Graph matching for BSP

Rob H. Bisseling

Mathematical Institute, Utrecht University
Joint work with Fredrik Manne (Bergen, Norway)

Universiteit Utrecht, November 25, 2015

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

Matching

- Introduction
- Greedy matching

BSP algorithm for edge-weighted matching

- Sequential approximation algorithm
- BSP approximation algorithm

Conclusion

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

Matchmaker, Matchmaker, Make me a match

From the film *Fiddler on the roof*

- Hodel: Well, somebody has to arrange the matches. Young people can't decide these things themselves.
- Hodel: For Papa, make him a scholar.
- Chava: For Mama, make him rich as a king.

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

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

65 SOCIAL OFF Like 963 Tweet 100

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

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

Motivation of graph matching

- Graph matching is a pairing of neighbouring vertices.
- It has applications in
  - medicine: finding suitable organ donors for patients
  - social networks: finding partners
  - scientific computing: finding pivot elements in matrix computations
  - graph coarsening: making the graph smaller by merging similar vertices before partitioning it for parallel computations
  - bioinformatics: finding similarity in Protein-Protein Interaction networks

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

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 10 billion vertices, $n = 10^{10}$, with 1000 edges per vertex, i.e. $m = 10^{13}$.
- Thus, a time of $\mathcal{O}(10^{23}) = 100,000,000$ Petaflop units is far too long. Fastest supercomputer today, the Tianhe-2, performs 34 Petaflop/s and would need 34 days.
- We need linear-time greedy or approximation algorithms.

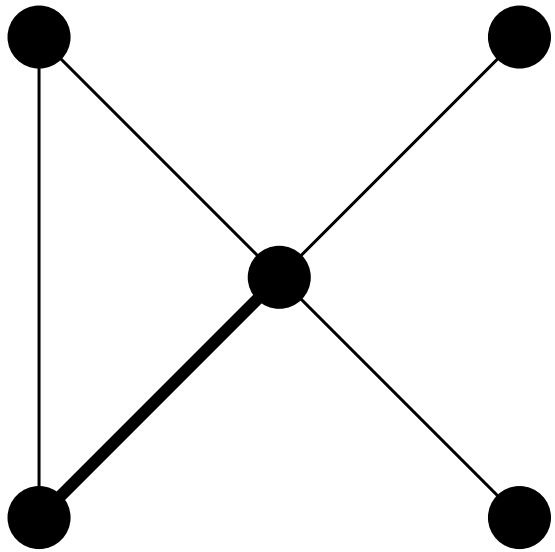
Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

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 $(v, w) = (w, v)$.

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Maximal matching

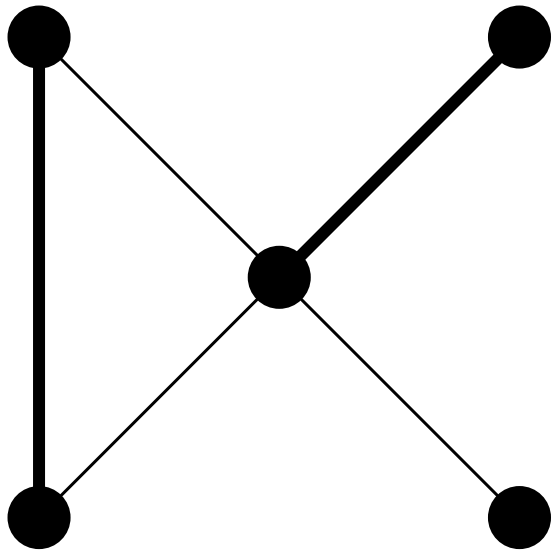


- A matching is maximal if we cannot enlarge it further by adding another edge to it.

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

Maximum matching



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

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

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.

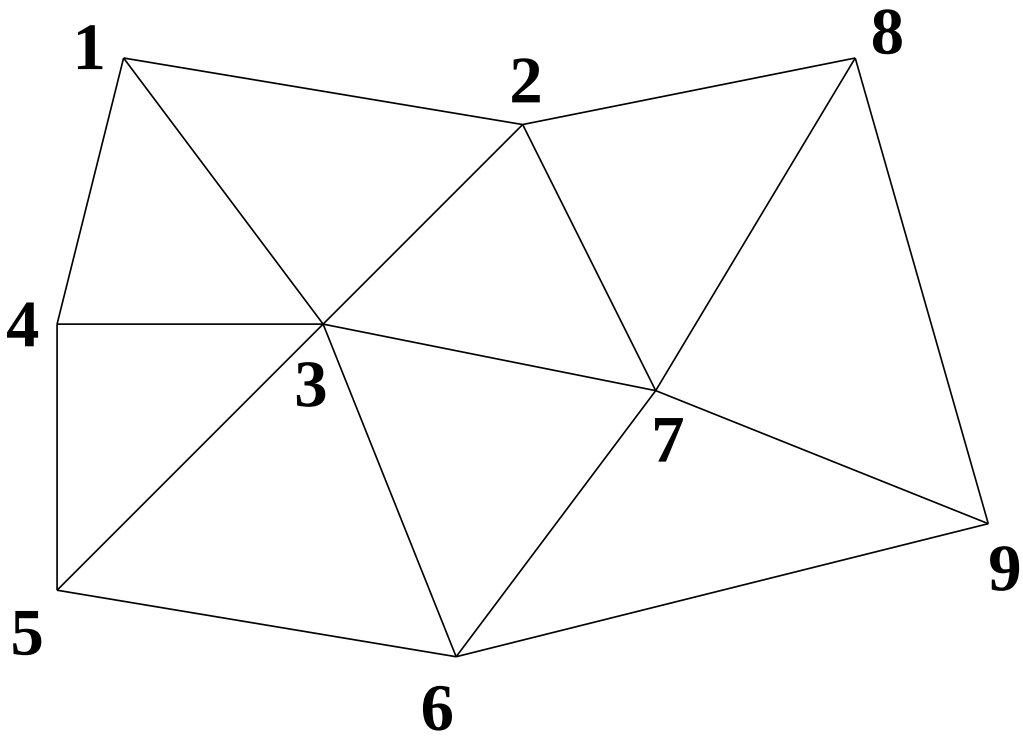
Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

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 the edges by weight, and successively match the vertices v and w of the heaviest available edge (v, w) (greedy matching)

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

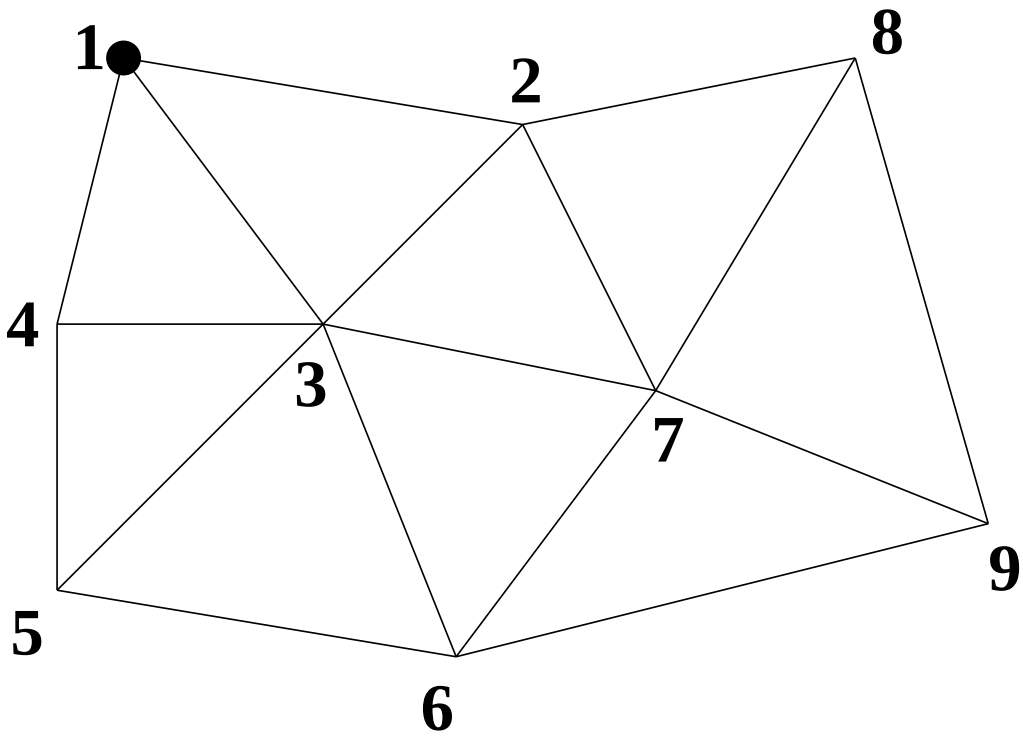
Sequential greedy matching



Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

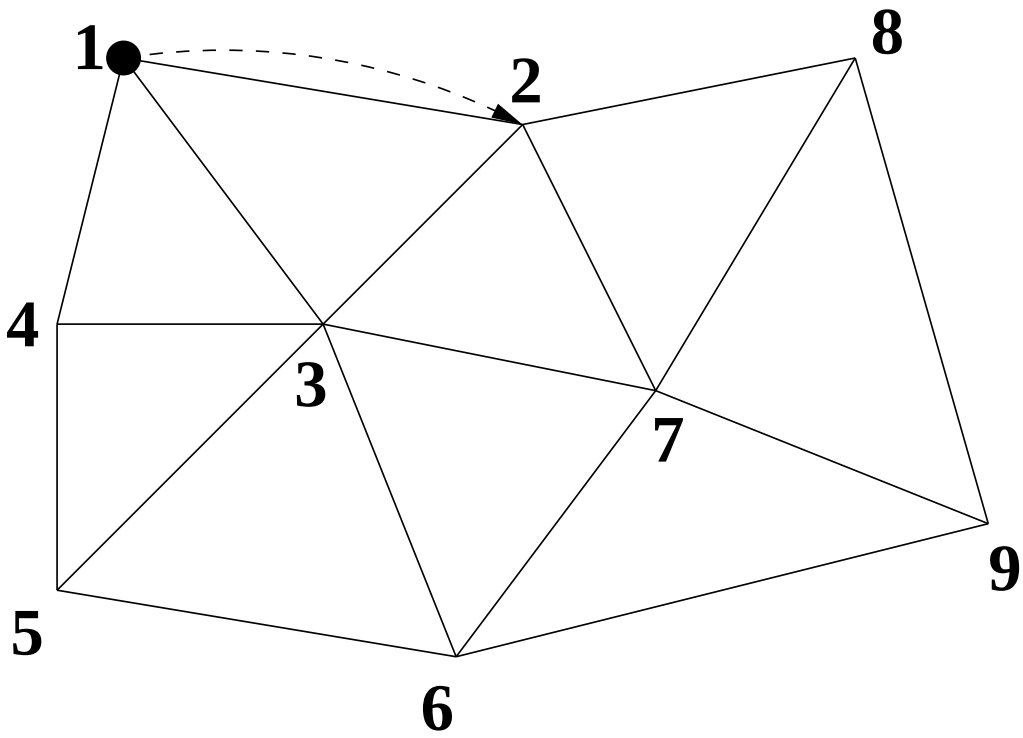
Sequential greedy matching



Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

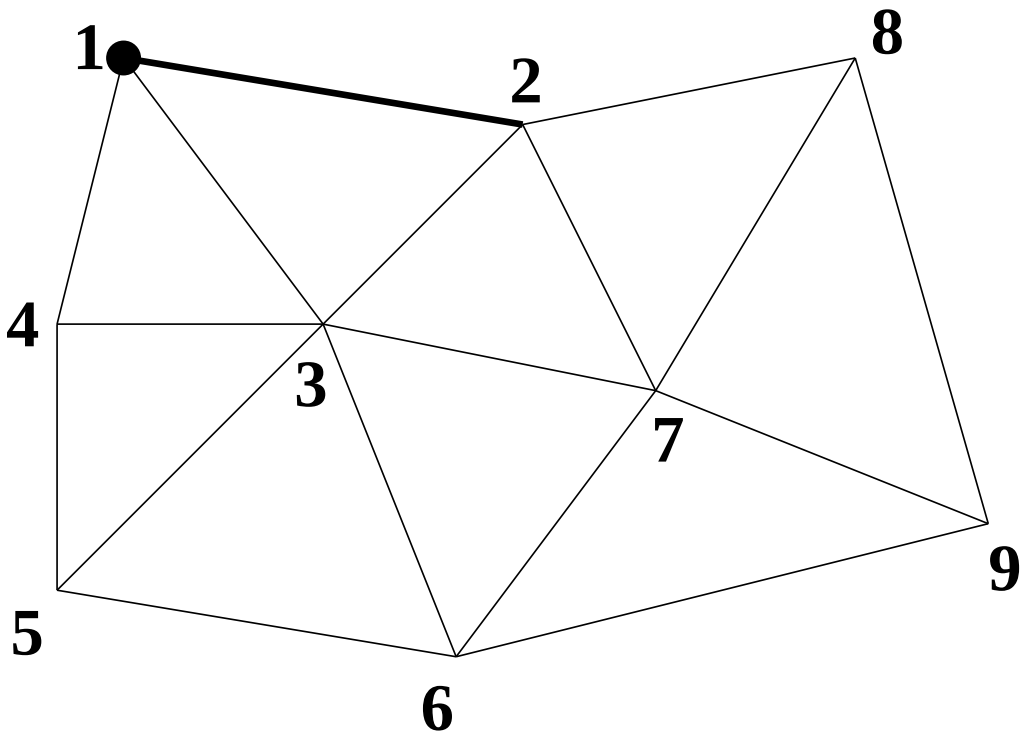
Sequential greedy matching



Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

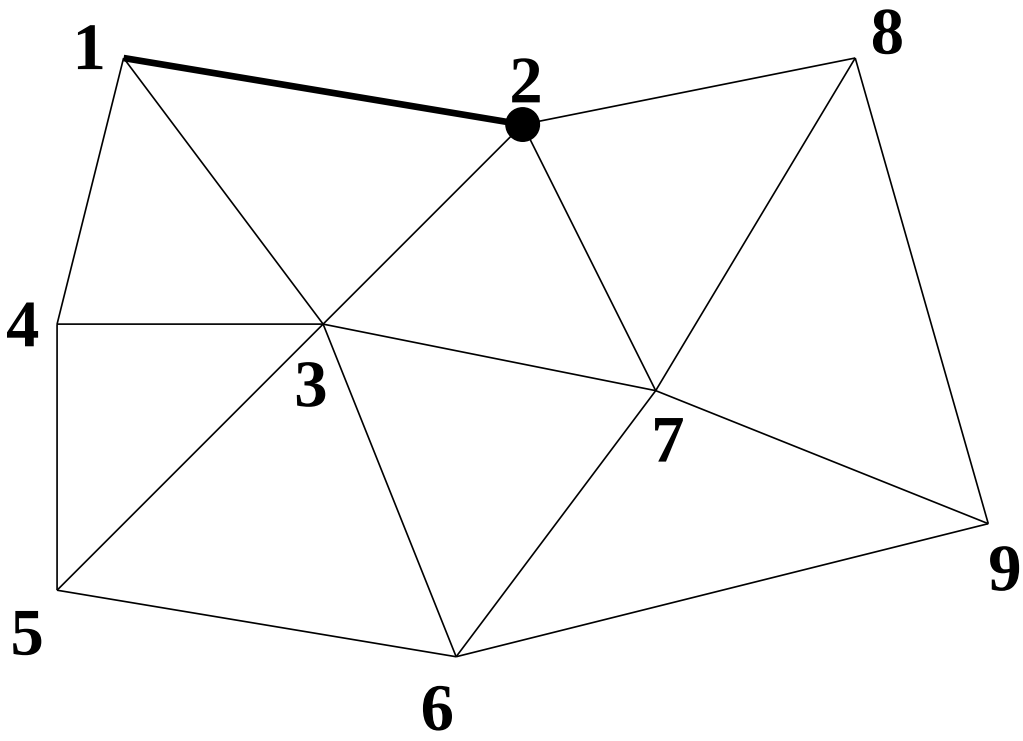
Sequential greedy matching



Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

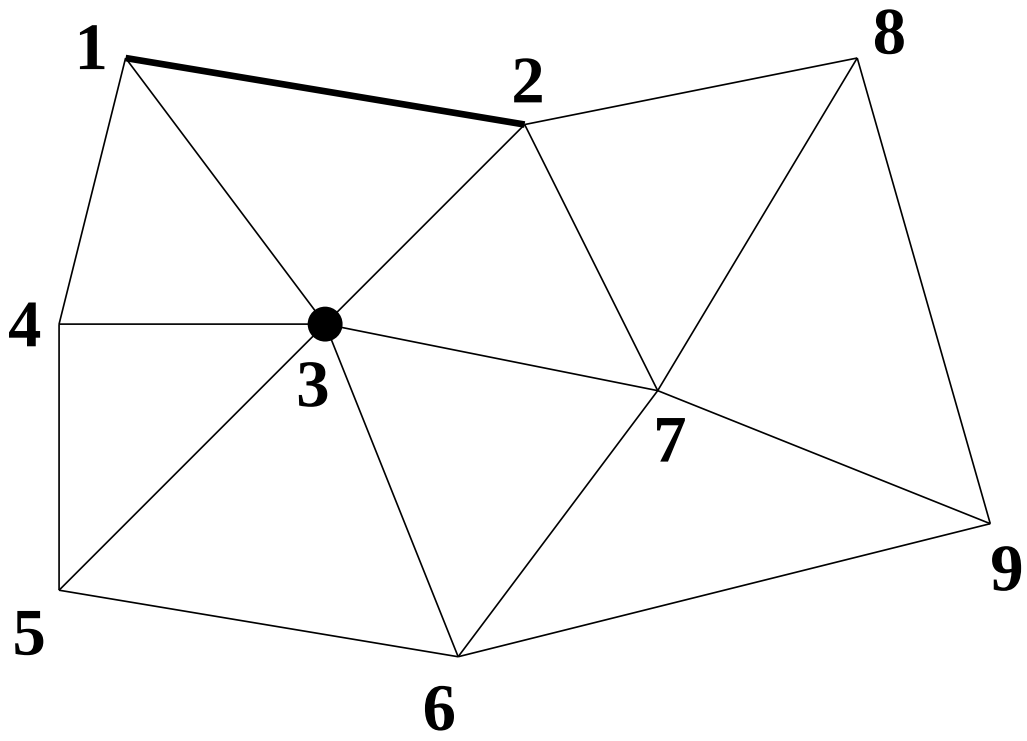
Sequential greedy matching



Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

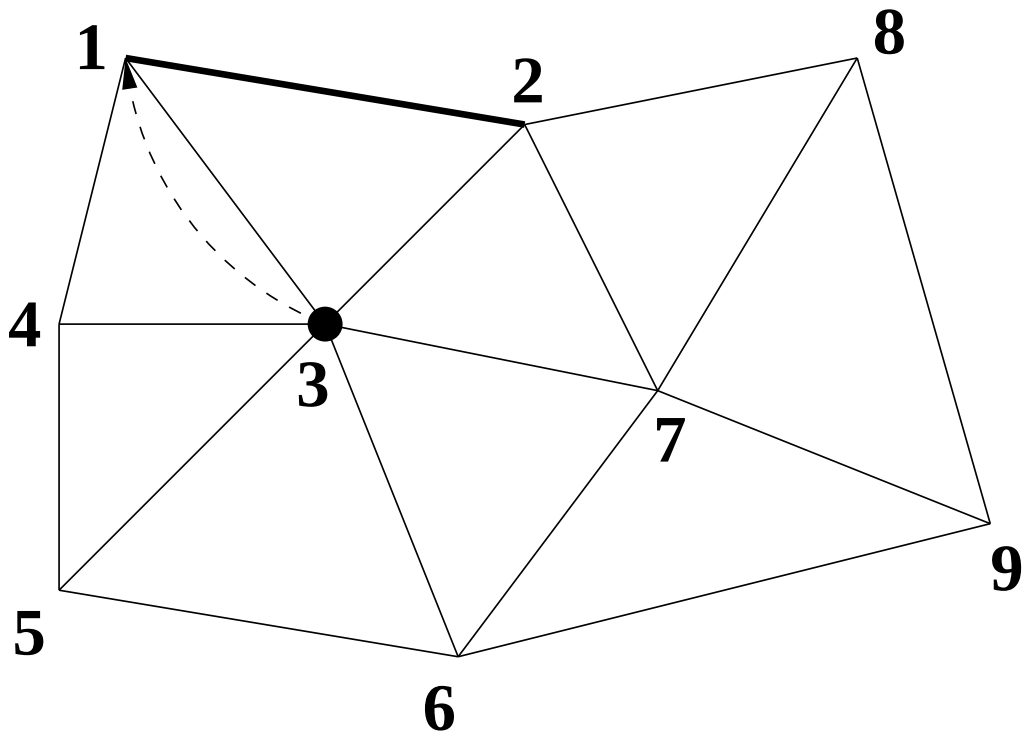
Sequential greedy matching



Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

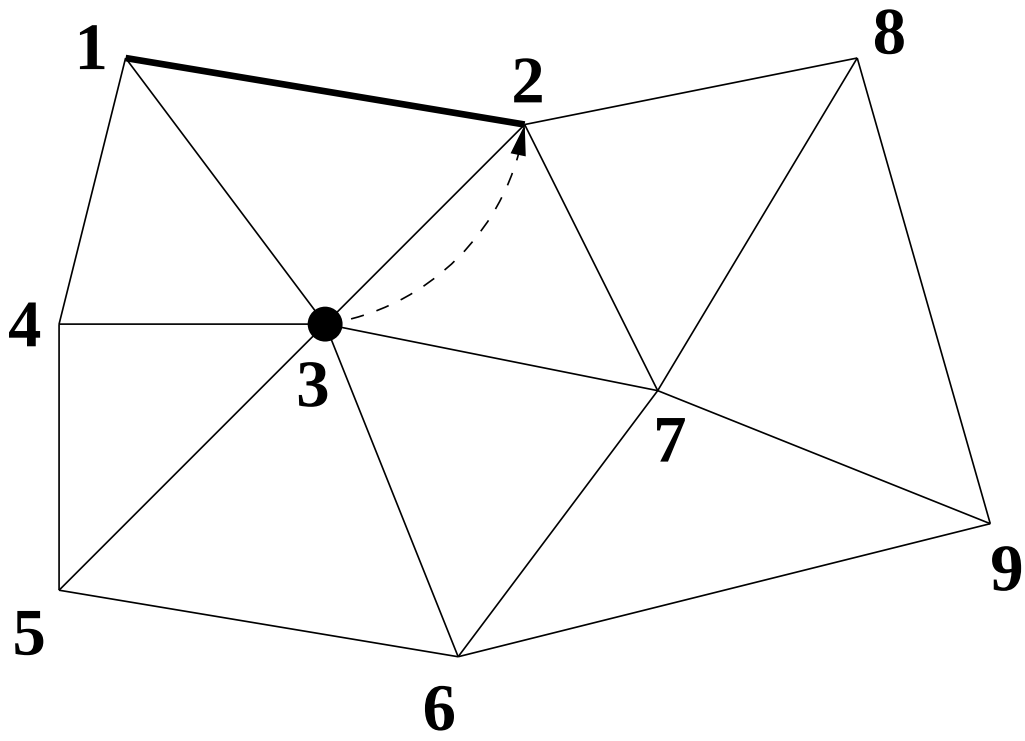
Sequential greedy matching



Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

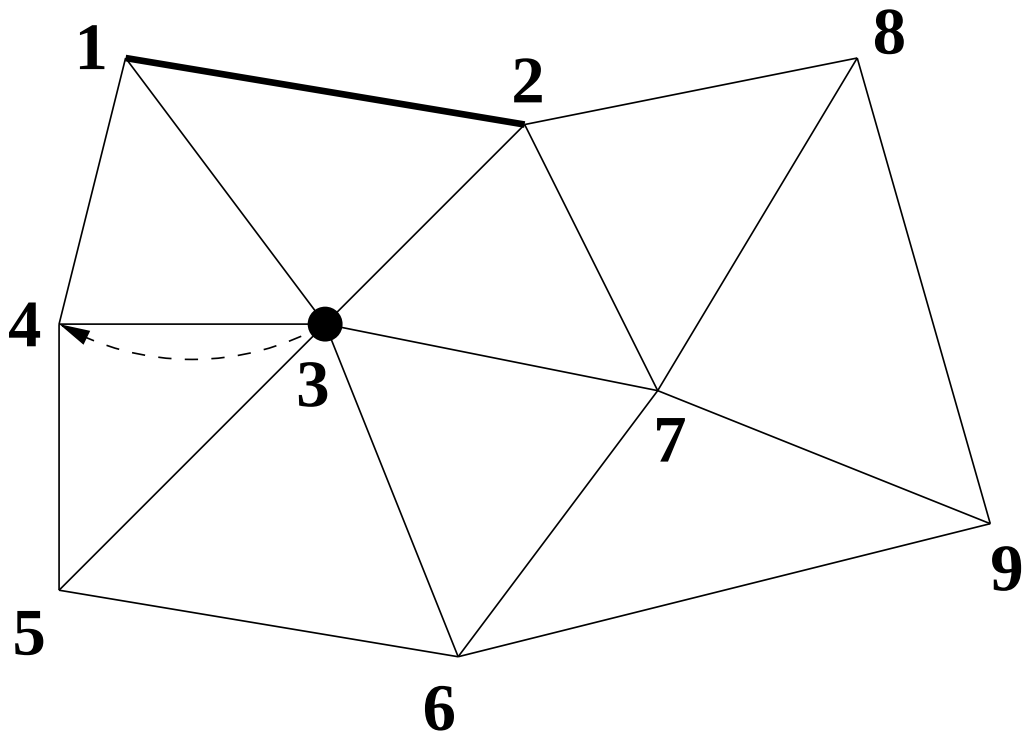
Sequential greedy matching



Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

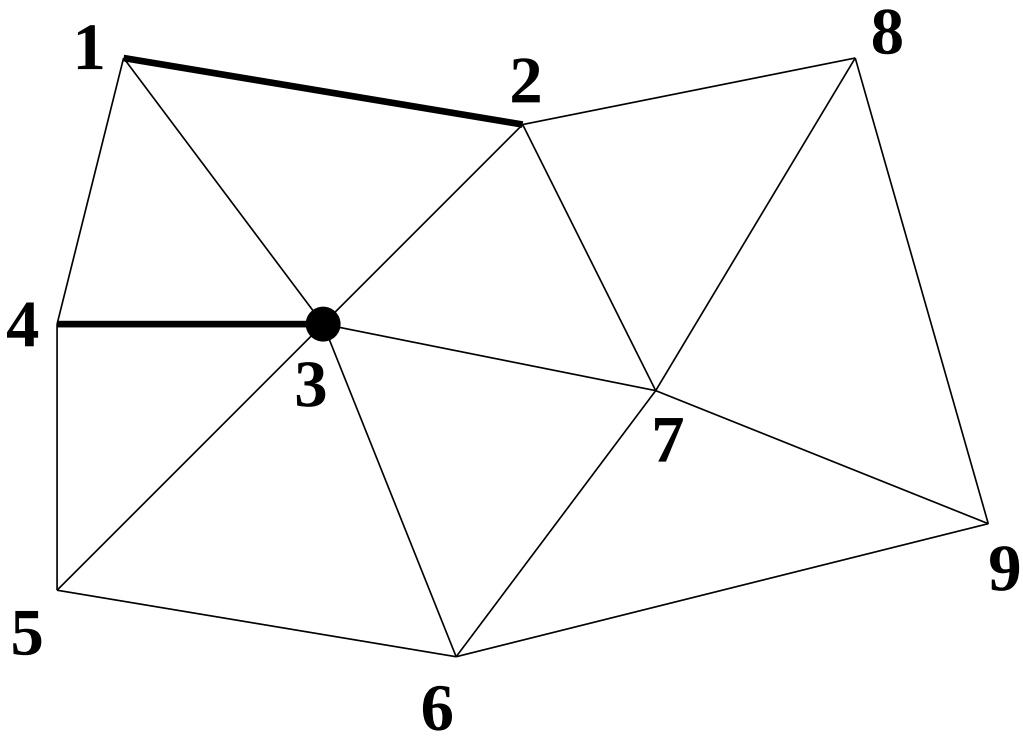
Sequential greedy matching



Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

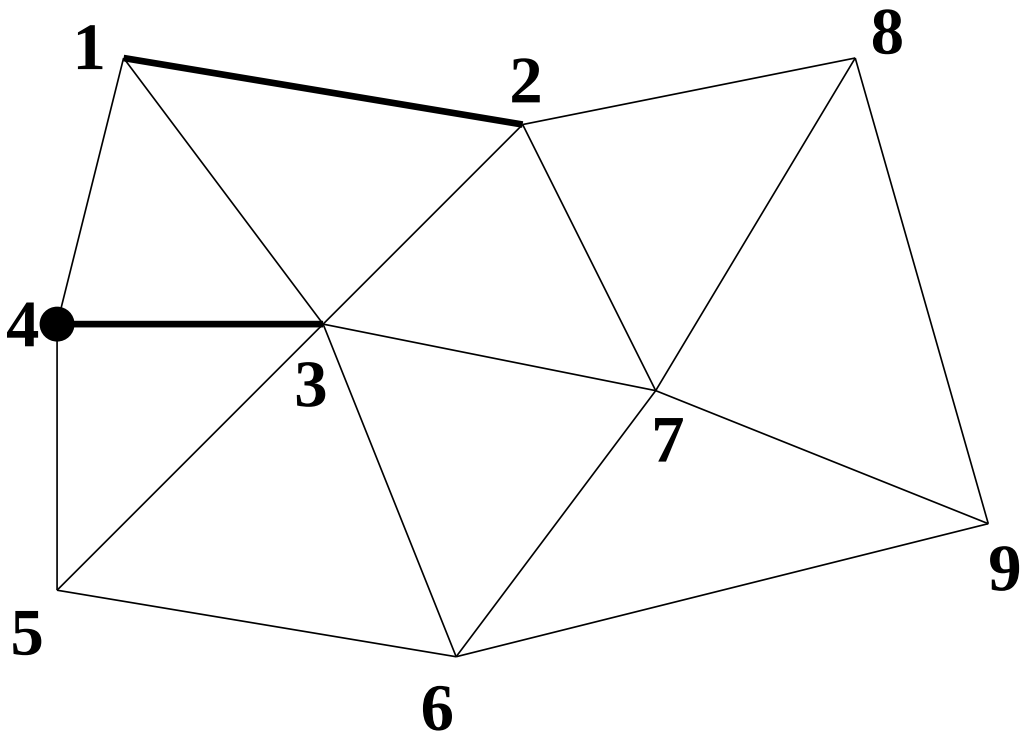
Sequential greedy matching



Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

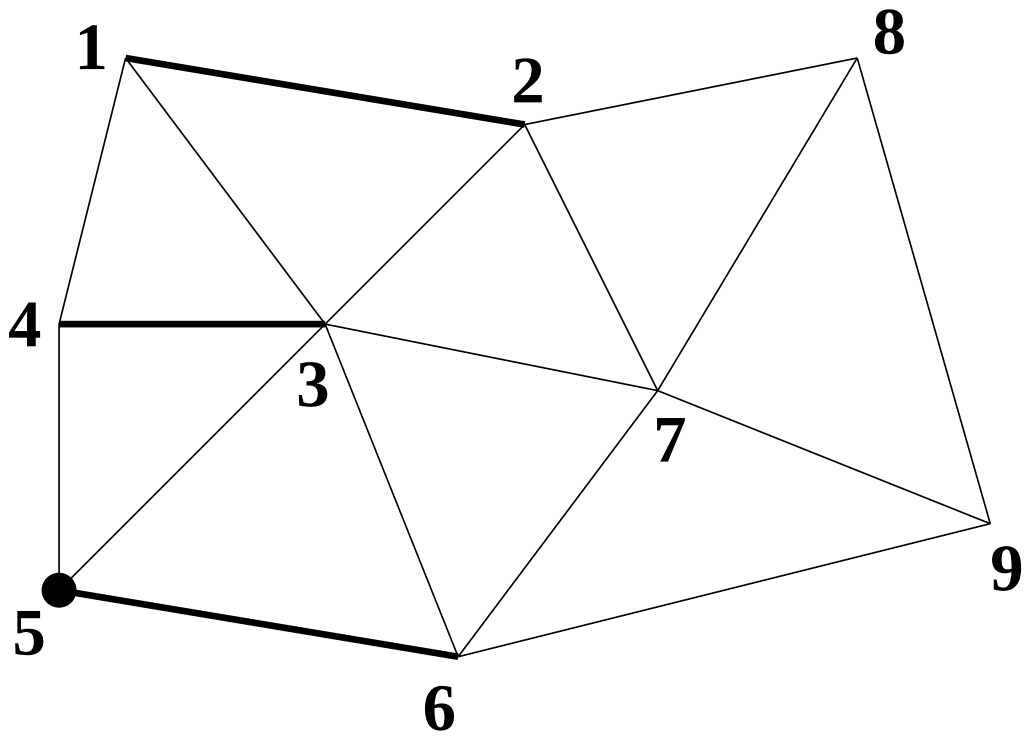
Sequential greedy matching



Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

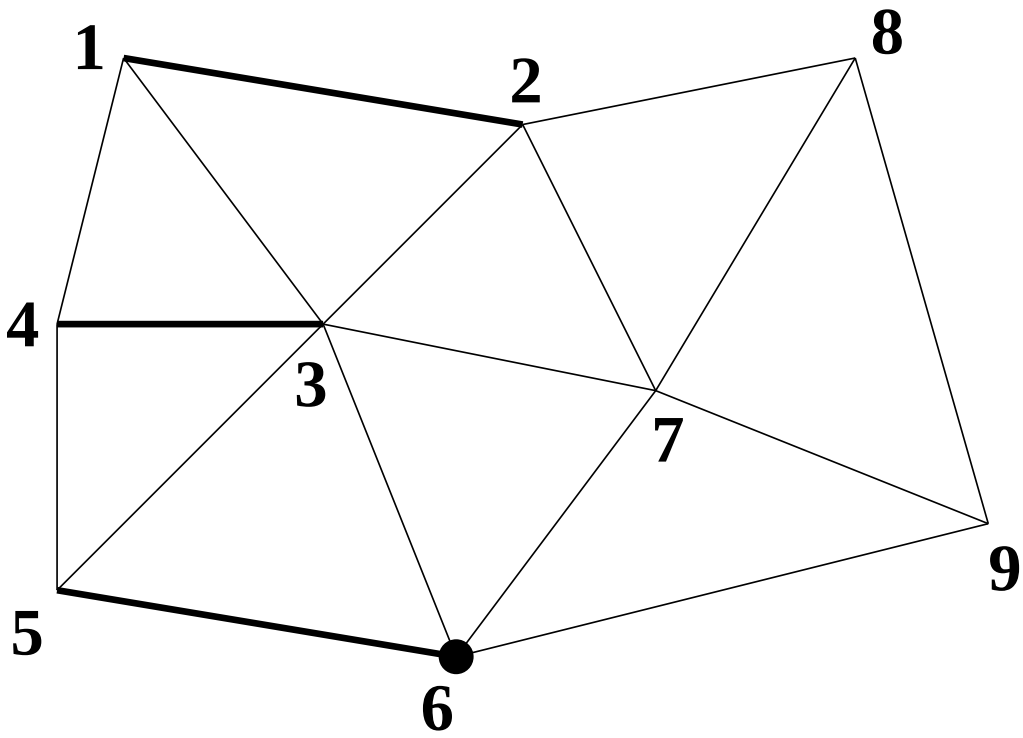
Sequential greedy matching



Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

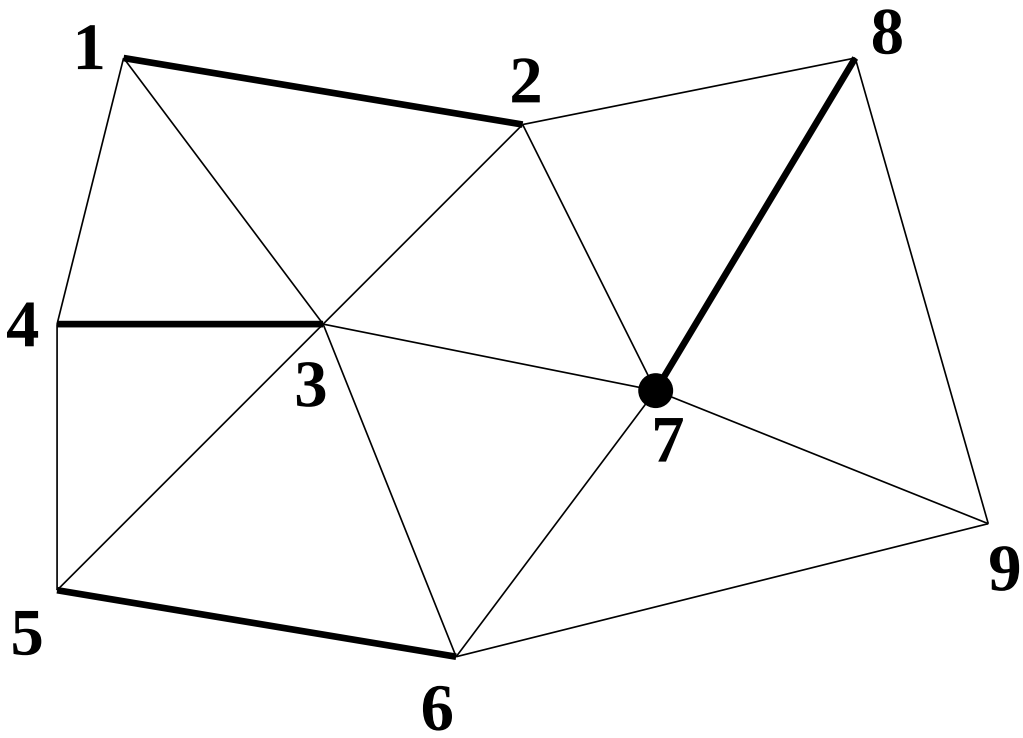
Sequential greedy matching



Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

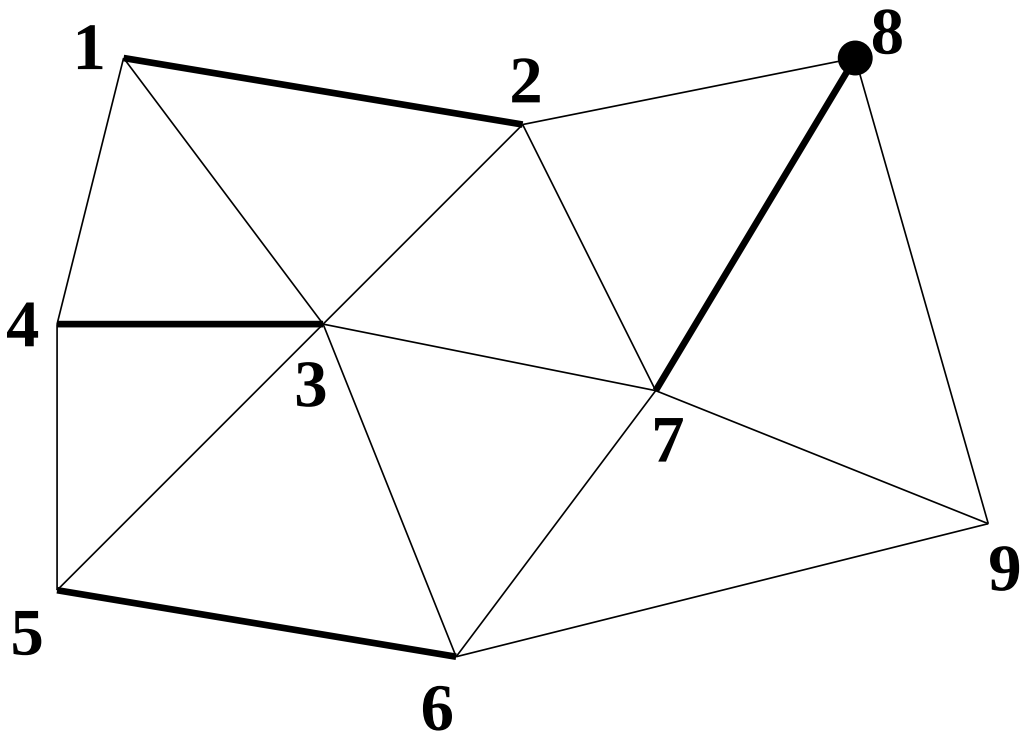
Sequential greedy matching



Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

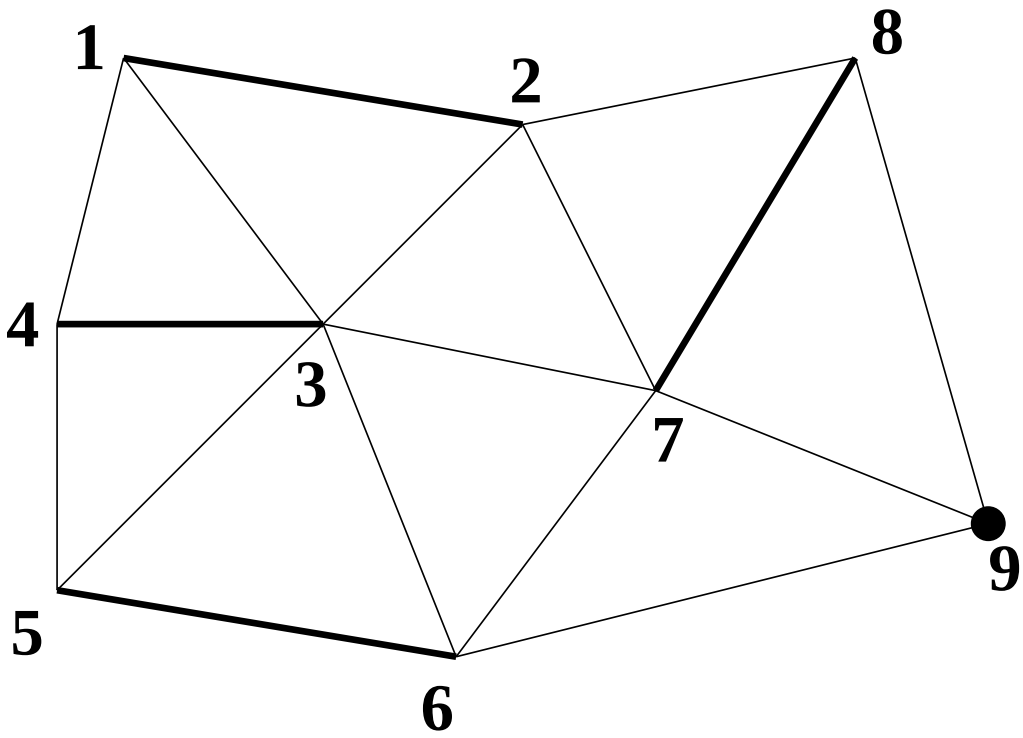
Sequential greedy matching



Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

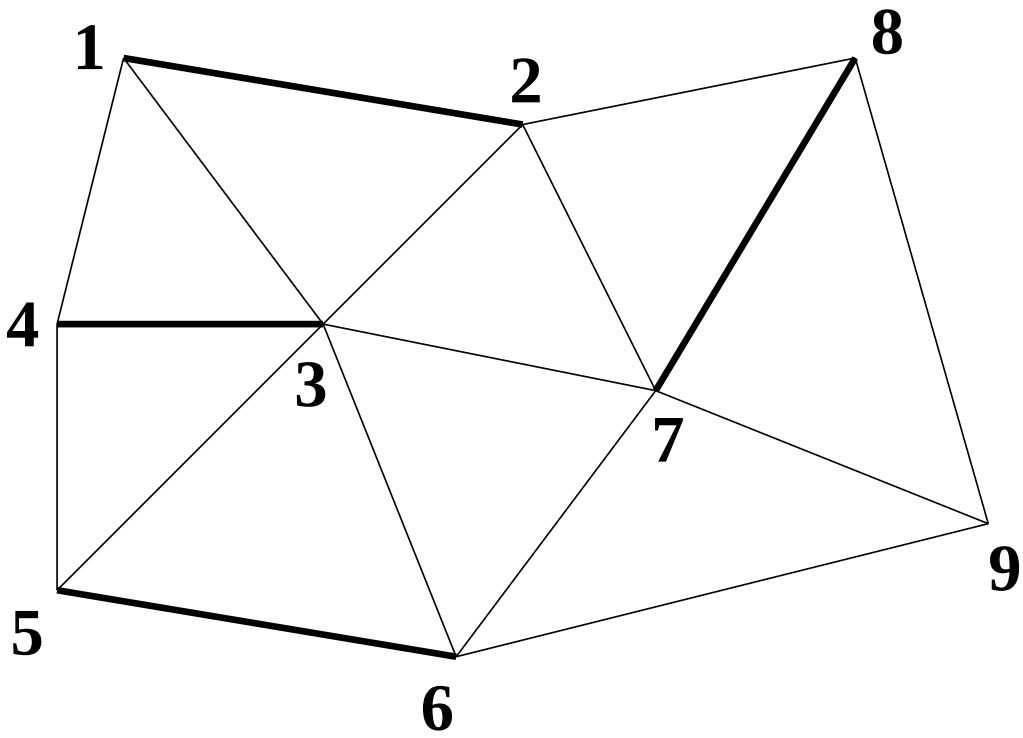
Sequential greedy matching



Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

Sequential greedy matching



Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

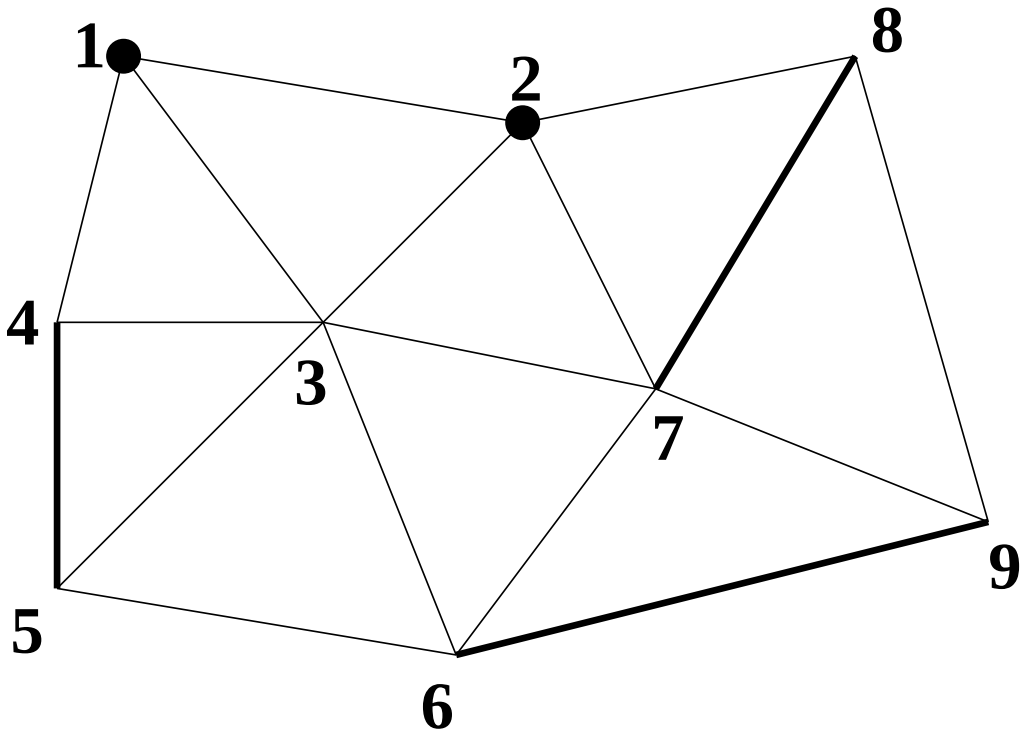
Universiteit Utrecht

Greedy 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.

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Parallel greedy matching: trouble

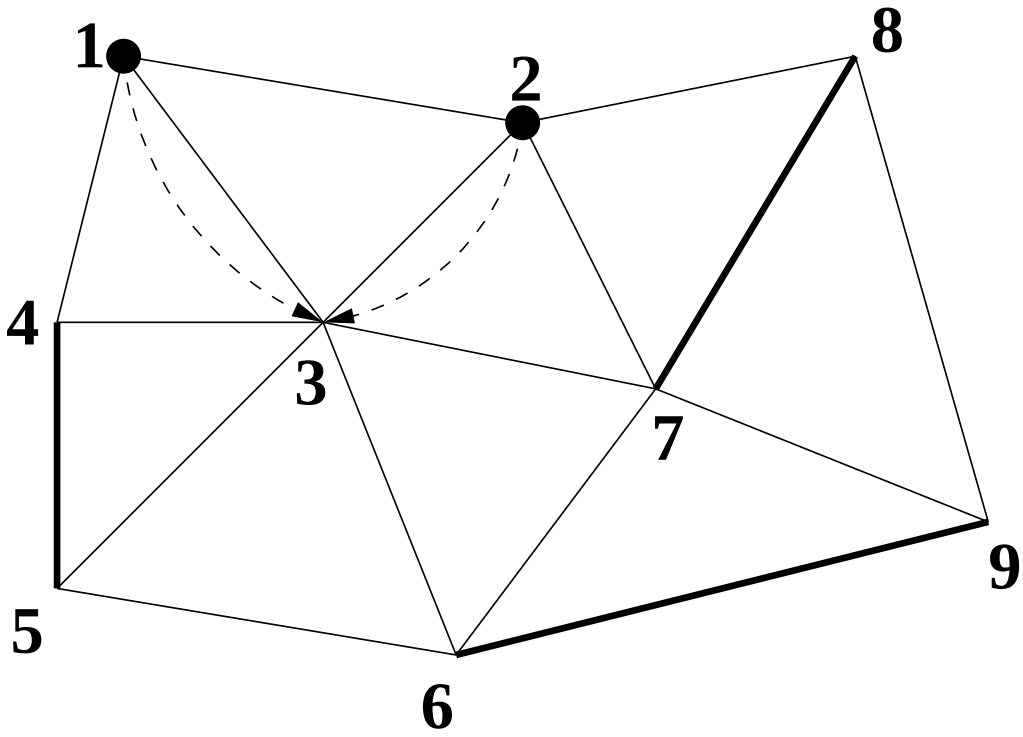


Suppose we match vertices simultaneously.

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

Parallel greedy matching: trouble

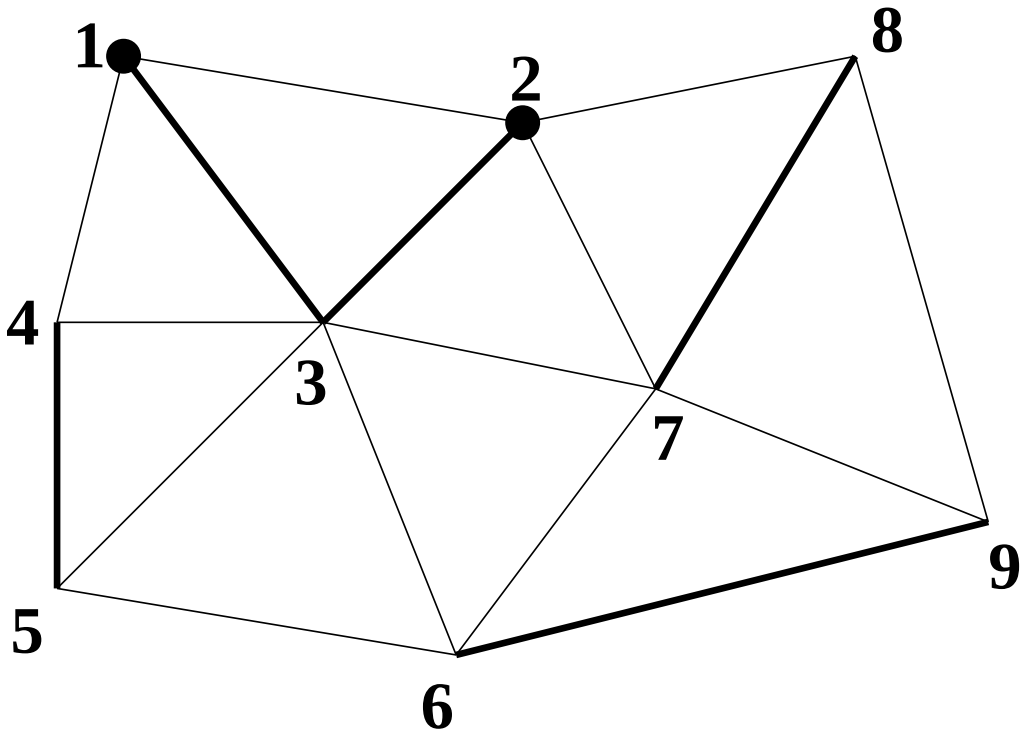


Two vertices each find an unmatched neighbour...

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

Parallel greedy matching: trouble



. . . but generate an invalid matching.

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

Dominant-edge algorithm

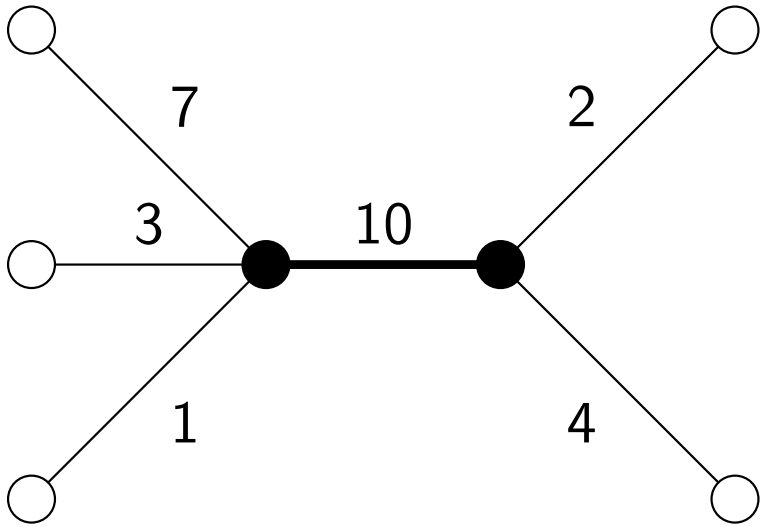
while $E \neq \emptyset$ **do**
  pick dominant edge $(v, w) \in E$
  $M := M \cup \{(v, w)\}$
  $E := E \setminus \{(x, y) \in E : x = v \vee x = w\}$
  $V := V \setminus \{v, w\}$
return M

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

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

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Dominant edge



Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

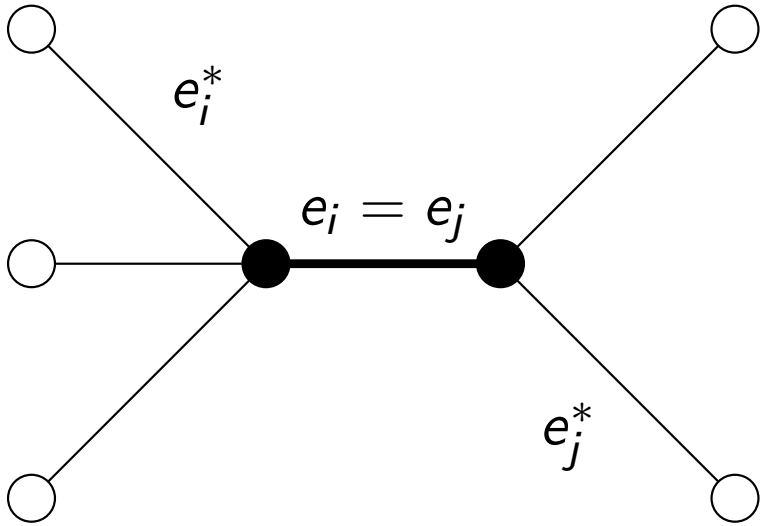
Universiteit Utrecht

Proof: algorithm is 1/2-approximation

- Let M be the matching produced by the dominant-edge algorithm.
- Let M^* be a maximum matching with weight ω_{optimal}.
- Let $M^* = \{e_0^*, \ldots, e_{k-1}^*\}$. For each edge $e_i^* \in M^*$, if $e_i^* \in M$, then let $e_i = e_i^*$, otherwise let e_i be the edge that removes e_i^* from E in the algorithm.
- It may happen that $e_i = e_j$ for $i \neq j$.
- $\omega(e_i) \geq \omega(e_i^*)$ for all i, since e_i is locally dominant in the algorithm and removes e_i^*, or $e_i = e_i^*$.

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Proof (cont'd)



- Every edge $e \in M$ can occur at most twice in the list of e_i's, since it can remove at most 2 edges in M^* from E.
- $$2\omega(M) \geq \sum_{i=0}^{k-1} \omega(e_i) \geq \sum_{i=0}^{k-1} \omega(e_i^*) = \omega_{\text{optimal}}$$
- Hence $\omega(M) \geq \omega_{\text{optimal}}/2$.

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

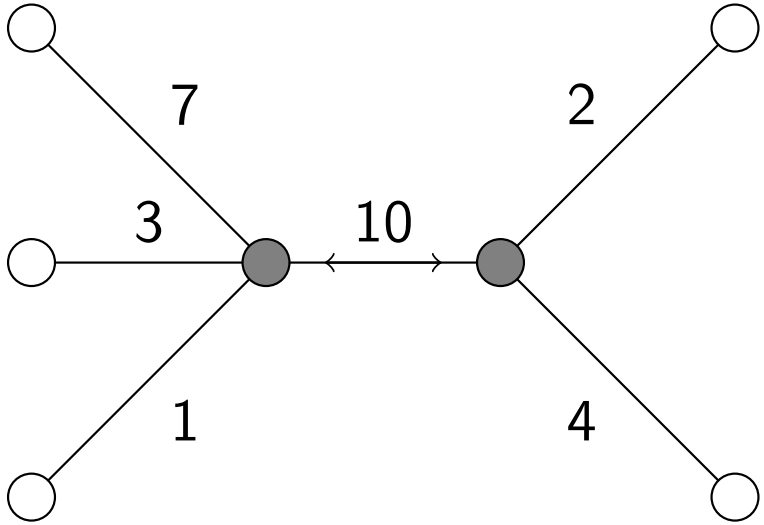
Sequential approximation algorithm: initialisation

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**
      $D := D \cup \{v, pref(v)\}$
      $M := M \cup \{(v, pref(v))\}$

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

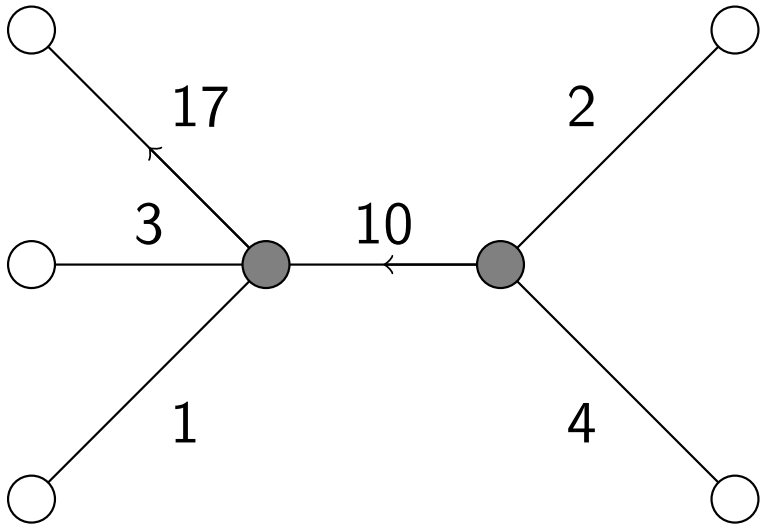
Mutual preferences



Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

Non-mutual preferences



Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

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

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Properties of the dominant-edge algorithm

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

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

- Dominant edge means mutual preference:

$$v = \mathit{pref}(w) \text{ and } w = \mathit{pref}(v).$$

- 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.

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

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.

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

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.

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

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.

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

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

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

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 $P(\phi(pref(v)))$

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

How to propose

Source: www.theguardian.com

proposal(v, w): v proposes to w

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

How not to propose

IJsselstein, The Netherlands, December 13, 2014
Source: ANP

accepted(v, w): w has accepted the proposal of v

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

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

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

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Parallel algorithm for $P(s)$: communication

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**
      $Adj_x := Adj_x \setminus \{v\}$
      $SetNewPreference(x, Adj_x, pref, V_s, D_s, M_s, \phi)$

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Termination

- The algorithm alternates supersteps of computation running the main loop and communication handling the received messages.
- The whole algorithm can terminate 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.

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

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 $\mathcal{O}(1)$).
- After every k ticks, everybody synchronises.

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

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: $k = 2l$?

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Sending messages

- The BSP system takes care that messages are sent automatically, in bulk. A useful BSPlib primitive for doing this is `bsp_send`.
- In the next superstep, all received messages are read (using `bsp_move`) and processed.
- Google's Pregel system (Malewicz 2010) follows this BSP style.

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Further improvement: edge-based (2D) distribution

Name	*SpMV* 1D	*SpMV* 2D	*Matching* 1D	*Matching* 2D
rw9 (af_shell10)	113	105	169	150
rw10 (boneS10)	150	145	228	189
rw11 (Stanford)	340	141	479	234
rw12 (gupta3)	710	44	1,305	61
rw13 (St_Berk.)	716	448	1,152	812
rw14 (F1)	139	130	148	139
sw1 (small world)	1,007	417	2,111	303
sw2	1,957	829	3,999	563
sw3	2,017	832	4,255	528
er1 (random)	1,856	1,133	1,788	1,157
er2	3,451	1,841	3,721	1,635
er3	5,476	2,569	6,350	1,990

Communication volume in sparse matrix–vector multiplication and Karp–Sipser matching.

Source: Patwary, Bisseling, Manne (2010).

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

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

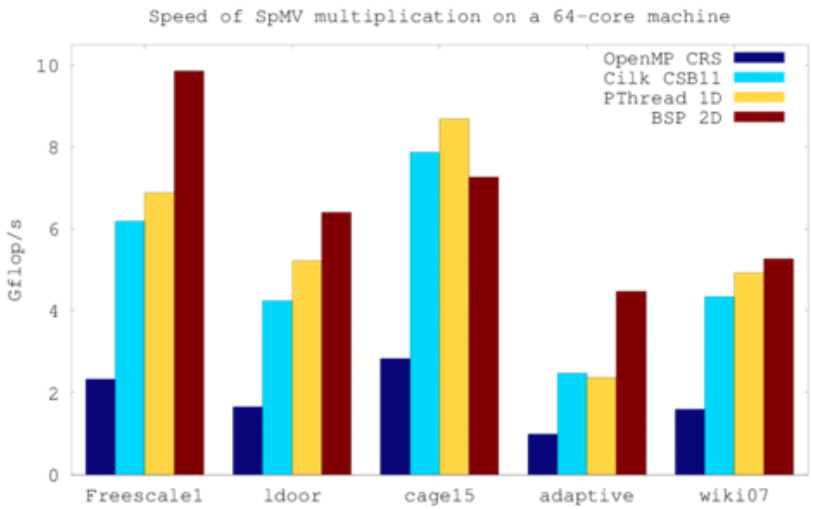
RECENT NEWS

- 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.com

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Universiteit Utrecht

Matching with MulticoreBSP

- BSP program can remain the same, giving portability.
- To exploit the ease of reading data in shared memory, the `bsp_direct_get` is available in MulticoreBSP.
- This performs the communication immediately and blocks until the communication has been carried out.
- Possible use: replace the set M_s of matched edges by a boolean array $matched_s$ marking the local matched vertices.
- This array can be read by all processors using `bsp_direct_get`, to replace the check $(x, pref(x)) \notin M_s$.

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion

Conclusions and outlook

- BSP is extremely suitable for parallel graph computations:
  - no need to worry about communication because we buffer messages until the next synchronisation;
  - no need for send-receive pairs;
  - BSP cost model gives synchronisation frequency;
  - correctness proof of algorithm becomes simpler;
  - no deadlock possible.

Outline
Matching
Introduction
Greedy matching
BSP matching
Approximation
BSP algorithm
Conclusion