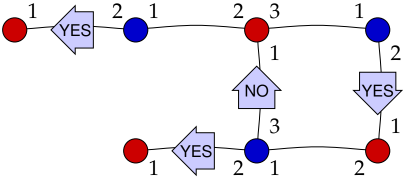
Stable matchings from the perspective of distributed algorithms

Jukka Suomela — HIIT, University of Helsinki, Finland

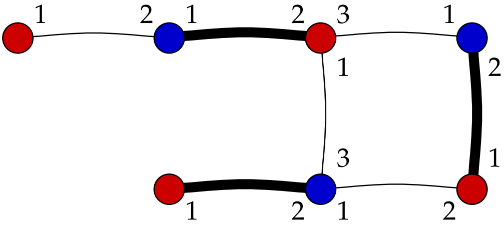
Joint work with Patrik Floréen,
Petteri Kaski, and Valentin Polishchuk



Zurich, 8 March 2010

Part I: Introduction

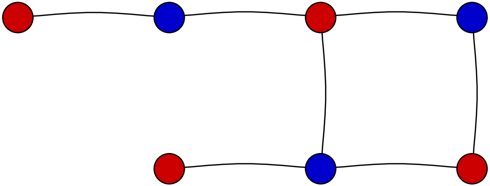
Stable matchings



☐ Stable marriage problem

Input: *bipartite graph* $\mathcal{G} = (R \cup B, E)$ ...

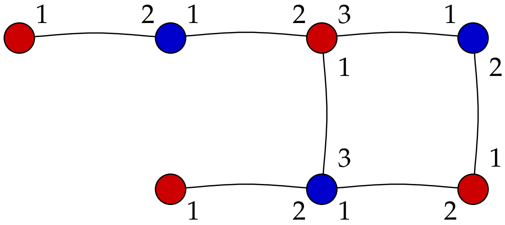
- R = red nodes
- B = blue nodes



Stable marriage problem

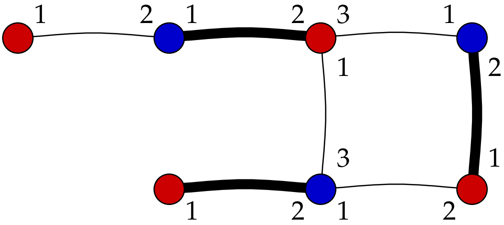
Input: *bipartite graph* $\mathcal{G} = (R \cup B, E)$ and *preferences*

- $1 =$ most preferred partner
- but anyone is better than no-one



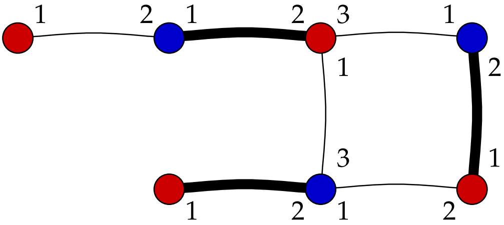
☐ Stable marriage problem

Output: a stable matching, i.e.,
a *matching* without *unstable edges*



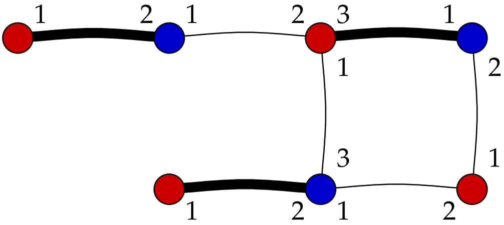
☐ Stable marriage problem

Matching: subset $M \subseteq E$ of edges such that each node adjacent to at most one edge in M



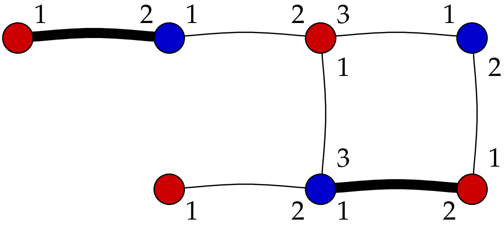
☐ Stable marriage problem

Matching: subset $M \subseteq E$ of edges such that each node adjacent to at most one edge in M



☐ Stable marriage problem

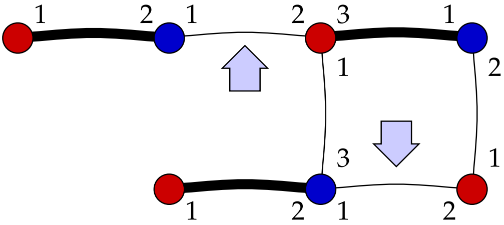
Matching: subset $M \subseteq E$ of edges such that each node adjacent to at most one edge in M



☐ Stable marriage problem

Unstable edge: edge $\{r, b\} \notin M$ such that

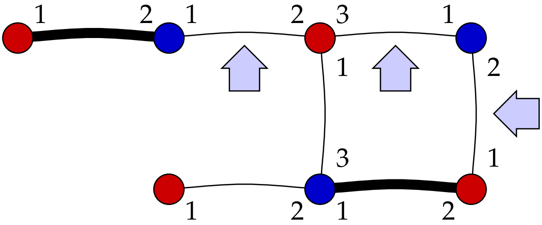
- r prefers b to r's current partner (if any)
- b prefers r to b's current partner (if any)



Stable marriage problem

Unstable edge: edge $\{r, b\} \notin M$ such that

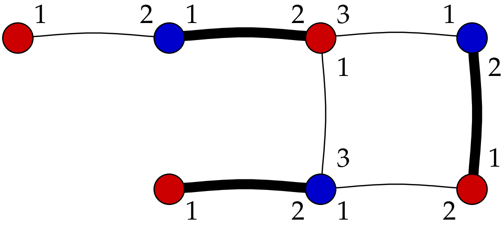
- r prefers b to r's current partner (if any)
- b prefers r to b's current partner (if any)



Stable marriage problem

Unstable edge: edge $\{r, b\} \notin M$ such that

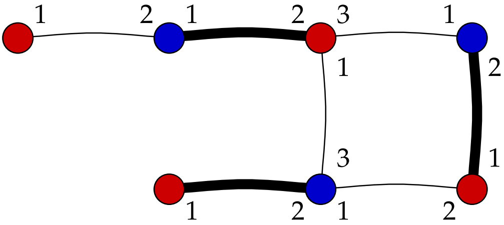
- r prefers b to r's current partner (if any)
- b prefers r to b's current partner (if any)



Stable marriage problem

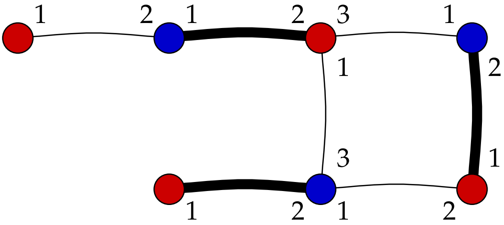
No unstable edges $\implies$ stable matching

- Does it always exist?
- How to find one?



☐ Part II:
Finding a stable matching

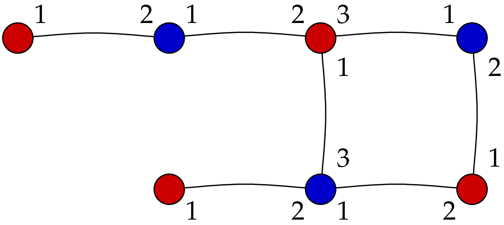
Gale–Shapley



Stable marriage problem

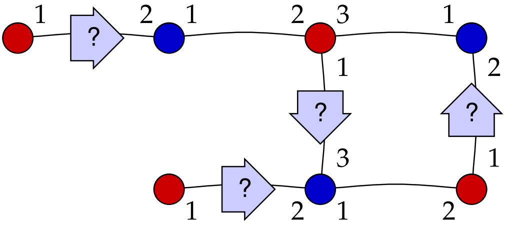
An adaptation of the Gale–Shapley algorithm (1962)

Begin with an empty matching



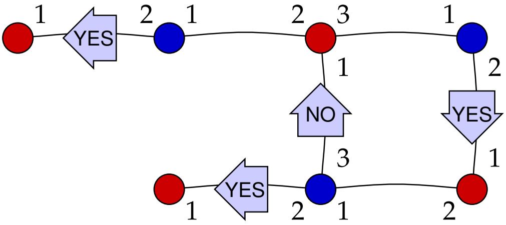
☐ Stable marriage problem

Unmatched red nodes send *proposals*
to their most-preferred neighbours



Stable marriage problem

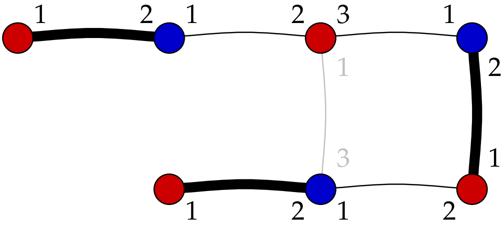
Blue nodes *accept* the best proposal



Stable marriage problem

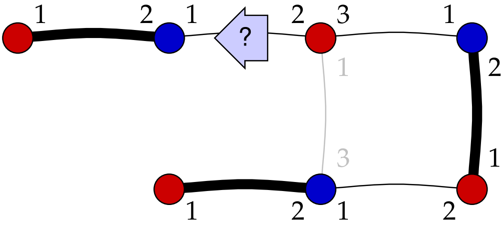
Blue nodes *accept* the best proposal

Remove rejected edges and repeat...



☐ Stable marriage problem

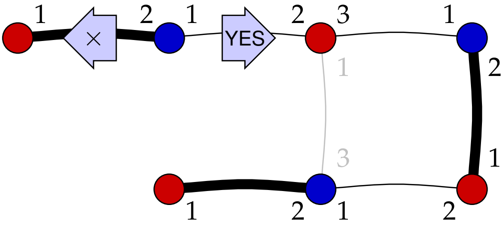
Unmatched red nodes send *proposals* to their most-preferred neighbours



Stable marriage problem

Blue nodes *accept* the best proposal

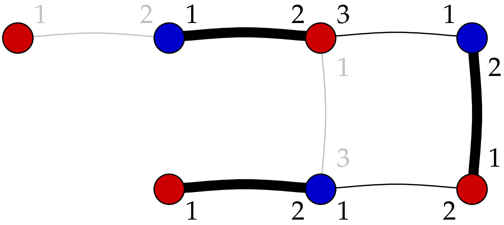
It is ok to change mind if a better proposal is received!



☐ Stable marriage problem

Blue nodes *accept* the best proposal

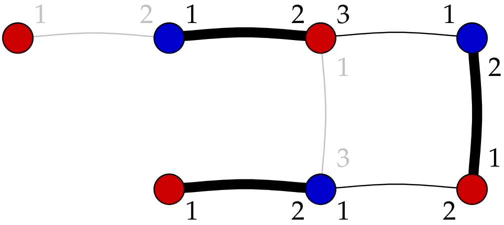
Remove rejected edges and repeat…



Stable marriage problem

Eventually each red node

- is matched, or
- has been rejected by all neighbours

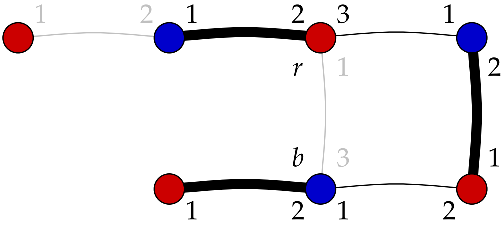


Stable marriage problem

Let $\{r,b\} \notin M$: (i) $b \in B$ rejected $r \in R$

$\implies$ b was matched to a more preferred neighbour

$\implies$ $\{r,b\}$ is not unstable

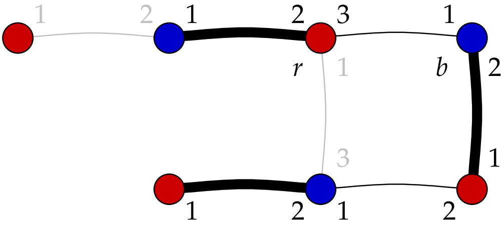


☐ Stable marriage problem

Let $\{r, b\} \notin M$: (ii) $r \in R$ did not ask $b \in B$

$\implies$ r is matched to a more preferred neighbour

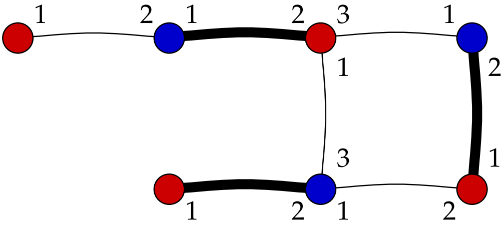
$\implies$ $\{r, b\}$ is not unstable



Stable marriage problem

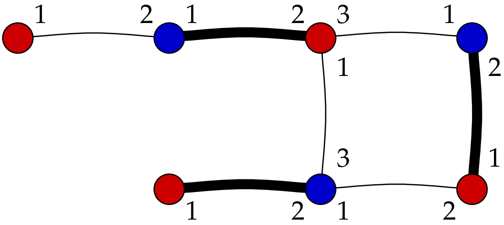
The Gale–Shapley algorithm finds a stable matching

Ok, that was published 48 years ago, more recent news?



Part III:
Stable matchings in a distributed setting

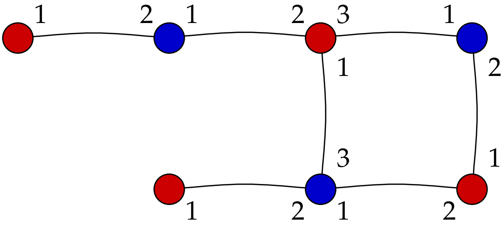
Stable matchings are unstable



Stable matchings in a distributed setting

Node = computer, edge = communication link

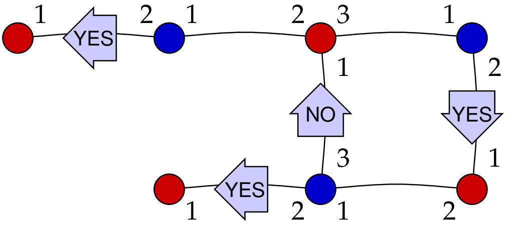
Efficient distributed algorithms for stable matchings?



Stable matchings in a distributed setting

The Gale–Shapley algorithm can be interpreted as a distributed algorithm

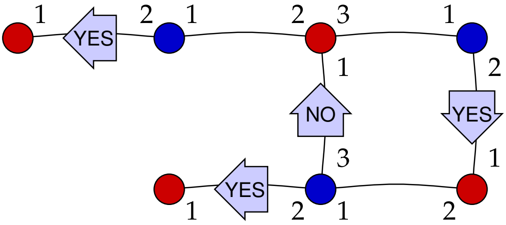
- proposal, acceptance, rejection: messages



Stable matchings in a distributed setting

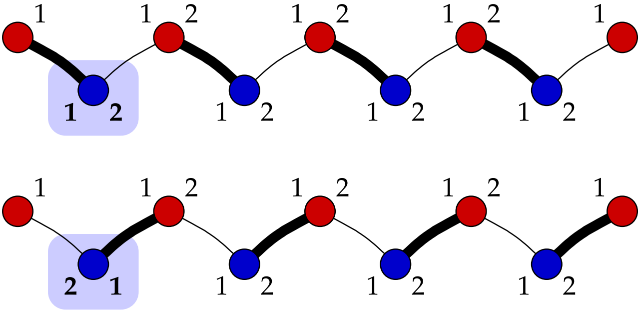
Many nice properties:

- small messages, deterministic
- unique identifiers not needed



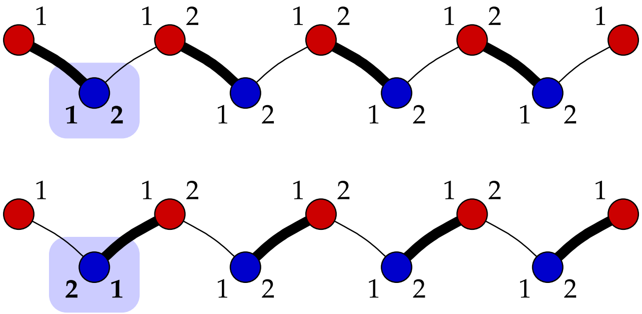
Stable matchings in a distributed setting

But Gale–Shapley isn't fast – it *cannot* be fast!



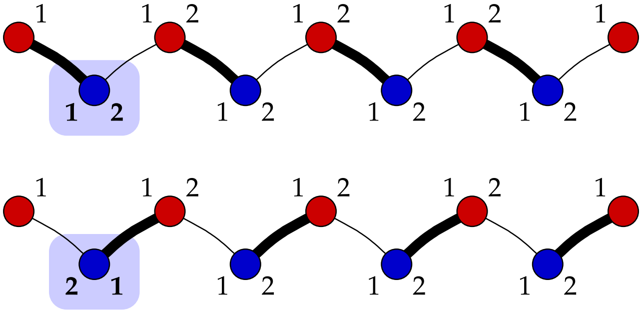
Stable matchings in a distributed setting

Solution depends on the input in distant parts of network
$\Longrightarrow$ worst-case running time $\Omega(\text{diameter})$



Stable matchings in a distributed setting

Stable matchings are unstable! Minor changes in input may require major changes in output



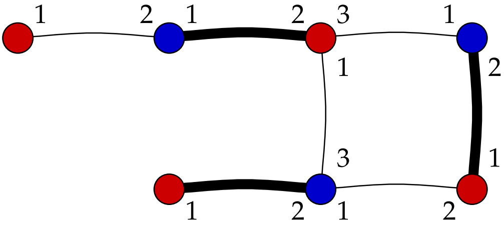
☐ Stable matchings in a distributed setting

Stable matchings are unstable! Minor changes in input may require major changes in output

- This isn't really what we would expect to happen, e.g., in real-world large scale social networks
- Very distant parts of the network should not affect my choices
- Are stable matchings the right problem to study? Matchings that are more *robust* and more *local*?

☐ Part IV: Almost stable matchings

Truncating Gale–Shapley



☐ Almost stable matchings

Our contribution: *asking the right questions*

- What if we allow a small fraction of unstable edges?
- What happens if we run Gale–Shapley for a small number of rounds?

Others have asked similar questions, too…

Almost stable matchings

What if we allow a small fraction of unstable edges?

- Biró et al. (2008): finding a *maximum* matching with few unstable edges is hard
- Finding *any* matching with few unstable edges?

Running Gale–Shapley for a small number of rounds?

- Quinn (1985): experimental work suggests that we get few unstable edges
- Any theoretical guarantees?

Almost stable matchings

Definition: A matching M is ϵ-stable
if there are at most $\epsilon|M|$ unstable edges

Main result: There is a distributed algorithm that
finds an ϵ-stable matching in $O(\Delta^2/\epsilon)$ rounds

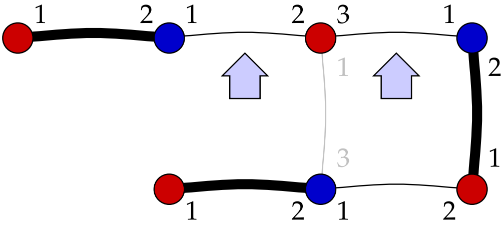
Algorithm: Just run the distributed version of
Gale–Shapley for that many steps!

$\Delta =$ maximum degree of $\mathcal{G}$

Almost stable matchings

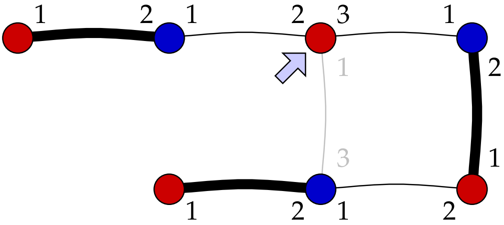
During the Gale–Shapley algorithm:

$\{r, b\} \in E$ is an unstable edge
$\implies$ r unmatched and r has not yet proposed b



Almost stable matchings

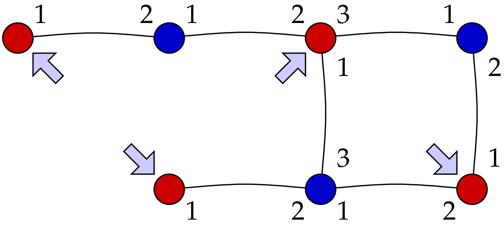
Key idea: define *total potential*
= number of unmatched red nodes with proposals left
= how much red nodes could "gain"
if we did not truncate Gale–Shapley



Almost stable matchings

Key idea: define *total potential*
$=$ number of unmatched red nodes with proposals left

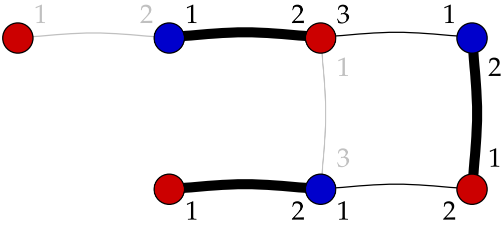
Initially high



Almost stable matchings

Key idea: define *total potential*
$=$ number of unmatched red nodes with proposals left

Zero if we run the full Gale–Shapley



☐ Almost stable matchings

- Potential is non-increasing: if a red node loses its partner, another red node gains a partner
- Assume that potential is α after round $k > 1$
  $\implies$ α nodes received 'no' or 'break' in round k
  $\implies$ at least α edges removed in round k
  $\implies$ at least $(k-1)\alpha$ edges removed in rounds $2, 3, \dots, k$
- At most $O(\Delta|M|)$ edges removed in total
  $\implies$ potential $O(\Delta|M|/k)$ after round k
  $\implies$ $O(\Delta^2|M|/k)$ unstable edges

☐ Almost stable matchings

Generalises to weighted matchings

Applications (in bipartite, bounded-degree graphs):

- Local $(2+\epsilon)$-approximation algorithm for maximum-weight matching
- Centralised randomised algorithm for estimating the size of a stable matching

(All stable matchings have the same size!)

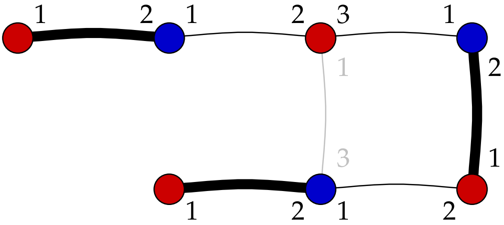
Almost stable matchings

But I think the most interesting observation is this:

- Almost stable matchings are a *local* problem (at least in bounded-degree graphs)
- There is a simple local algorithm that finds a *robust*, almost stable matching M
- The matching M can be easily maintained in a dynamic network, constructed by using an efficient self-stabilising algorithm, etc.

Almost stable matchings

Research question: are *almost stable matchings* the right concept when we try to understand and analyse real-world social networks, matching markets, etc.?



☐ Summary

Stable matching:

- global problem, any solution is unrobust

Almost stable matching:

- local problem, robust solutions exist

No new algorithms needed, just a new analysis
of the Gale–Shapley algorithm from 1962

http://www.cs.helsinki.fi/jukka.suomela/