Stable allocations and flows

Tamás Fleiner[1]

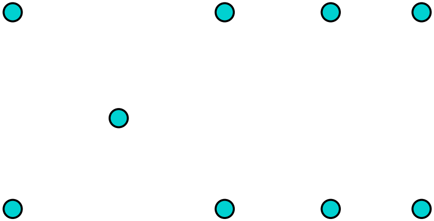
Summer School on
Matching Problems, Markets, and Mechanisms
26 June 2013, Budapest

[1]Budapest University of Technology and Economics

Stable matchings

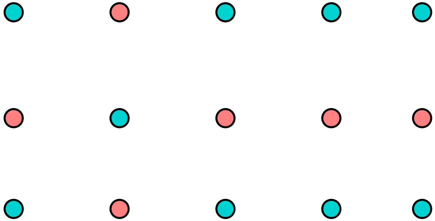
Model:

Stable matchings



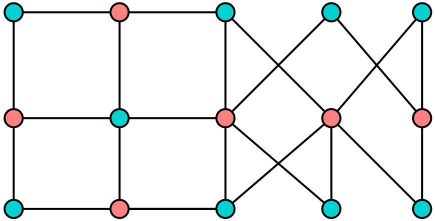
Model: **Boys**

Stable matchings



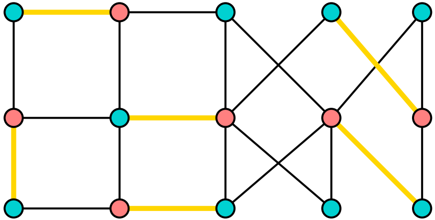
Model: **Boys** and **girls**

Stable matchings



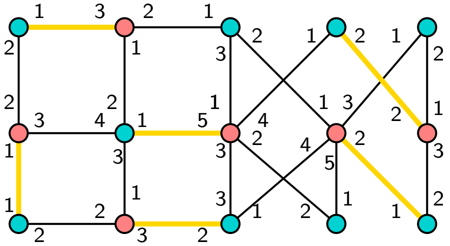
Model: **Boys** and **girls** with possible marriages are given.

Stable matchings



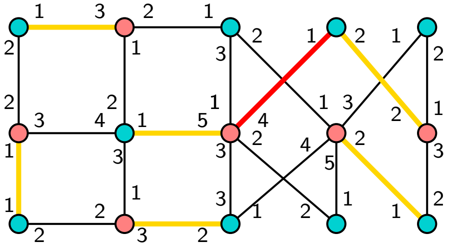
Model: **Boys** and **girls** with possible marriages are given.
Marriage scheme: **matching**.

Stable matchings



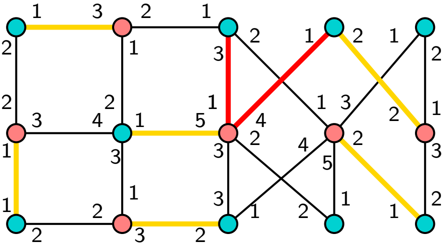
Model: Boys and girls with possible marriages are given.
Marriage scheme: matching. **Preferences** on possible partners.

Stable matchings



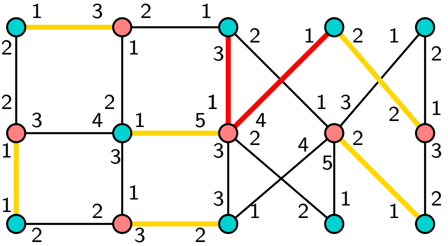
Model: **Boys** and **girls** with possible marriages are given.
Marriage scheme: **matching**. **Preferences** on possible partners.
Instability may occur along **blocking edges**.

Stable matchings



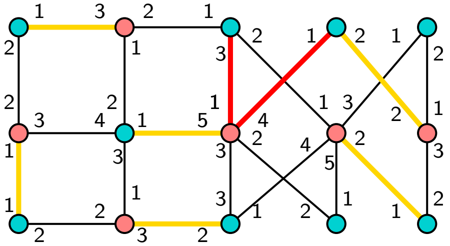
Model: **Boys** and **girls** with possible marriages are given.
Marriage scheme: **matching**. **Preferences** on possible partners.
Instability may occur along **blocking edges**.

Stable matchings



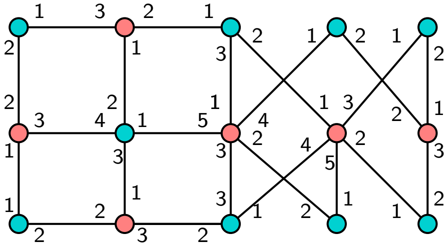
Model: **Boys** and **girls** with possible marriages are given.
Marriage scheme: **matching**. **Preferences** on possible partners.
Instability may occur along **blocking edges**.
A matching is **stable** if no blocking edge exists.

Stable matchings



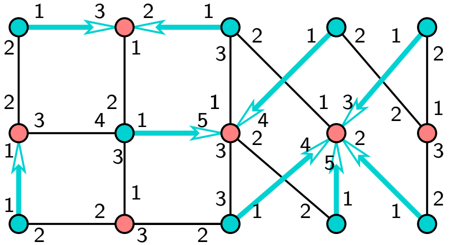
Model: **Boys** and **girls** with possible marriages are given.
Marriage scheme: **matching**. **Preferences** on possible partners.
Instability may occur along **blocking edges**.
A matching is **stable** if no blocking edge exists.
The proposal algorithm of Gale and Shapley always finds one:

Stable matchings



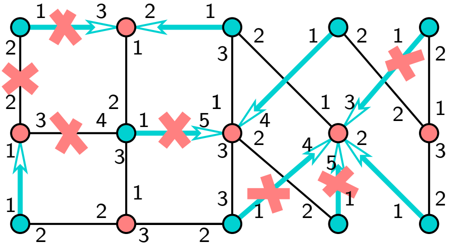
Model: **Boys** and **girls** with possible marriages are given.
Marriage scheme: **matching**. **Preferences** on possible partners.
Instability may occur along **blocking edges**.
A matching is **stable** if no blocking edge exists.
The proposal algorithm of Gale and Shapley always finds one:

Stable matchings



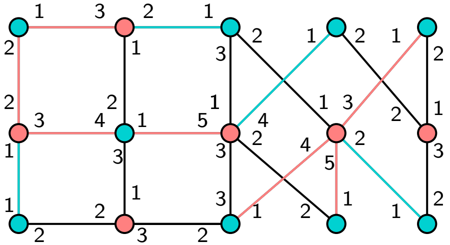
Model: **Boys** and **girls** with possible marriages are given.
Marriage scheme: **matching**. **Preferences** on possible partners.
Instability may occur along **blocking edges**.
A matching is **stable** if no blocking edge exists.
The proposal algorithm of Gale and Shapley always finds one:
Boys **propose** to best partners,

Stable matchings



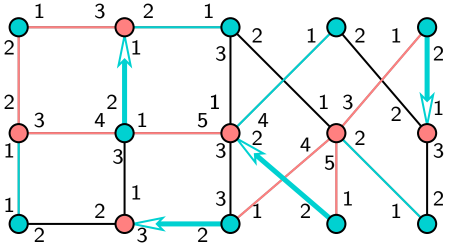
Model: **Boys** and **girls** with possible marriages are given.
Marriage scheme: **matching**. **Preferences** on possible partners.
Instability may occur along **blocking edges**.
A matching is **stable** if no blocking edge exists.
The proposal algorithm of Gale and Shapley always finds one:
Boys **propose** to best partners, girls **reject** boys with no chance.

Stable matchings



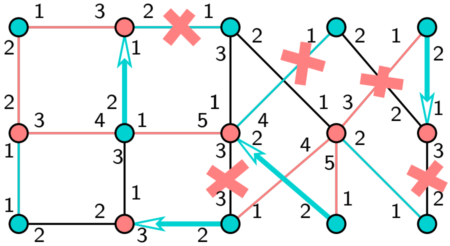
Model: **Boys** and **girls** with possible marriages are given.
Marriage scheme: **matching**. **Preferences** on possible partners.
Instability may occur along **blocking edges**.
A matching is **stable** if no blocking edge exists.
The proposal algorithm of Gale and Shapley always finds one:
Boys **propose** to best partners, girls **reject** boys with no chance.
We iterate:

Stable matchings



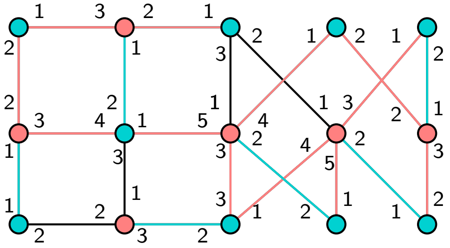
Model: Boys and girls with possible marriages are given.
Marriage scheme: matching. **Preferences** on possible partners.
Instability may occur along **blocking edges**.
A matching is **stable** if no blocking edge exists.
The proposal algorithm of Gale and Shapley always finds one:
Boys **propose** to best partners, girls **reject** boys with no chance.
We iterate: rejected boys **propose**

Stable matchings



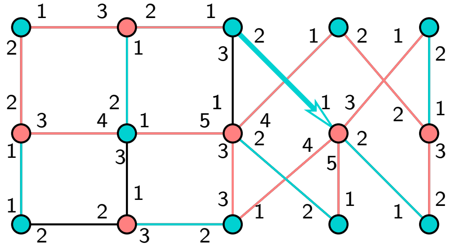
Model: **Boys** and **girls** with possible marriages are given.
Marriage scheme: **matching**. **Preferences** on possible partners.
Instability may occur along **blocking edges**.
A matching is **stable** if no blocking edge exists.
The proposal algorithm of Gale and Shapley always finds one:
Boys **propose** to best partners, girls **reject** boys with no chance.
We iterate: rejected boys **propose** and girls **reject** alternatingly.

Stable matchings



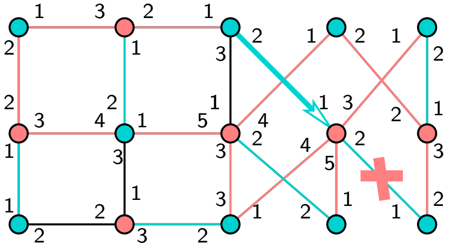
Model: **Boys** and **girls** with possible marriages are given.
Marriage scheme: **matching**. **Preferences** on possible partners.
Instability may occur along **blocking edges**.
A matching is **stable** if no blocking edge exists.
The proposal algorithm of Gale and Shapley always finds one:
Boys **propose** to best partners, girls **reject** boys with no chance.
We iterate: rejected boys **propose** and girls **reject** alternatingly.

Stable matchings



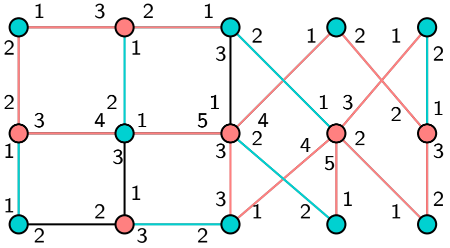
Model: **Boys** and **girls** with possible marriages are given.
Marriage scheme: **matching**. **Preferences** on possible partners.
Instability may occur along **blocking edges**.
A matching is **stable** if no blocking edge exists.
The proposal algorithm of Gale and Shapley always finds one:
Boys **propose** to best partners, girls **reject** boys with no chance.
We iterate: rejected boys **propose** and girls **reject** alternatingly.

Stable matchings



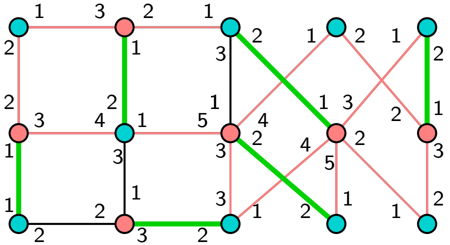
Model: **Boys** and **girls** with possible marriages are given.
Marriage scheme: **matching**. **Preferences** on possible partners.
Instability may occur along **blocking edges**.
A matching is **stable** if no blocking edge exists.
The proposal algorithm of Gale and Shapley always finds one:
Boys **propose** to best partners, girls **reject** boys with no chance.
We iterate: rejected boys **propose** and girls **reject** alternatingly.

Stable matchings



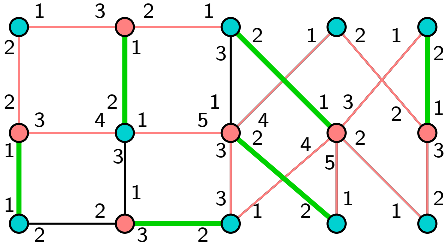
Model: **Boys** and **girls** with possible marriages are given.
Marriage scheme: **matching**. **Preferences** on possible partners.
Instability may occur along **blocking edges**.
A matching is **stable** if no blocking edge exists.
The proposal algorithm of Gale and Shapley always finds one:
Boys **propose** to best partners, girls **reject** boys with no chance.
We iterate: rejected boys **propose** and girls **reject** alternatingly.
When no boy proposes

Stable matchings



Model: **Boys** and **girls** with possible marriages are given.
Marriage scheme: **matching**. **Preferences** on possible partners.
Instability may occur along **blocking edges**.
A matching is **stable** if no blocking edge exists.
The proposal algorithm of Gale and Shapley always finds one:
Boys **propose** to best partners, girls **reject** boys with no chance.
We iterate: rejected boys **propose** and girls **reject** alternatingly.
When no boy proposes then we got a **stable matching**.

Stable matchings

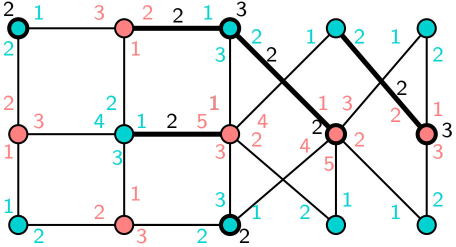


Model: **Boys** and **girls** with possible marriages are given.
Marriage scheme: **matching**. **Preferences** on possible partners.
Instability may occur along **blocking edges**.
A matching is **stable** if no blocking edge exists.
The proposal algorithm of Gale and Shapley always finds one:
Boys **propose** to best partners, girls **reject** boys with no chance.
We iterate: rejected boys **propose** and girls **reject** alternatingly.
When no boy proposes then we got a **stable matching**.
Man-optimality: each boy gets the best stable partner.

Stable allocations and properties

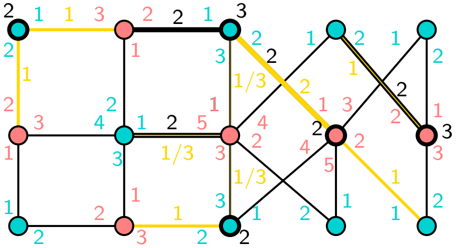
Extension of the model: capacities for vxs and edges (partnerships).

Stable allocations and properties



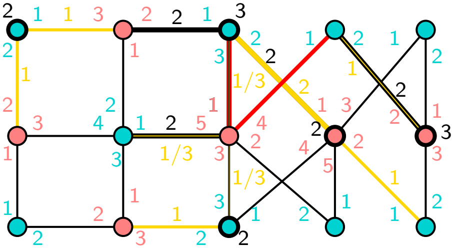
Extension of the model: capacities for vxs and edges (partnerships).

Stable allocations and properties



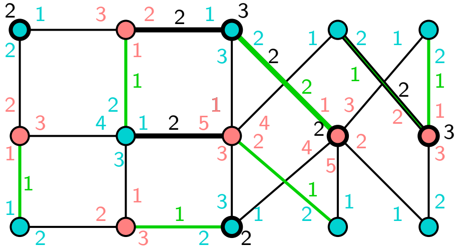
Extension of the model: capacities for vxs and edges (partnerships).
An **allocation** an assignment of intensities to edges st capacities of edges and vxs are observed.

Stable allocations and properties



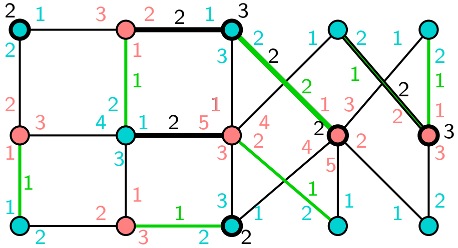
Extension of the model: capacities for vxs and edges (partnerships). An **allocation** an assignment of intensities to edges st capacities of edges and vxs are observed. An edge is **blocking** the allocation if both end vertices prefer to increase the intensity of the partnership.

Stable allocations and properties



Extension of the model: capacities for vxs and edges (partnerships). An allocation an assignment of intensities to edges st capacities of edges and vxs are observed. An edge is **blocking** the allocation if both end vertices prefer to increase the intensity of the partnership. An allocation is stable if no blocking edge occurs.

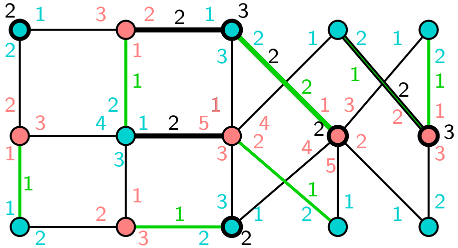
Stable allocations and properties



Extension of the model: capacities for vxs and edges (partnerships). An **allocation** an assignment of intensities to edges st capacities of edges and vxs are observed. An edge is **blocking** the allocation if both end vertices prefer to increase the intensity of the partnership. An allocation is stable if no blocking edge occurs.

Thm (Baïou-Balinski) A stable allocation always exists.

Stable allocations and properties

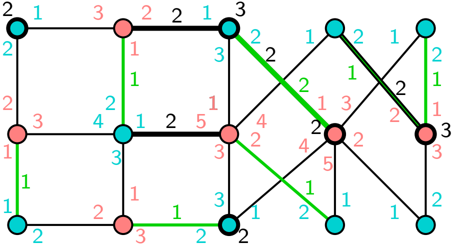


Extension of the model: capacities for vxs and edges (partnerships). An **allocation** an assignment of intensities to edges st capacities of edges and vxs are observed. An edge is **blocking** the allocation if both end vertices prefer to increase the intensity of the partnership. An allocation is stable if no blocking edge occurs.

Thm (Baïou-Balinski) A stable allocation always exists.

Extended GS algorithm finds a “man optimal” stable allocation.

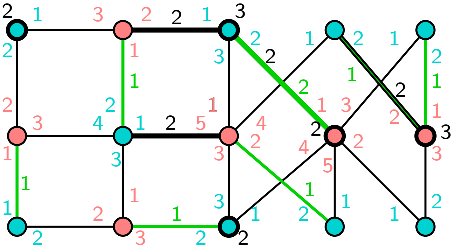
Stable allocations and properties



Extension of the model: capacities for vxs and edges (partnerships). An **allocation** an assignment of intensities to edges st capacities of edges and vxs are observed. An edge is **blocking** the allocation if both end vertices prefer to increase the intensity of the partnership. An allocation is stable if no blocking edge occurs.

Thm (Baïou-Balinski) A stable allocation always exists.
Extended GS algorithm finds a "man optimal" stable allocation.
Lattice property: if boys freely select from two stable alloc's then a stable alloc is created where girls get their worse choice.

Stable allocations and properties



Extension of the model: capacities for vxs and edges (partnerships). An **allocation** an assignment of intensities to edges st capacities of edges and vxs are observed. An edge is **blocking** the allocation if both end vertices prefer to increase the intensity of the partnership. An allocation is stable if no blocking edge occurs.

Thm (Baïou-Balinski) A stable allocation always exists.

Extended GS algorithm finds a "man optimal" stable allocation.

Lattice property: if boys freely select from two stable alloc's then a stable alloc is created where girls get their worse choice.

If someone is left with free capacity in some stable alloc then each stable alloc is the same for him/her.

Stable flows

Network flows: generalization of bipartite matching.

Stable flows

Network flows: generalization of bipartite matching.
Allocation model: (nonintegral) stable matching with capacities.

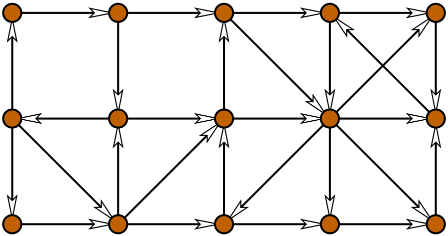
Stable flows

Network flows: generalization of bipartite matching.
Allocation model: (nonintegral) stable matching with capacities.
Stability for network flows??

Stable flows

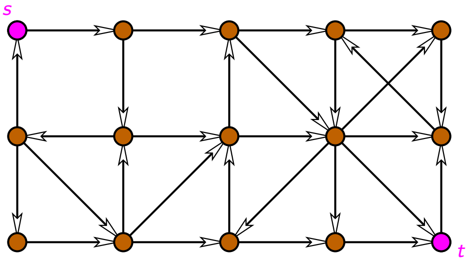
Model:

Stable flows



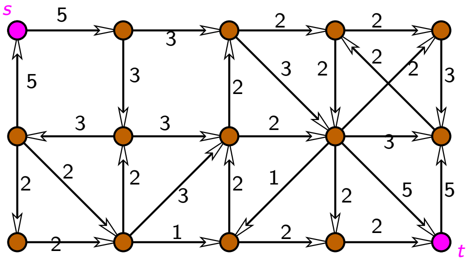
Model: Digraph

Stable flows



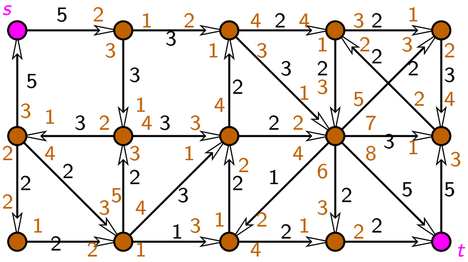
Model: Digraph, terminals s, t

Stable flows



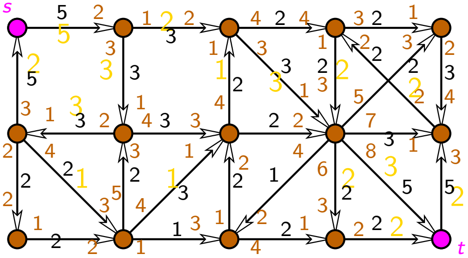
Model: Digraph, terminals s, t, capacities on the arcs

Stable flows



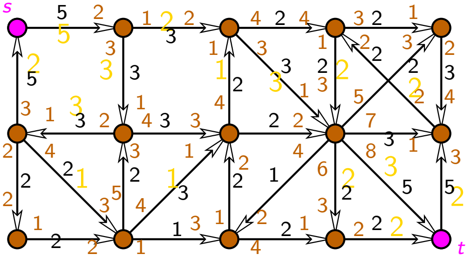
Model: Digraph, terminals s, t, capacities on the arcs and **preferences** on the arcs of the nonterminals.

Stable flows



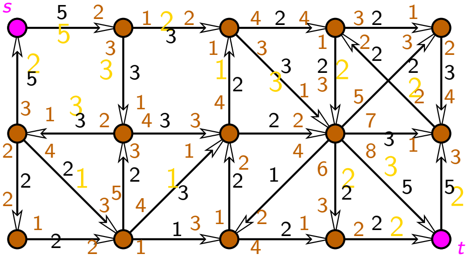
Model: Digraph, terminals s, t, capacities on the arcs and **preferences** on the arcs of the nonterminals. A **flow** is a function on the arcs obeying the capacity constraints and the Kirchhoff rule.

Stable flows



Model: Digraph, terminals s, t, capacities on the arcs and **preferences** on the arcs of the nonterminals. A **flow** is a function on the arcs obeying the capacity constraints and the Kirchhoff rule. Vxs are trading and each strives to achieve a best trading position.

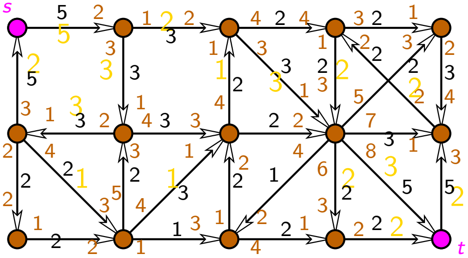
Stable flows



Model: Digraph, terminals s, t, capacities on the arcs and **preferences** on the arcs of the nonterminals. A **flow** is a function on the arcs obeying the capacity constraints and the Kirchhoff rule. Vxs are trading and each strives to achieve a best trading position.

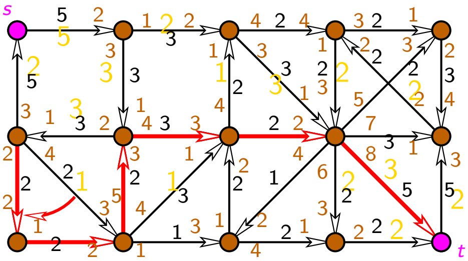
Instability: (1) some vx can increase its throughput or

Stable flows



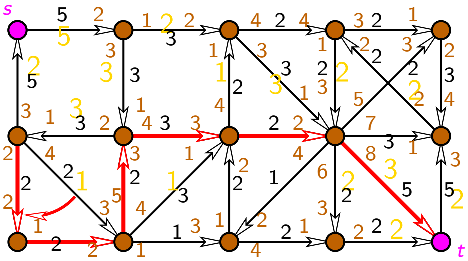
Model: Digraph, terminals s, t, capacities on the arcs and **preferences** on the arcs of the nonterminals. A **flow** is a function on the arcs obeying the capacity constraints and the Kirchhoff rule. Vxs are trading and each strives to achieve a best trading position. Instability: (1) some vx can increase its throughput or (2) a vx can move some flow from a one arc to a better one.

Stable flows



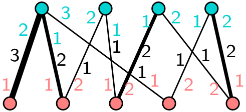
Model: Digraph, terminals s, t, capacities on the arcs and **preferences** on the arcs of the nonterminals. A **flow** is a function on the arcs obeying the capacity constraints and the Kirchhoff rule. Vxs are trading and each strives to achieve a best trading position.
Instability: (1) some vx can increase its throughput or (2) a vx can move some flow from a one arc to a better one.
Formally: a flow is stable if no **blocking walk** exists, i.e. a directed walk on unsaturated arcs such that both ends of the walk is either a terminal or can improve its position by moving some flow from a worse arc onto the walk.

Stable flows



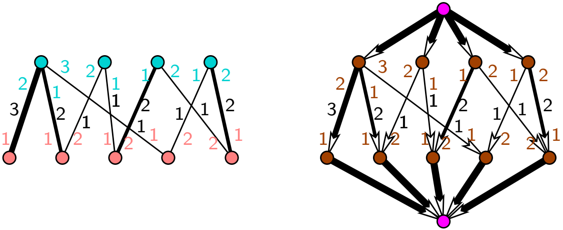
Model: Digraph, terminals *s*, *t*, capacities on the arcs and **preferences** on the arcs of the nonterminals. A **flow** is a function on the arcs obeying the capacity constraints and the Kirchhoff rule. Vxs are trading and each strives to achieve a best trading position.
Instability: (1) some vx can increase its throughput or (2) a vx can move some flow from a one arc to a better one.
Formally: a flow is stable if no **blocking walk** exists, i.e. a directed walk on unsaturated arcs such that both ends of the walk is either a terminal or can improve its position by moving some flow from a worse arc onto the walk. **Thm** A stable flow always exists.

Stable allocations as stable flows



The stable allocation problem is a special case of the stable flow problem.

Stable allocations as stable flows

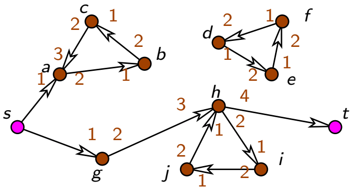


The stable allocation problem is a special case of the stable flow problem.

Introduce new terminals s and t and high capacity arcs from s to one color class, and to t from the other color class. Orient all edges from one color class to the other one and keep preferences. (...)

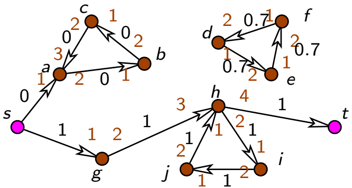
This way any stable allocation can be naturally transformed into a stable flow and any stable flow induces a stable allocation on the original instance.

An example



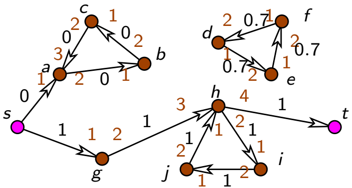
What is a stable allocation here? (All capacities are 1.)

An example



What is a stable allocation here? (All capacities are 1.)

An example



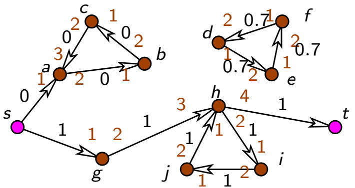
What is a stable allocation here? (All capacities are 1.)

Directed cycle *abc* cannot carry any flow as otherwise *sa* would be a blocking path.

Directed cycle *def* can carry any flow between 0 and 1.

Directed cycle *hij* must carry unit flow as otherwise closed walk *hij* would be blocking.

An example



What is a stable allocation here? (All capacities are 1.)

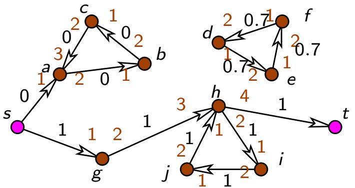
Directed cycle *abc* cannot carry any flow as otherwise *sa* would be a blocking path.

Directed cycle *def* can carry any flow between 0 and 1.

Directed cycle *hij* must carry unit flow as otherwise closed walk *hij* would be blocking.

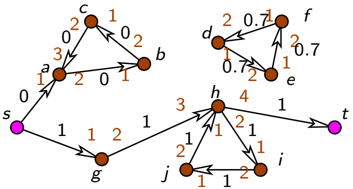
Def: A stable flow is **fully stable** if no cycle is unsaturated.

An example



What is a stable allocation here? (All capacities are 1.)
Directed cycle *abc* cannot carry any flow as otherwise *sa* would be a blocking path.
Directed cycle *def* can carry any flow between 0 and 1.
Directed cycle *hij* must carry unit flow as otherwise closed walk *hij* would be blocking.
Def: A stable flow is **fully stable** if no cycle is unsaturated.
A fully stable flow might not exist.

An example



What is a stable allocation here? (All capacities are 1.)

Directed cycle *abc* cannot carry any flow as otherwise *sa* would be a blocking path.

Directed cycle *def* can carry any flow between 0 and 1.

Directed cycle *hij* must carry unit flow as otherwise closed walk *hij* would be blocking.

Def: A stable flow is **fully stable** if no cycle is unsaturated.

A fully stable flow might not exist.

Theorem: Deciding the existence of a fully stable flow is NP-complete.

Stable flows and stable allocations

Possible proof: extension of the Gale-Shapley algorithm.

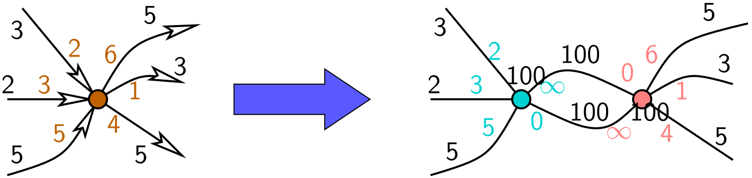
Stable flows and stable allocations

Possible proof: extension of the Gale-Shapley algorithm. But...

Stable flows and stable allocations

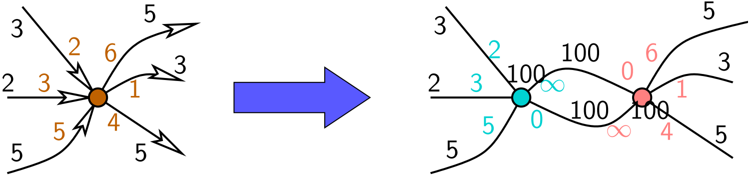
Possible proof: extension of the Gale-Shapley algorithm. But...
We deduce the thm from its special case: the Baïou-Balinski result.

Stable flows and stable allocations



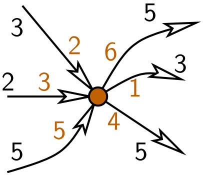
Possible proof: extension of the Gale-Shapley algorithm. But...
We deduce the thm from its special case: the Baïou-Balinski result.
Proof: Split each nonterminal vertex into a receiver and a transmitter with “high” capacity and introduce new edges with “high” capacities and “first-last” ranks.

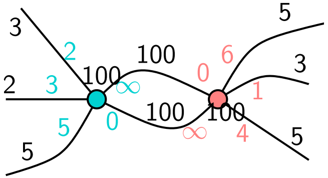
Stable flows and stable allocations



Possible proof: extension of the Gale-Shapley algorithm. But...
We deduce the thm from its special case: the Baïou-Balinski result.
Proof: Split each nonterminal vertex into a receiver and a transmitter with "high" capacity and introduce new edges with "high" capacities and "first-last" ranks. We get a bipartite graph with edge and vertex capacities and inherited preferences.

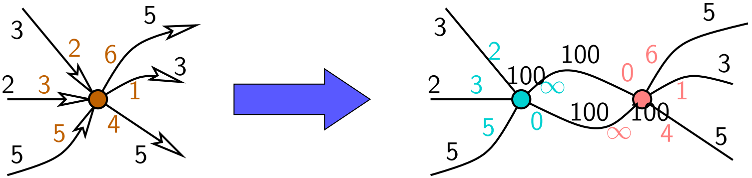
Stable flows and stable allocations





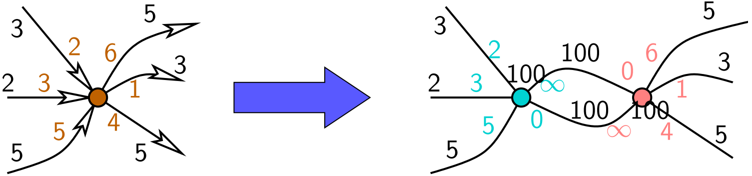
Possible proof: extension of the Gale-Shapley algorithm. But...
We deduce the thm from its special case: the Baïou-Balinski result.
Proof: Split each nonterminal vertex into a receiver and a transmitter with "high" capacity and introduce new edges with "high" capacities and "first-last" ranks. We get a bipartite graph with edge and vertex capacities and inherited preferences. So there is a stable allocation.

Stable flows and stable allocations



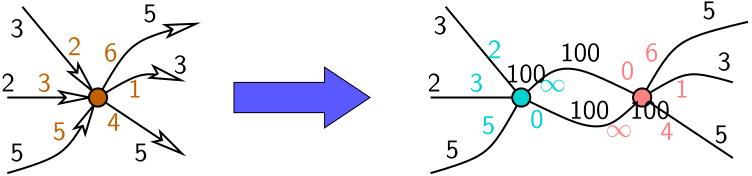
Possible proof: extension of the Gale-Shapley algorithm. But...
We deduce the thm from its special case: the Baïou-Balinski result.
Proof: Split each nonterminal vertex into a receiver and a transmitter with "high" capacity and introduce new edges with "high" capacities and "first-last" ranks. We get a bipartite graph with edge and vertex capacities and inherited preferences. So there is a stable allocation. The "restriction" of any stable allocation is a stable flow

Stable flows and stable allocations



Possible proof: extension of the Gale-Shapley algorithm. But...
We deduce the thm from its special case: the Baïou-Balinski result.
Proof: Split each nonterminal vertex into a receiver and a transmitter with "high" capacity and introduce new edges with "high" capacities and "first-last" ranks. We get a bipartite graph with edge and vertex capacities and inherited preferences. So there is a stable allocation. The "restriction" of any stable allocation is a stable flow, and each stable flow can be extended to a "canonical" stable allocation. □

Stable flows and stable allocations

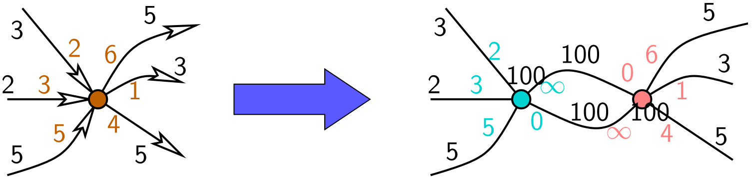


Possible proof: extension of the Gale-Shapley algorithm. But...
We deduce the thm from its special case: the Baïou-Balinski result.

Proof: Split each nonterminal vertex into a receiver and a transmitter with "high" capacity and introduce new edges with "high" capacities and "first-last" ranks. We get a bipartite graph with edge and vertex capacities and inherited preferences. So there is a stable allocation. The "restriction" of any stable allocation is a stable flow, and each stable flow can be extended to a "canonical" stable allocation. □

Facts: (1) Any two stable flows have the same value.

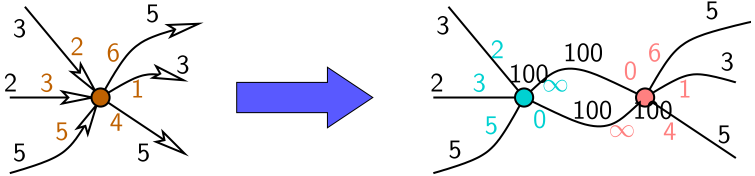
Stable flows and stable allocations



Possible proof: extension of the Gale-Shapley algorithm. But...
We deduce the thm from its special case: the Baïou-Balinski result.
Proof: Split each nonterminal vertex into a receiver and a transmitter with “high” capacity and introduce new edges with “high” capacities and “first-last” ranks. We get a bipartite graph with edge and vertex capacities and inherited preferences. So there is a stable allocation. The “restriction” of any stable allocation is a stable flow, and each stable flow can be extended to a “canonical” stable allocation. □

Facts: (1) Any two stable flows have the same value.
(2) Each arc incident with s or t has the same flow in a stable flow.

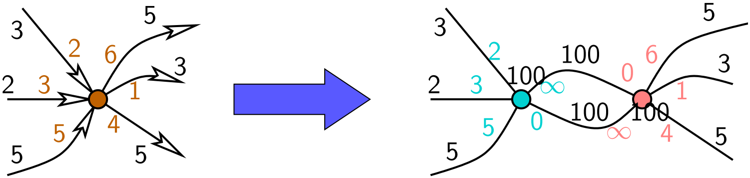
Stable flows and stable allocations



Possible proof: extension of the Gale-Shapley algorithm. But...
We deduce the thm from its special case: the Baïou-Balinski result.
Proof: Split each nonterminal vertex into a receiver and a transmitter with "high" capacity and introduce new edges with "high" capacities and "first-last" ranks. We get a bipartite graph with edge and vertex capacities and inherited preferences. So there is a stable allocation. The "restriction" of any stable allocation is a stable flow, and each stable flow can be extended to a "canonical" stable allocation. □

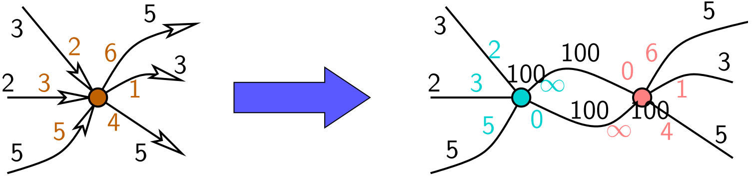
Facts: (1) Any two stable flows have the same value.
(2) Each arc incident with s or t has the same flow in a stable flow.
(3) The lattice structure of stable allocations can be generalized.

Lattice structure of stable flows



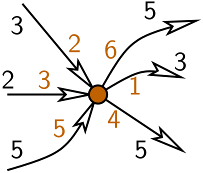
If f is a stable flow, then each nonterminal vertex is either a "customer" or a "vendor" determined by the canonical stable allocation of f.

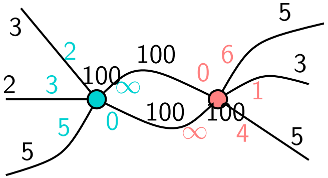
Lattice structure of stable flows



If f is a stable flow, then each nonterminal vertex is either a "customer" or a "vendor" determined by the canonical stable allocation of f. Nonterminals have preferences on stable flows.

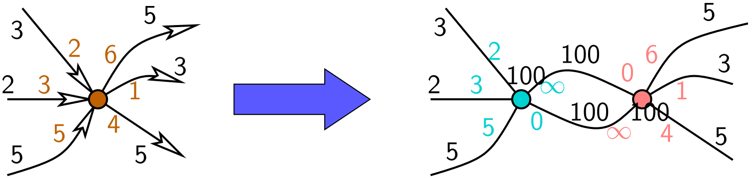
Lattice structure of stable flows





If f is a stable flow, then each nonterminal vertex is either a "customer" or a "vendor" determined by the canonical stable allocation of f. Nonterminals have preferences on stable flows. A customer position is better than a vendor position.

Lattice structure of stable flows

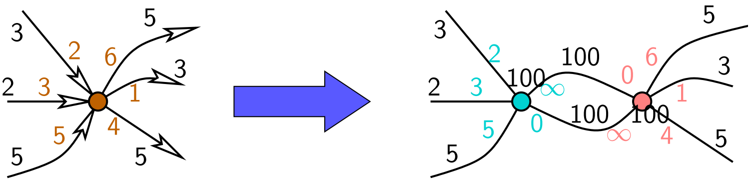


If f is a stable flow, then each nonterminal vertex is either a "customer" or a "vendor" determined by the canonical stable allocation of f. Nonterminals have preferences on stable flows.

A customer position is better than a vendor position.

A vendor prefers to transmit more flow.

Lattice structure of stable flows



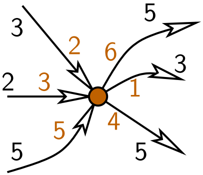
If f is a stable flow, then each nonterminal vertex is either a "customer" or a "vendor" determined by the canonical stable allocation of f. Nonterminals have preferences on stable flows.

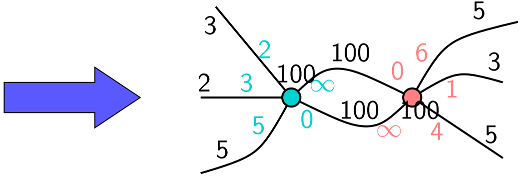
A customer position is better than a vendor position.

A vendor prefers to transmit more flow.

A customer prefers to transmit less flow.

Lattice structure of stable flows





If f is a stable flow, then each nonterminal vertex is either a "customer" or a "vendor" determined by the canonical stable allocation of f. Nonterminals have preferences on stable flows.

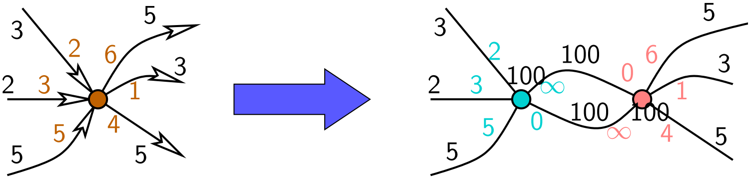
A customer position is better than a vendor position.

A vendor prefers to transmit more flow.

A customer prefers to transmit less flow.

Otherwise the better selling (worst buying) position is preferred.

Lattice structure of stable flows



If f is a stable flow, then each nonterminal vertex is either a "customer" or a "vendor" determined by the canonical stable allocation of f. Nonterminals have preferences on stable flows.
A customer position is better than a vendor position.
A vendor prefers to transmit more flow.
A customer prefers to transmit less flow.
Otherwise the better selling (worst buying) position is preferred.
Lattice property of stable flows: If two stable flows are given and each nonterminal picks the better (worse) position from the two flows then another stable flow is constructed.

Conclusion

Closely related: Ostrovsky has an earlier result on supply chains.
On one hand, he assumed that the network is **acyclic**.
On the other hand, he could considerably relax the Kirchhoff rule to so called same side substitutability and cross side complementarity. His requirement is that each "agent" transmits goods in a certain monotone manner: buying more means selling more and vice versa.

Conclusion

Closely related: Ostrovsky has an earlier result on supply chains.
On one hand, he assumed that the network is **acyclic**.
On the other hand, he could considerably relax the Kirchhoff rule to so called same side substitutability and cross side complementarity. His requirement is that each "agent" transmits goods in a certain monotone manner: buying more means selling more and vice versa.
Natural question: Common generalization?

Conclusion

Closely related: Ostrovsky has an earlier result on supply chains. On one hand, he assumed that the network is **acyclic**. On the other hand, he could considerably relax the Kirchhoff rule to so called same side substitutability and cross side complementarity. His requirement is that each "agent" transmits goods in a certain monotone manner: buying more means selling more and vice versa.

Natural question: Common generalization?

Ongoing work with Akihisa Tamura and Zsuzsi Jankó.

Conclusion

Closely related: Ostrovsky has an earlier result on supply chains. On one hand, he assumed that the network is **acyclic**.
On the other hand, he could considerably relax the Kirchhoff rule to so called same side substitutability and cross side complementarity. His requirement is that each "agent" transmits goods in a certain monotone manner: buying more means selling more and vice versa.
Natural question: Common generalization?

Ongoing work with Akihisa Tamura and Zsuzsi Jankó.

Thank you