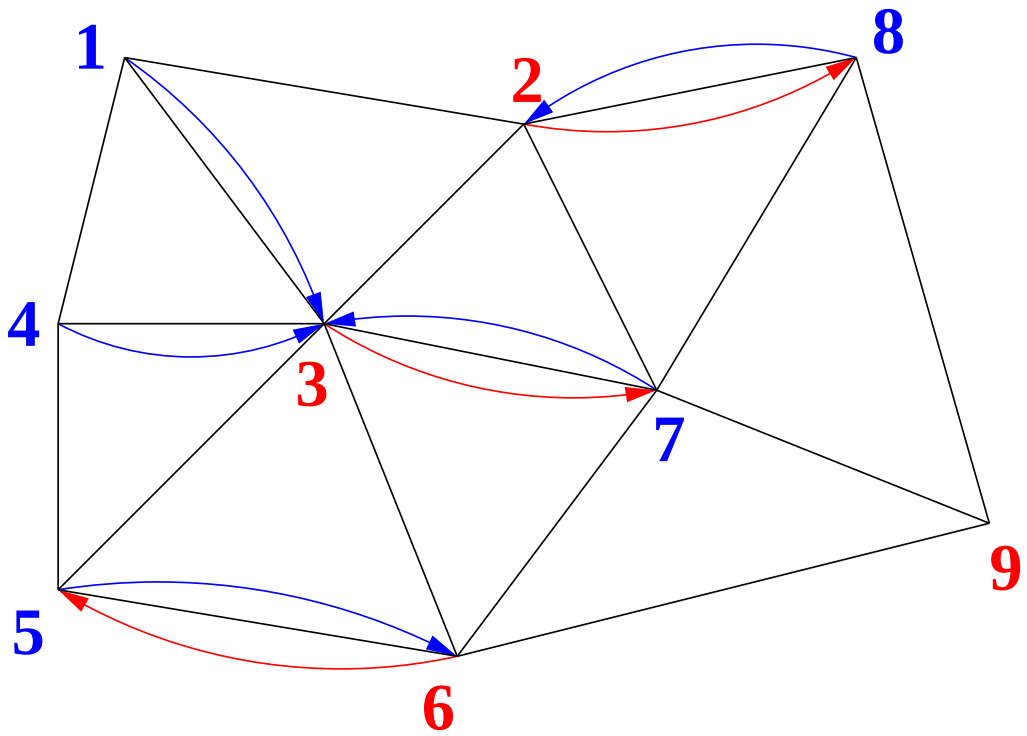


	1	2	3	4	5	6	7	8	9
π	b	r	r	b	b	r	b	b	r
σ	3	-	-	3	6	-	3	2	-

Universiteit Utrecht

GPU matching

Colour
Propose
Respond
Match

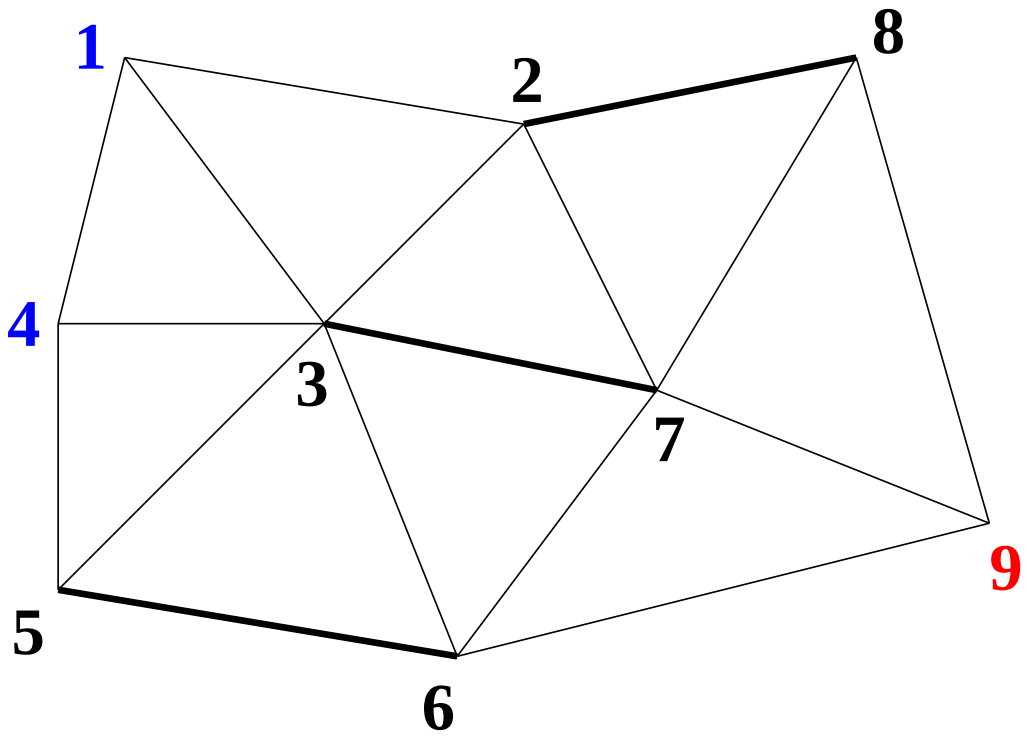


	1	2	3	4	5	6	7	8	9
π	**b**	**r**	**r**	**b**	**b**	**r**	**b**	**b**	**r**
σ	3	8	7	3	6	5	3	2	-

Universiteit Utrecht

GPU matching

Colour
Propose
Respond
Match

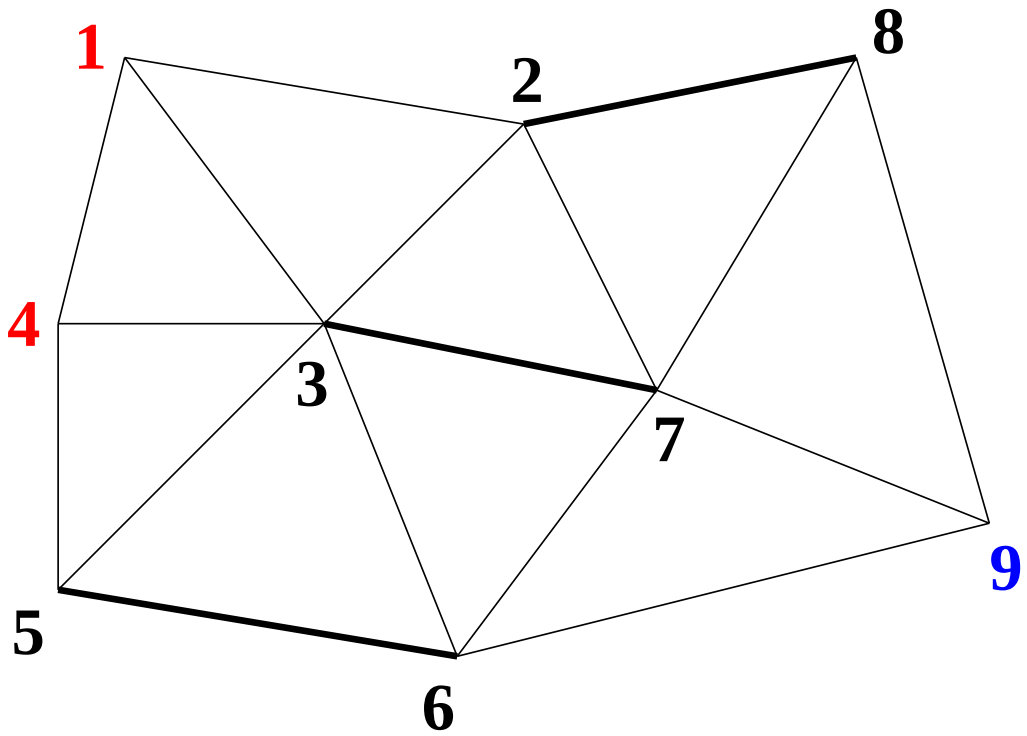


	1	2	3	4	5	6	7	8	9
π	**b**	2	3	**b**	5	5	3	2	**r**
σ	3	8	7	3	6	5	3	2	-

Universiteit Utrecht

GPU matching

Colour
Propose
Respond
Match

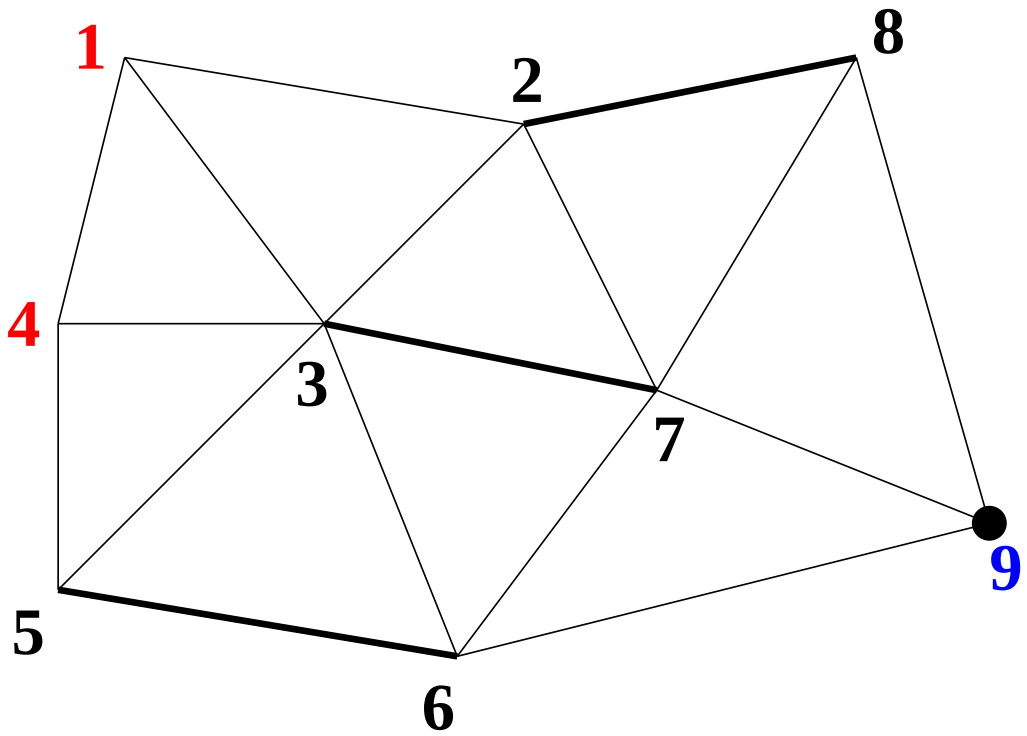


	1	2	3	4	5	6	7	8	9
π	r	2	3	r	5	5	3	2	b
σ	3	8	7	3	6	5	3	2	-

Universiteit Utrecht

GPU matching

Colour
Propose
Respond
Match

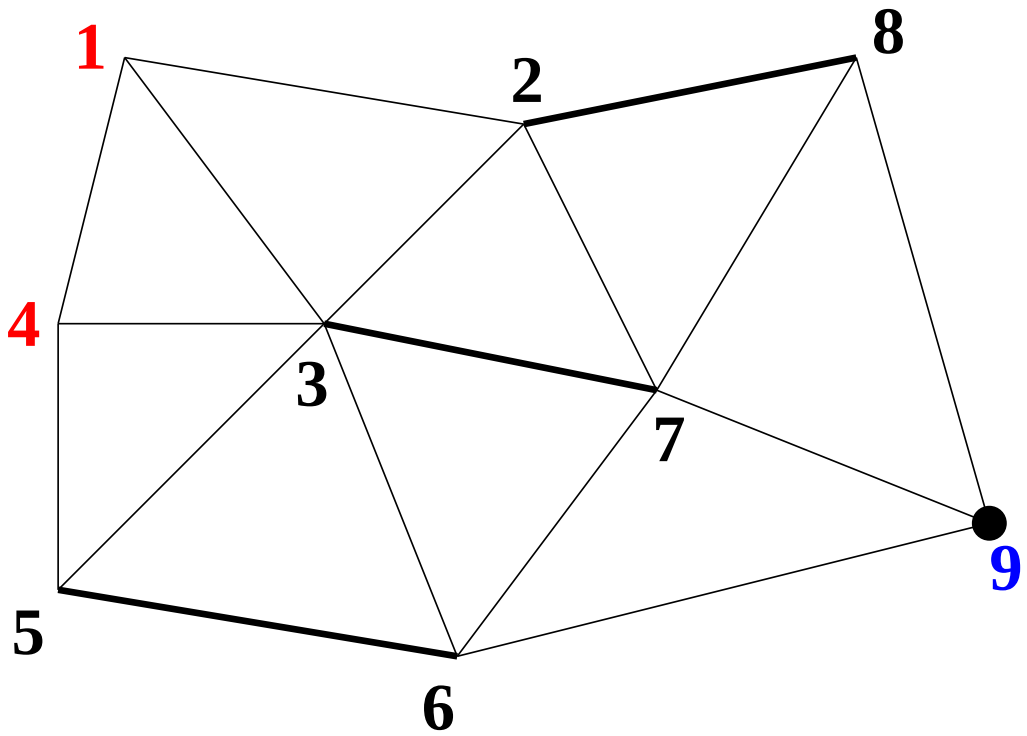


	1	2	3	4	5	6	7	8	9
π	**r**	2	3	**r**	5	5	3	2	**b**
σ	-	-	-	-	-	-	-	-	**d**

Universiteit Utrecht

GPU matching

Colour
Propose
Respond
Match

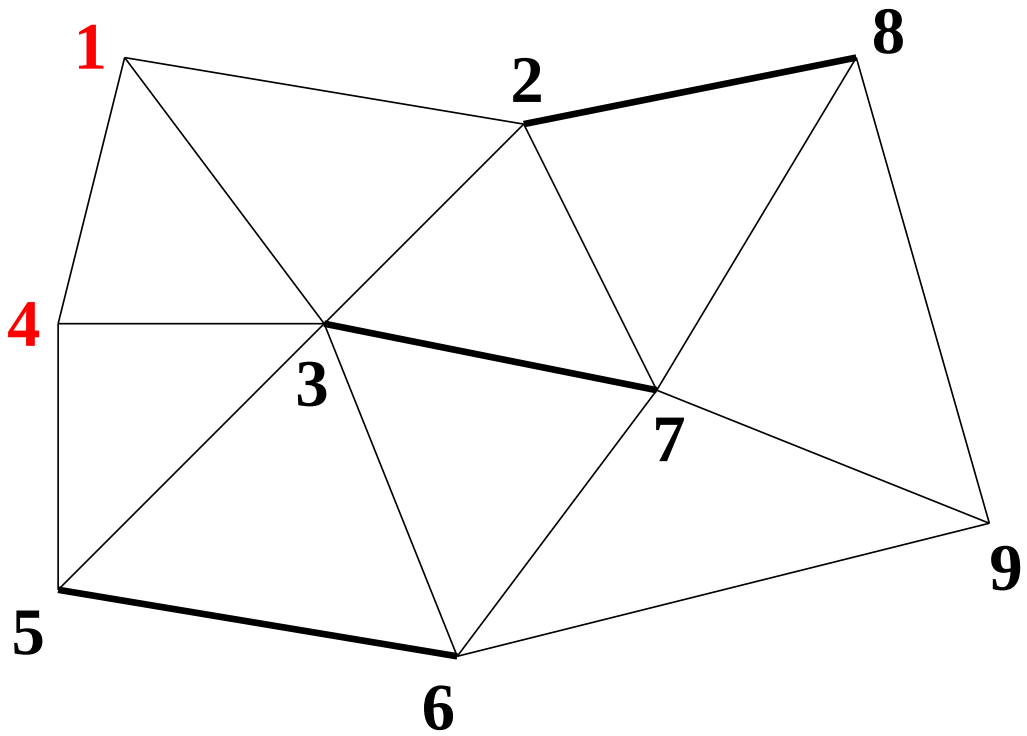


	1	2	3	4	5	6	7	8	9
π	**r**	2	3	**r**	5	5	3	2	**b**
σ	-	-	-	-	-	-	-	-	**d**

Universiteit Utrecht

GPU matching

Colour
Propose
Respond
Match

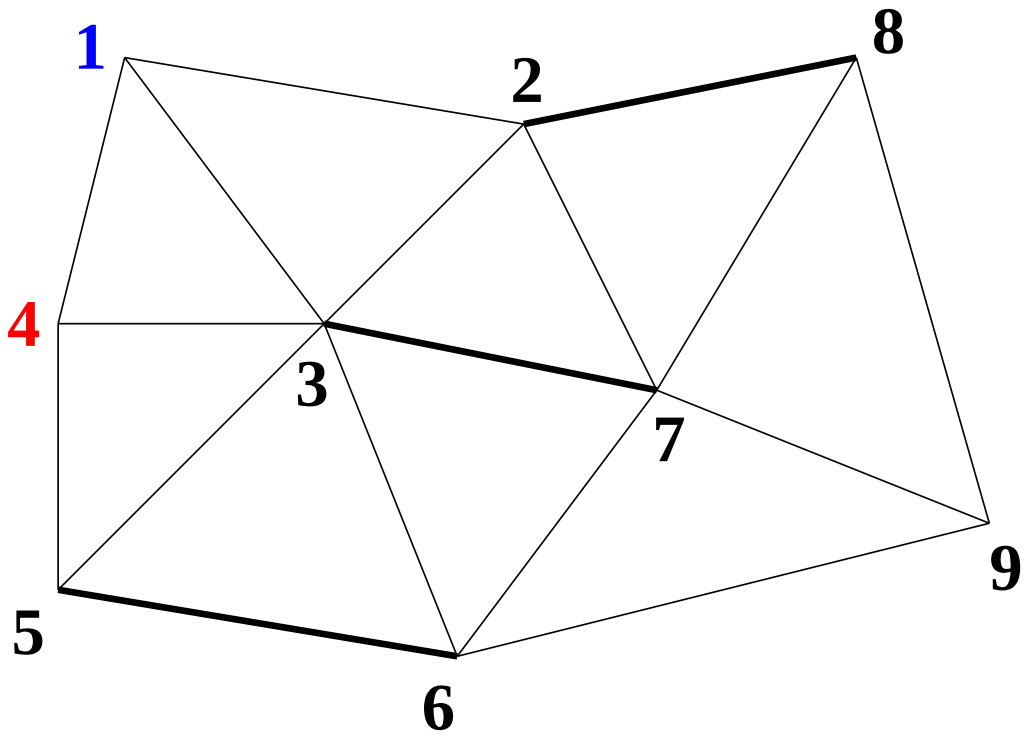


	1	2	3	4	5	6	7	8	9
π	**r**	2	3	**r**	5	5	3	2	**d**
σ	-	-	-	-	-	-	-	-	**d**

Universiteit Utrecht

GPU matching

Colour
Propose
Respond
Match

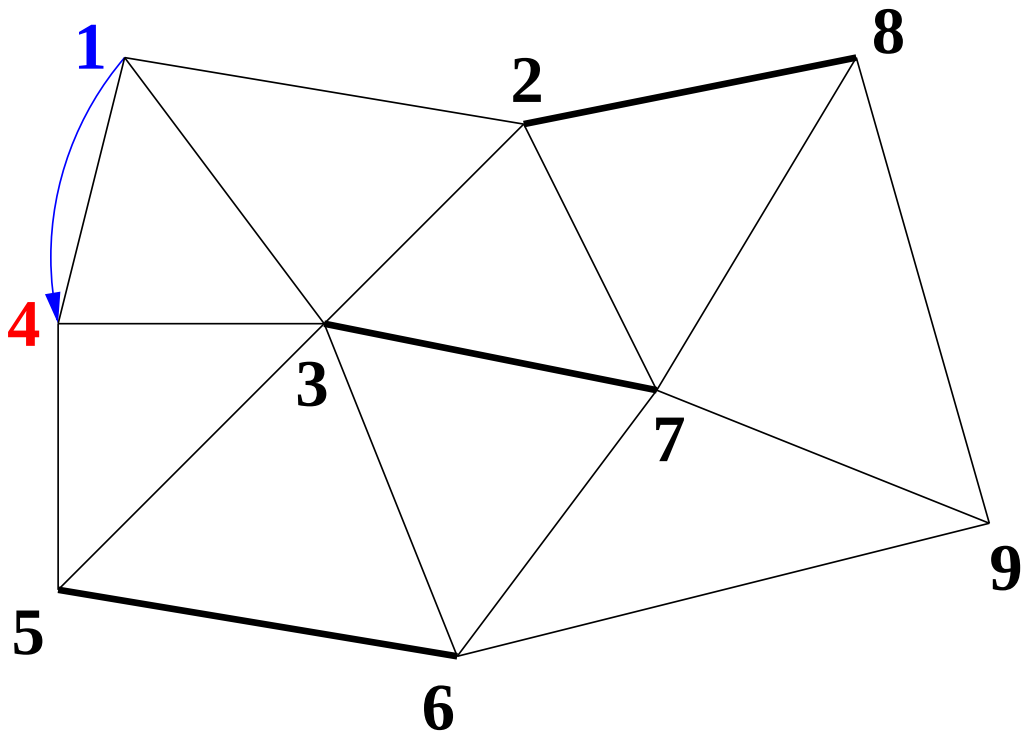


	1	2	3	4	5	6	7	8	9
π	**b**	2	3	**r**	5	5	3	2	**d**
σ	-	-	-	-	-	-	-	-	**d**

Universiteit Utrecht

GPU matching

Colour
Propose
Respond
Match

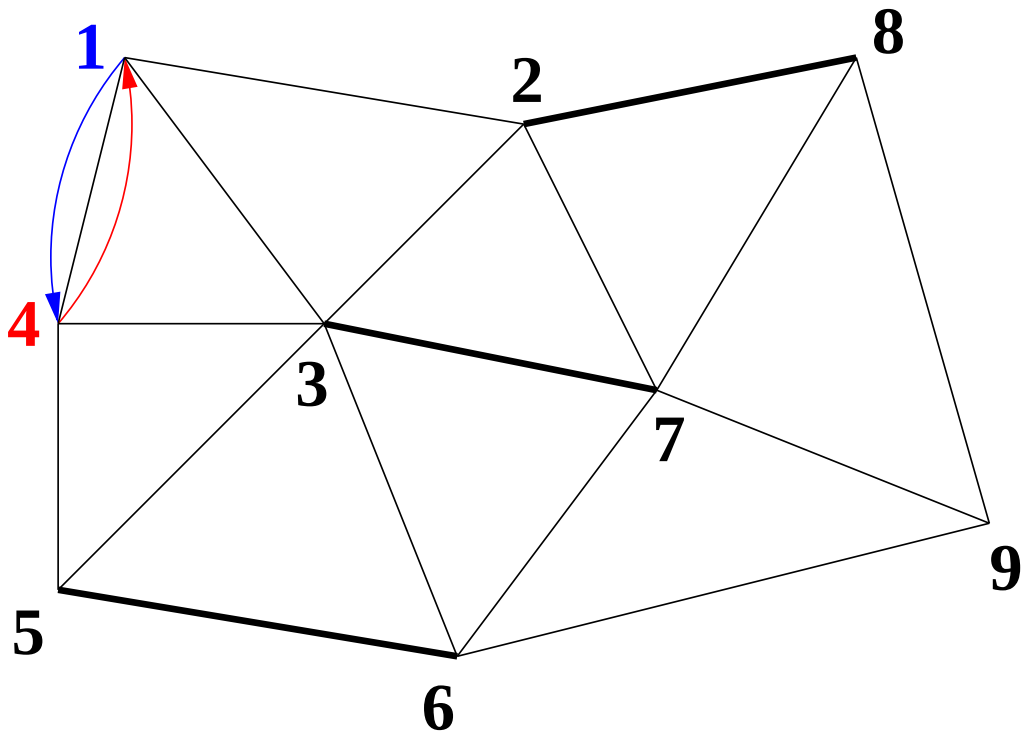


	1	2	3	4	5	6	7	8	9
π	**b**	2	3	**r**	5	5	3	2	**d**
σ	4	-	-	-	-	-	-	-	-

Universiteit Utrecht

GPU matching

Colour
Propose
Respond
Match

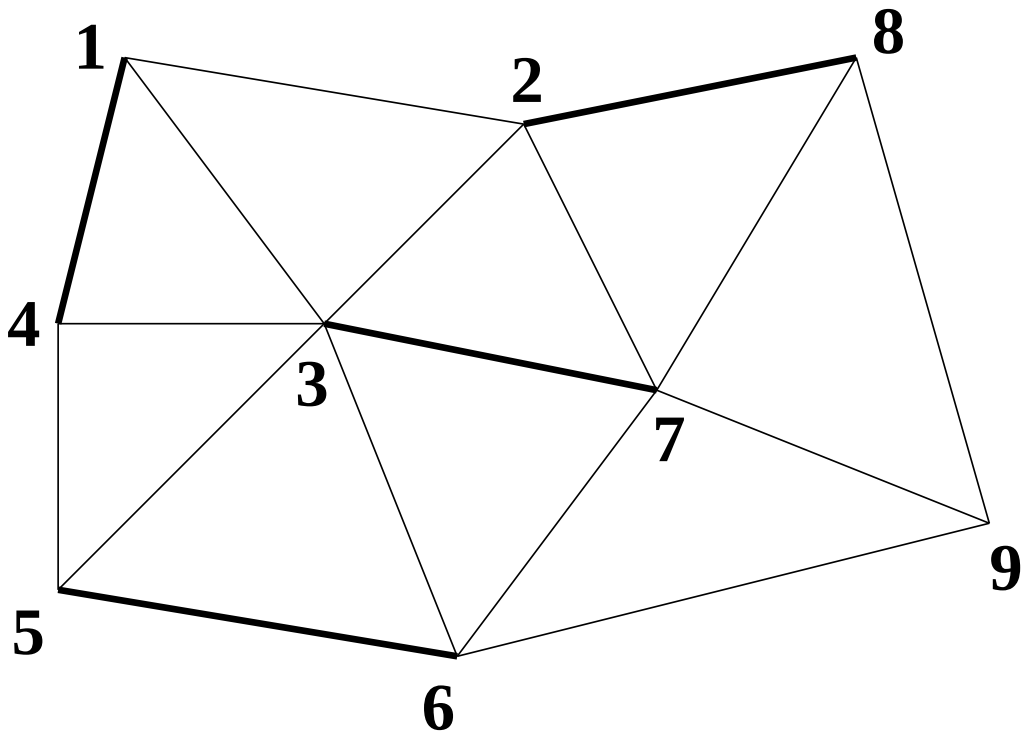


	1	2	3	4	5	6	7	8	9
π	**b**	2	3	**r**	5	5	3	2	**d**
σ	4	-	-	1	-	-	-	-	-

Universiteit Utrecht

GPU matching

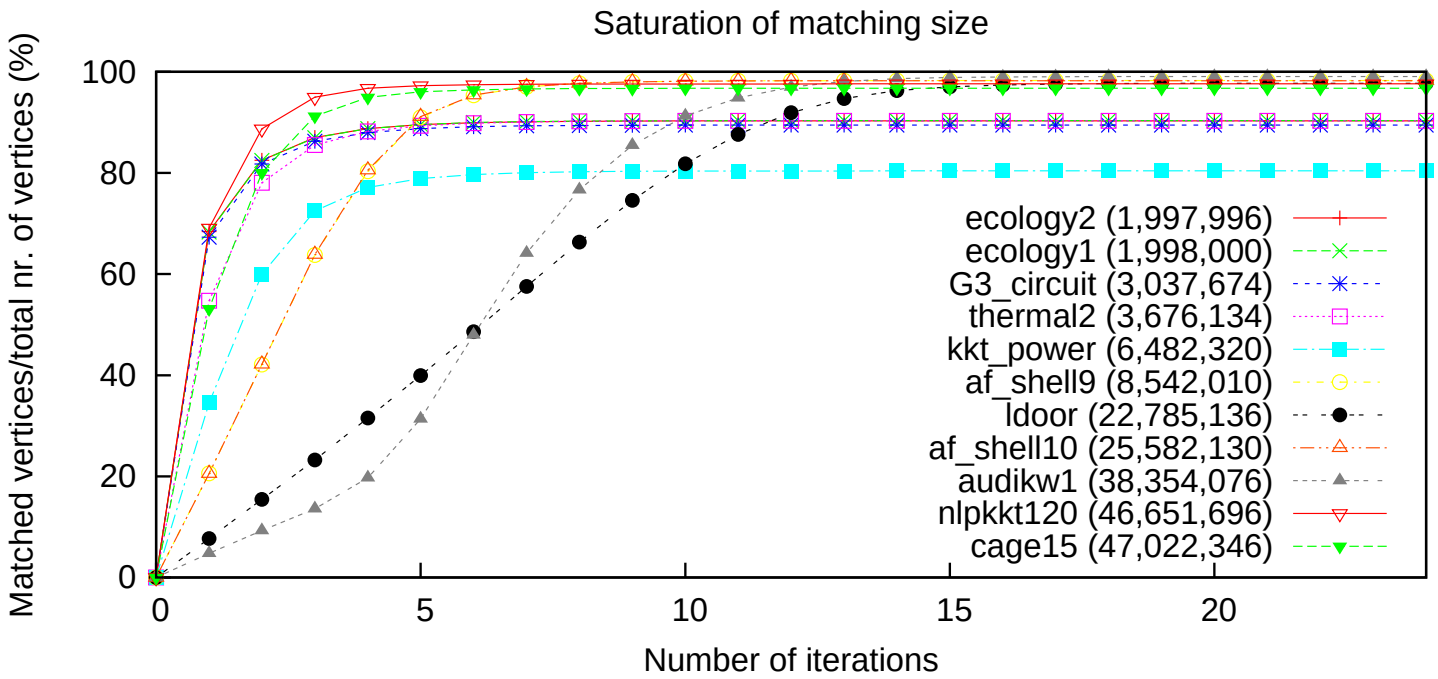
Colour
Propose
Respond
Match



	1	2	3	4	5	6	7	8	9
π	1	2	3	1	5	5	3	2	**d**
σ	4	-	-	1	-	-	-	-	-

Universiteit Utrecht

Quality of the matching



Fraction of matched vertices as a function of the number of iterations. Number of edges between 2 and 47 million.

Universiteit Utrecht

Random colouring of the vertices

- At each iteration, we colour the vertices $v \in V$ differently.
- For a fixed $p \in [0, 1]$

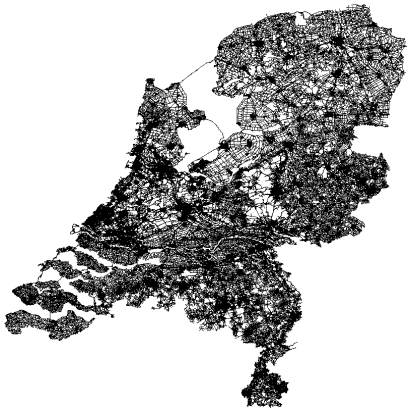
$$\textbf{colour}(v) = \begin{cases} \textbf{blue} & \text{with probability } p, \\ \textbf{red} & \text{with probability } 1 - p. \end{cases}$$

- How to choose p? Maximise the number of matched vertices.
- For large random graphs, the expected fraction of matched vertices can be approximated by

$$2\,(1-p)\left(1 - e^{-\frac{p}{1-p}}\right).$$

This is independent of the edge density.

Clustering of road network of the Netherlands

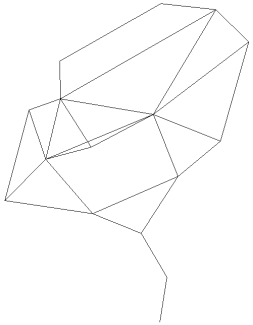


(a) G^0

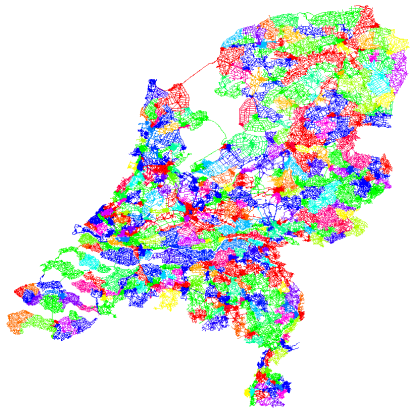
(b) G^{11}

(c) G^{21}

(d) G^{26}



(e) G^{33}



(f) Best clustering (G^{21})

Graph with 2,216,688 vertices and 2,441,238 edges yields 506 clusters with modularity 0.995.

Universiteit Utrecht

Formal definition of a clustering

- A clustering of an undirected graph $G = (V, E)$ is a collection $\mathcal{C}$ of disjoint subsets of V satisfying

$$V = \bigcup_{C \in \mathcal{C}} C.$$

- Elements $C \in \mathcal{C}$ are called clusters.
- The number of clusters is not fixed beforehand.

Quality measure for clustering: modularity

- The quality measure modularity was introduced by Newman and Girvan in 2004 for finding communities.
- Let $G = (V, E, \omega)$ be a weighted undirected graph without self-edges. We define

$$\zeta(v) = \sum_{(u,v)\in E} \omega(u,v), \qquad \Omega = \sum_{e\in E} \omega(e).$$

- Then, the modularity of a clustering $\mathcal{C}$ of G is defined by

$$\mathrm{mod}(\mathcal{C}) = \frac{\sum_{C\in\mathcal{C}} \sum_{\substack{(u,v)\in E \\ u,v\in C}} \omega(u,v)}{\Omega} - \frac{\sum_{C\in\mathcal{C}} \left(\sum_{v\in C} \zeta(v) \right)^2}{4\Omega^2}.$$

- $-\frac{1}{2} \leq \mathrm{mod}(\mathcal{C}) \leq 1$.

Merging clusters: change in modularity

- The set of all cut edges between clusters C and C' is

$$\text{cut}(C, C') = \{\{u, v\} \in E \mid u \in C, v \in C'\}$$

- If we merge clusters C and C' from $\mathcal{C}$ into one cluster $C \cup C'$, then we get a new clustering $\mathcal{C}'$ with

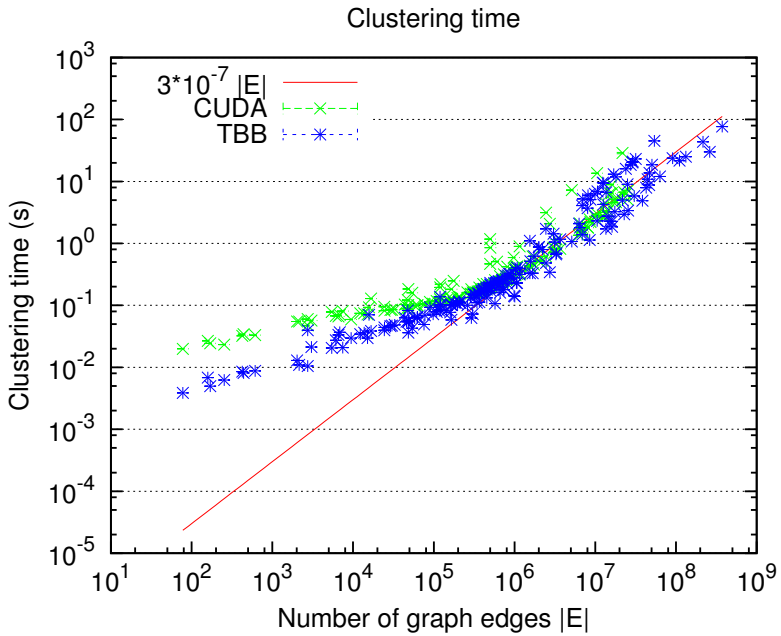
$$\text{mod}(\mathcal{C}') = \text{mod}(\mathcal{C}) + \frac{1}{4\,\Omega^2}\left(4\,\Omega\,\omega(\text{cut}(C, C')) - 2\,\zeta(C)\,\zeta(C')\right),$$

$$\zeta(C \cup C') = \zeta(C) + \zeta(C').$$

Agglomerative greedy clustering heuristic

$\max \leftarrow -\infty$
$G^0 = (V^0, E^0, \omega^0, \zeta^0)$
$i \leftarrow 0$
$\mathcal{C}^0 \leftarrow \{\{v\} \mid v \in V\}$
while $|V^i| > 1$ **do**
    if $\mathrm{mod}(G, \mathcal{C}^i) \geq \max$ **then**
        $\max \leftarrow \mathrm{mod}(G, \mathcal{C}^i)$
        $\mathcal{C}^{\mathrm{best}} \leftarrow \mathcal{C}^i$
    $\mu \leftarrow$ **weighted_match_clusters**(G^i)
    $(\pi^i, G^{i+1}) \leftarrow$ **coarsen**(G^i, μ)
    $\mathcal{C}^{i+1} \leftarrow \{\{v \in V \mid (\pi^i \circ \cdots \circ \pi^0)(v) = u\} \mid u \in V^{i+1}\}$
    $i \leftarrow i + 1$
return $\mathcal{C}^{\mathrm{best}}$

Results: clustering time for DIMACS graphs



- DIMACS categories: `clustering/`, `coauthor/`, `streets/`, `random/`, `delaunay/`, `matrix/`, `walshaw/`, `dyn-frames/`, and `redistrict/`.
- CUDA implementation with the Thrust template library and Intel TBB implementation.
- Web link graph `uk-2002` with 0.26 billion vertices clustered in 30 s using Intel TBB.

DIMACS road networks and coauthor graphs

G	$\|V\|$	$\|E\|$	mod CU	t CU	mod TBB	t TBB
luxembourg	114,599	119,666	0.99	0.13	0.99	0.14
belgium	1,441,295	1,549,970	0.99	0.44	0.99	1.11
netherlands	2,216,688	2,441,238	0.99	0.62	0.99	1.72
italy	6,686,493	7,013,978	1.00	1.54	1.00	5.26
great-britain	7,733,822	8,156,517	1.00	1.79	1.00	6.00
germany	11,548,845	12,369,181	1.00	2.82	1.00	9.57
asia	11,950,757	12,711,603	1.00	2.69	1.00	9.33
europe	50,912,018	54,054,660	-	-.-	1.00	45.21
coAuthorsCite	227,320	814,134	0.84	0.42	0.85	0.23
coAuthorsDBLP	299,067	977,676	0.75	0.59	0.76	0.28
citationCite	268,495	1,156,647	0.64	0.89	0.68	0.32
coPapersDBLP	540,486	15,245,729	0.64	6.43	0.67	2.28
coPapersCite	434,102	16,036,720	0.75	6.49	0.77	2.27

mod = modularity, t = time in s, CU = CUDA

Conclusions

- BSP is extremely suitable for parallel graph computations:
  - no worries about communication because we buffer messages until the next synchronisation;
  - no send-receive pairs, but one-sided put or get operations;
  - BSP cost model gives synchronisation frequency;
  - correctness proof of algorithm becomes simpler;
  - no deadlock possible.
- Matching can be the basis for clustering, as demonstrated for GPUs and multicore CPUs.
- We clustered Asia's road network with 12M vertices and 12.7M edges in 2.7 seconds on a GPU.