Graph Searching Games and Routing Reconfiguration in WDM Networks

Nicolas Nisse

MASCOTTE, INRIA, I3S, CNRS, UNS, Sophia Antipolis, France

Séminaire LIFL, March 5th, 2009

Plan

1. Introduction, Motivations, Many Variants
   - General Problem
   - Many Variants
2. Overview of Invisible Graph Searching
   - Definitions
   - Link with Graph decompositions
   - Distributed Graph Searching
3. Model for Routing Reconfiguration in Networks
   - Problem, Definitions and Previous Results
   - Our Contribution

Cops & robber/pursuit-evasion/graph searching

Capture an intruder in a network

a team of cops/searchers *vs.* a robber/intruder/fugitive

Combinatorial Problem:
search number, $s(G)$: minimum number of searchers to capture the fugitive whatever he does in a network G.

Algorithmic Problem:
strategy (sequence of moves) of the searchers allowing them to capture the fugitive

Motivations

- Graph Decompositions and Complexity Theory

tree(path)-decomposition = representation of a graph as a tree(path)

- structural theory of graphs
  (Graph Minor theory, Robertson & Seymour)
- many NP-complete problems tractable when bounded treewidth
  (Bodlaender, Courcelle)

Graph Searching = Algorithmic interpretation of tree-decompositions

different point of view leading to new results/algorithms

Motivations

- Graph Decompositions and Complexity Theory
- Distributed Algorithms for Clearing Networks

Graph Searching ⇔ to clear a contaminated network

robots with local view must clear unknown networks

tradeoffs between

- clearing-time
- number of robots
- knowledge about the network
- memory of robots

Motivations

- Graph Decompositions and Complexity Theory
- Distributed Algorithms for Clearing Networks
- Algorithmic Aspects of Routing Reconfiguration

second part of this talk

Motivations

- Graph Decompositions and Complexity Theory
- Distributed Algorithms for Clearing Networks
- Algorithmic Aspects of Routing Reconfiguration
- and also:

structural characterizations of graphs, localization of mobile targets, study of tradeoffs between space/time complexity, link with network exploration, etc.

Taxonomy of graph searching games

Capture = Surround and/or Occupy the same node as the fugitive

	robber's characteristics			
	bounded speed		arbitrary fast	
	visible	invisible	visible	invisible
turn by turn **Occupy**	Cops & Robber	X	X	?
simultaneous **Occ. & Sur.**	?	X	graph searching treewidth	graph searching pathwidth

? = No studies (as far as I know)
X = Very few studies

Taxonomy of graph searching games

Capture = Surround and/or Occupy the same node as the fugitive

	robber's characteristics			
	bounded speed		arbitrary fast	
	visible	invisible	visible	invisible
turn by turn **Occupy**	Cops & Robber	X	X	?
simultaneous **Occ. & Sur.**	?	X	graph searching treewidth	pathwidth

Capture = Surround the fugitive

simultaneous **Surround**	?	?	?	routing reconfig.

Plan

Graph Searching (Breish 67, Parson 78)

network modelized by a graph G

Graph Searching (Breish 67, Parson 78)

network modelized by a graph G

searchers and fugitive **on nodes**, move **along edges** and play **simultaneously**

Graph Searching (Breish 67, Parson 78)

network modelized by a graph G

searchers and fugitive **on nodes**, move **along edges** and play **simultaneously**

the fugitive is

- **invisible**
- **arbitrary fast** (as far as he does not meet a searcher)
- **omniscient** (best possible strategy to avoid searchers)

Graph Searching (Breish 67, Parson 78)

network modelized by a graph G

searchers and fugitive **on nodes**, move **along edges** and play **simultaneously**

the fugitive is

- **invisible**
- **arbitrary fast** (as far as he does not meet a searcher)
- **omniscient** (best possible strategy to avoid searchers)

capture: the fugitive occupies the same node as a searcher and cannot flee

Graph Searching (Breish 67, Parson 78)

network modelized by a graph G

searchers and fugitive **on nodes**, move **along edges** and play **simultaneously**

the fugitive is

- **invisible**
- **arbitrary fast** (as far as he does not meet a searcher)
- **omniscient** (best possible strategy to avoid searchers)

capture: the fugitive occupies the same node as a searcher and cannot flee

equivalent to clear a contaminated network (toxic gas, virus)

Search Strategy (Parson 78) Variant of Kirousis & Papadimitriou [TCS 86]

Sequence of two basic operations,. . .

1. Place a searcher at a node;
2. Remove a searcher from a node.

. . . that must result in catching the fugitive

capture: to surround the fugitive **and** to occupy his node
- node cleared when occupied by a searcher
- node contaminated if the fugitive can access it

Minimize the number of searchers.

Let $s(G)$ be the smallest number of searchers needed to capture the fugitive in G *whatever he does* (worst case).

Search Strategy (Parson 78)
Variant of Kirousis & Papadimitriou [TCS 86]

Sequence of two basic operations,. . .

1. Place a searcher at a node;
2. Remove a searcher from a node.

. . . that must result in catching the fugitive

capture: to surround the fugitive **and** to occupy his node
- node cleared when occupied by a searcher
- node contaminated if the fugitive can access it

Minimize the number of searchers.

Let $s(G)$ be the smallest number of searchers needed to capture the fugitive in G *whatever he does* (worst case).

Search Strategy (Parson 78)
Variant of Kirousis & Papadimitriou [TCS 86]

Sequence of two basic operations,...

1. Place a searcher at a node;
2. Remove a searcher from a node.

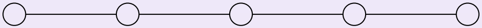
...that must result in catching the fugitive

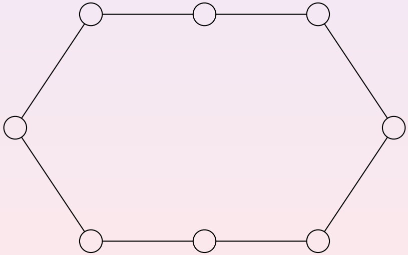
capture: to surround the fugitive **and** to occupy his node
- node cleared when occupied by a searcher
- node contaminated if the fugitive can access it

Minimize the number of searchers.

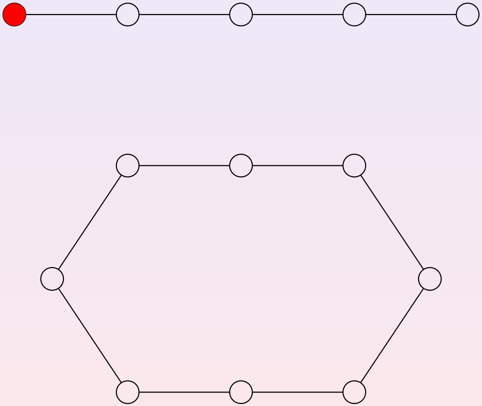
Let $s(G)$ be the smallest number of searchers needed to capture the fugitive in G *whatever he does* (worst case).

Basic examples: Path and Ring

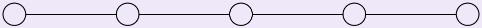


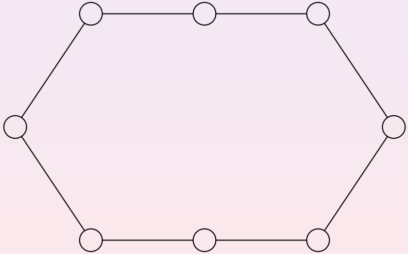


Basic examples: Path and Ring

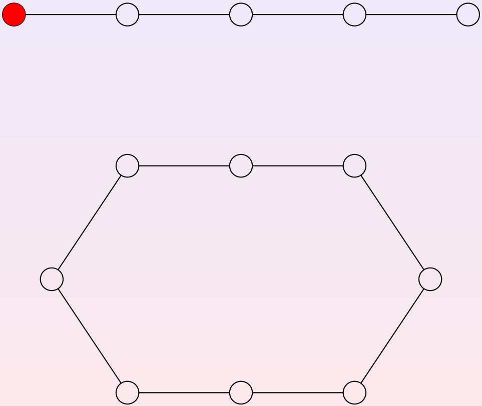


Basic examples: Path and Ring

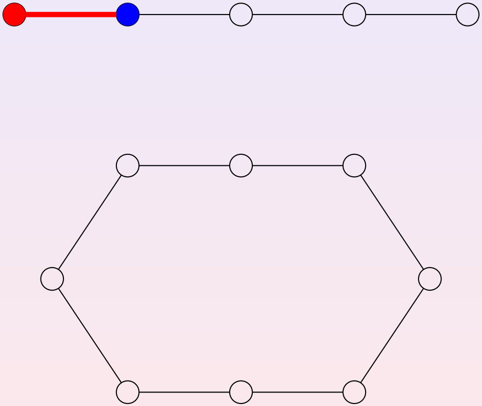




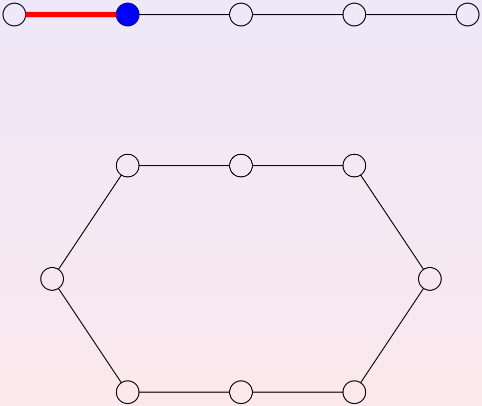
Basic examples: Path and Ring



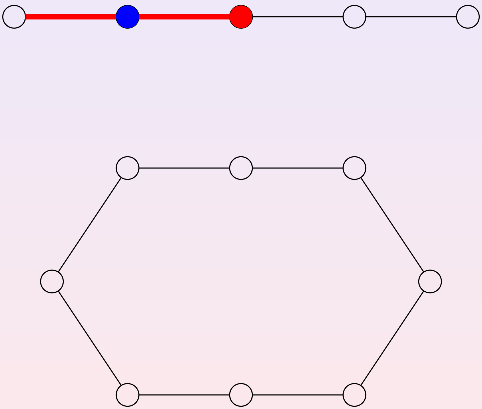
Basic examples: Path and Ring



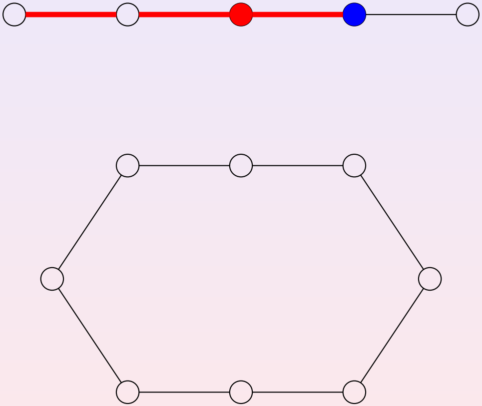
Basic examples: Path and Ring



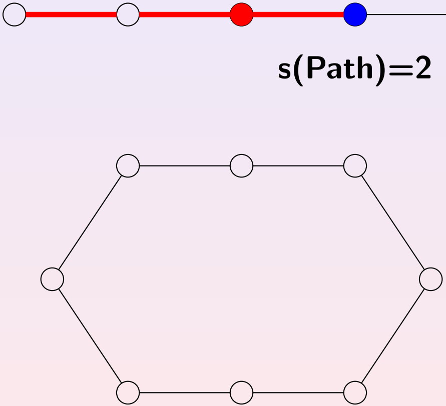
Basic examples: Path and Ring



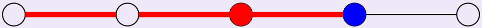
Basic examples: Path and Ring



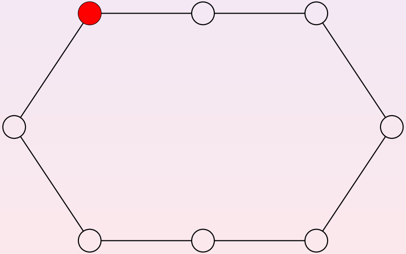
Basic examples: Path and Ring



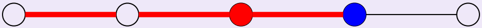
Basic examples: Path and Ring



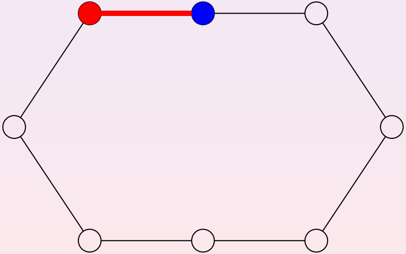
s(Path)=2



Basic examples: Path and Ring

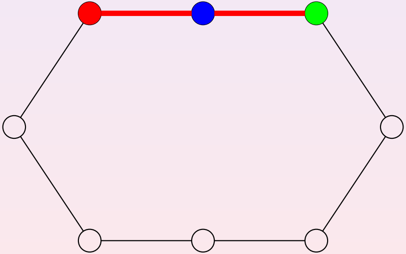


s(Path)=2



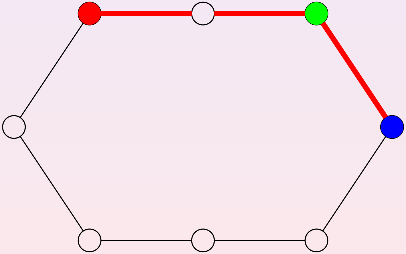
Basic examples: Path and Ring

s(Path)=2

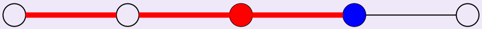


Basic examples: Path and Ring

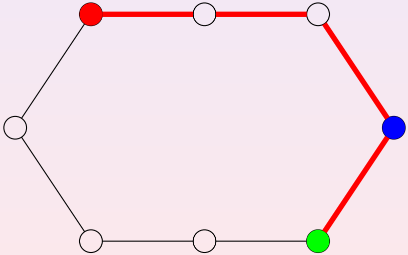
s(Path)=2



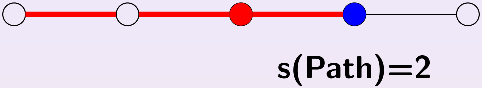
Basic examples: Path and Ring

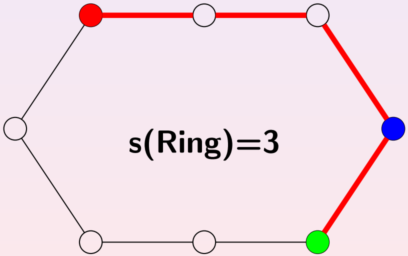


s(Path)=2



Basic examples: Path and Ring





NP-hardness

The following problem is NP-hard

Input: a graph G, an integer $k > 0$, Megiddo *et al.*,
Output: $s(G) \leq k$? [JACM 88]

Remark: linear in the class of trees, Skodinis [JAlg 03]

Monotonicity and NP-completeness

A vertex v is recontaminated if the fugitive can move to v after v has been occupied by a searcher.

Monotonicity

A search strategy is monotone if no recontamination ever occurs. That is, a vertex is occupied by a searcher only once.

Recontamination does not help

There always exists an optimal monotone search strategy.
LaPaugh [JACM 93], Bienstock & Seymour [JAlg 91]

Corollary: The above problems belong to NP.

Monotonicity and NP-completeness

A vertex v is recontaminated if the fugitive can move to v after v has been occupied by a searcher.

Monotonicity

A search strategy is monotone if no recontamination ever occurs. That is, a vertex is occupied by a searcher only once.

Recontamination does not help

There always exists an optimal monotone search strategy.
LaPaugh [JACM 93], Bienstock & Seymour [JAlg 91]

Corollary: The above problems belong to NP.

Graph Searching and Graph Decompositions

Thanks to the monotonicity, we get:

Search number and Pathwidth (pw)

For any graph G, $s(G) = pw(G) + 1$,

Kinnersley [IPL 92], Ellis, Sudborough, and Turner [Inf.Comp.94]

Why Graph Searching interpretation is (so?) important?

- led to results for directed graphs decompositions
  Baràt [Graphs Comb. 06], Hunter & Kreutzer [TCS 08], Hunter *et al.* [STACS 06], Adler [JCTB 07]
- led to results for unified and generalized decompositions
  Mazoit, N. [TCS 08], Amini, Mazoit, N., Thomassé [SIAM Disc. Maths], Fomin, Fraigniaud, N. [Algorithmica], Berthomé, N. (submitted)

Graph Searching and Graph Decompositions

Thanks to the monotonicity, we get:

Search number and Pathwidth (pw)

For any graph G, $s(G) = pw(G) + 1$,

Kinnersley [IPL 92], Ellis, Sudborough, and Turner [Inf.Comp.94]

Why Graph Searching interpretation is (so?) important?

- led to results for directed graphs decompositions
  Baràt [Graphs Comb. 06], Hunter & Kreutzer [TCS 08], Hunter *et al.* [STACS 06], Adler [JCTB 07]
- led to results for unified and generalized decompositions
  Mazoit, N. [TCS 08], Amini, Mazoit, N., Thomassé [SIAM Disc. Maths], Fomin, Fraigniaud, N. [Algorithmica], Berthomé, N. (submitted)

Distributed Setting, Connected Graph Searching

Goal in Distributed Settings

A team of robots must clear a contaminated network
Robots must compute online their strategy,
with a local view of an unknown network

New constraint: Connectivity **(Barrière *et al.* [SPAA 02])**
Need of safe communications impose connected clear part

Cost of connectivity (in terms of number of searchers)

$\leq 2s(T)$ searchers may be used in any tree T **(Barrière *et al.* [WG 03])**
$\leq \log(n) \cdot s(G)$ searchers may be used in any G **(Fraigniaud, N. [LATIN 06])**
Recontamination helps **(Alspach, Dyer, Yang [ISAAC 04])**

strictly more searchers may be used in this setting

$mcs(G)$ smallest # searchers to clear G in a monotone connected way

Distributed Setting, Connected Graph Searching

Goal in Distributed Settings

A team of robots must clear a contaminated network
Robots must compute online their strategy,
with a local view of an unknown network

New constraint: Connectivity **(Barrière** ***et al.*** **[SPAA 02])**
Need of safe communications impose connected clear part

Cost of connectivity (in terms of number of searchers)

$\leq 2s(T)$ searchers may be used in any tree T **(Barrière** ***et al.*** **[WG 03])**
$\leq \log(n) \cdot s(G)$ searchers may be used in any G **(Fraigniaud, N. [LATIN 06])**
Recontamination helps **(Alspach, Dyer, Yang [ISAAC 04])**

strictly more searchers may be used in this setting
$mcs(G)$ smallest # searchers to clear G in a monotone connected way

Distributed Setting, Connected Graph Searching

Goal in Distributed Settings

A team of robots must clear a contaminated network
Robots must compute online their strategy,
with a local view of an unknown network

New constraint: Connectivity **(Barrière** *et al.* **[SPAA 02])**
Need of safe communications impose connected clear part

Cost of connectivity (in terms of number of searchers)

$\leq 2s(T)$ searchers may be used in any tree T **(Barrière** *et al.* **[WG 03])**
$\leq \log(n) \cdot s(G)$ searchers may be used in any G **(Fraigniaud, N. [LATIN 06])**
Recontamination helps **(Alspach, Dyer, Yang [ISAAC 04])**

strictly more searchers may be used in this setting
$mcs(G)$ smallest # searchers to clear G in a monotone connected way

Distributed Setting, Connected Graph Searching

Goal in Distributed Settings

A team of robots must clear a contaminated network
Robots must compute online their strategy,
with a local view of an unknown network

New constraint: Connectivity **(Barrière** *et al.* **[SPAA 02])**
Need of safe communications impose connected clear part

Cost of connectivity (in terms of number of searchers)

$\leq 2s(T)$ searchers may be used in any tree T **(Barrière** *et al.* **[WG 03])**
$\leq \log(n) \cdot s(G)$ searchers may be used in any G **(Fraigniaud, N. [LATIN 06])**
Recontamination helps **(Alspach, Dyer, Yang [ISAAC 04])**

strictly more searchers may be used in this setting

$mcs(G)$ smallest # searchers to clear G in a monotone connected way

Distributed Algorithms to Clear Networks (1)

The searchers have a prior knowledge of the topology.

Protocols to clear specific topologies

- Tree. **(Barrière *et al.* [SPAA 02])**
- Mesh. **(Flocchini, Luccio, and Song [CIC 05])**
- Hypercube. **(Flocchini, Huang, and Luccio [IPDPS 05])**
- Tori. **(Flocchini, Luccio, and Song [IPDPS 06])**

Optimal number (*mcs*) of searchers are used
Searchers have $\log n$ bits of memory.
Local node memory of $\log n$ bits.

Distributed Algorithms to Clear Networks (2)

What about distributed algorithms for clearing any graphs?

Clearing any network G using $mcs(G)$ searchers

Blin, Fraigniaud, N., Vial [TCS 08]

But, strategy non monotone: **may take exponential time**

Tradeoff knowledge/ number of searchers/ clearing time

monotone (Polynomial-time) clearing using $mcs(G)$ searchers

requires $\Theta(n \log n)$ bits of information (N., Soguet [TCS])

monotone strategy in any unknown graph G

requires $mcs(G) \cdot \Theta(n/\log n)$ searchers (Ilcinkas, N., Soguet [OPODIS 07])

Distributed Algorithms to Clear Networks (2)

What about distributed algorithms for clearing any graphs?

Clearing any network G using $mcs(G)$ searchers

Blin, Fraigniaud, N., Vial [TCS 08]

But, strategy non monotone: **may take exponential time**

Tradeoff knowledge/ number of searchers/ clearing time

monotone (Polynomial-time) clearing using $mcs(G)$ searchers

requires $\Theta(n \log n)$ bits of information (N., Soguet [TCS])

monotone strategy in any unknown graph G

requires $mcs(G) \cdot \Theta(n/\log n)$ searchers (Ilcinkas, N., Soguet [OPODIS 07])

Distributed Algorithms to Clear Networks (2)

What about distributed algorithms for clearing any graphs?

Clearing any network G using $mcs(G)$ searchers

Blin, Fraigniaud, N., Vial [TCS 08]

But, strategy non monotone: **may take exponential time**

Tradeoff knowledge/ number of searchers/ clearing time

monotone (Polynomial-time) clearing using $mcs(G)$ searchers

requires $\Theta(n \log n)$ **bits of information (N., Soguet [TCS])**

monotone strategy in any unknown graph G

requires $mcs(G) \cdot \Theta(n/\log n)$ **searchers (Ilcinkas, N., Soguet [OPODIS 07])**

Plan

Short Historic

Comes from a problem in WDM Networks **(Jose & Somani, 03)**
New model to handle it **(Coudert & Sereni, 05)**[1]

It looks like graph searching ????

Yes, it is !!!

Leads to new results and algorithms...

[1] Thank you to David Coudert for some of the following slides

Routing in WDM Networks

Physical Network = **multi-graph** $G = (V, A)$

Links provide several wavelengths = **multi arcs**

Routing of a set of requests/connections

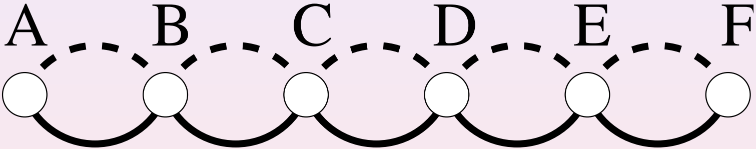
set of requests $\mathcal{R} \subseteq 2^{V \times V}$
routing: for each request (u, v),
a path from u to v and 1 wavelength.

Problem: due to dynamicity of traffic, failures,

how to maintain an efficient routing?

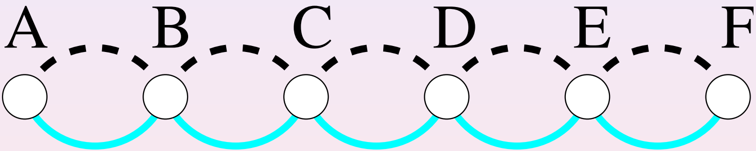
Basic Example

Network = Path with two wavelengths per link
(2 parallel edges).



Basic Example

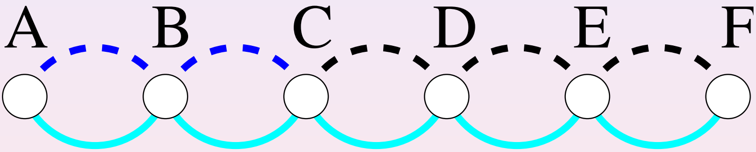
Network = Path with two wavelengths per link
(2 parallel edges).



request for a A-F connection

Basic Example

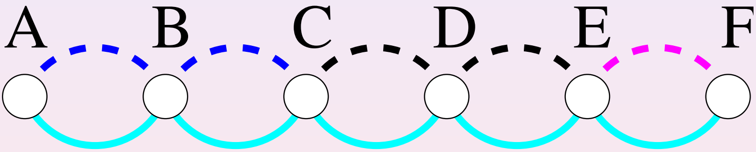
Network = Path with two wavelengths per link
(2 parallel edges).



request for a A-C connection

Basic Example

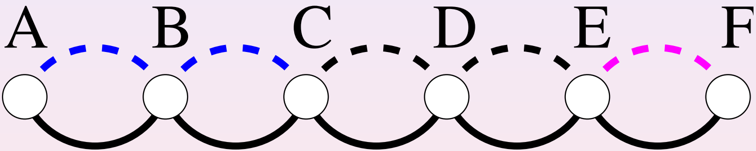
Network = Path with two wavelengths per link
(2 parallel edges).



request for a E-F connection

Basic Example

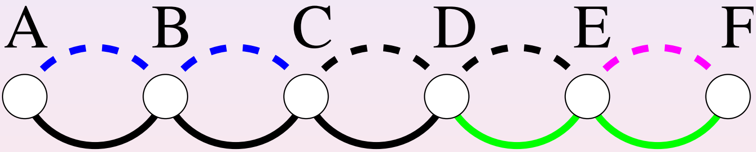
Network = Path with two wavelengths per link
(2 parallel edges).



end of the A-F connection

Basic Example

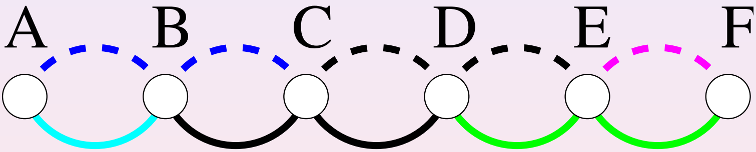
Network = Path with two wavelengths per link
(2 parallel edges).



request for a D-F connection

Basic Example

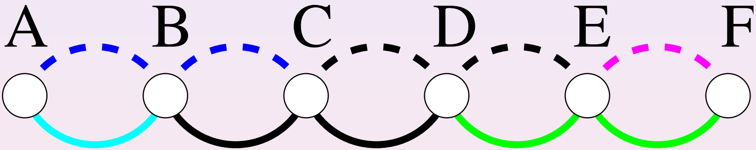
Network = Path with two wavelengths per link
(2 parallel edges).



request for a A-B connection

Basic Example

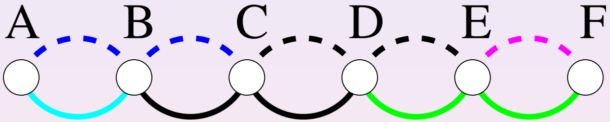
Network = Path with two wavelengths per link
(2 parallel edges).

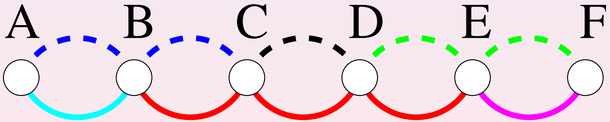


What if there is a request for a B-E connection?
(using ONE wavelength) Impossible with the current routing...

Basic Example

Network = Path with two wavelengths per link
(2 parallel edges).





... While it is possible !!

What can we do ?

- Reject the new request → *blocking probabilities*
- Stop all requests and restart with new “optimal” routing
- Sequence of switching to converge to new routing
- Find the most suitable route for incoming request with eventual rerouting of pre-established connections

Our problem:

Inputs: Set of connection requests
+ current **and** new routing

Output: Scheduling for switching connection requests from current to new routes

Constraint: A connection is switched only once

What can we do ?

- Reject the new request → *blocking probabilities*
- Stop all requests and restart with new "optimal" routing
- Sequence of switching to converge to new routing
- Find the most suitable route for incoming request with eventual rerouting of pre-established connections

Our problem:

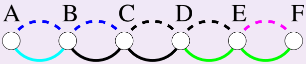
Inputs: Set of connection requests
+ current **and** new routing

Output: Scheduling for switching connection requests from current to new routes

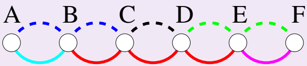
Constraint: A connection is switched only once

Tool: the Dependency Digraph (Jose & Somani, 03)

initial routing $\mathcal{I}$ + request BE



final routing $\mathcal{F}$ (pre-computed)

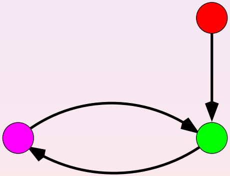


Dependancy Digraph

- one vertex per connection with different routes in I and F
- arc from u to v if ressources needed by u in $\mathcal{F}$ are used by v in $\mathcal{I}$

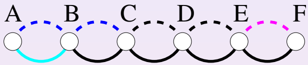
If cycles exist $\Rightarrow$ cyclic dependencies $\Rightarrow$

some requests must be interrupted

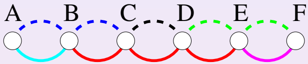


Tool: the Dependency Digraph (Jose & Somani, 03)

initial routing $\mathcal{I}$ + request BE



final routing $\mathcal{F}$ (pre-computed)

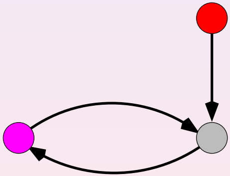


Dependancy Digraph

- one vertex per connection with different routes in I and F
- arc from u to v if ressources needed by u in $\mathcal{F}$ are used by v in $\mathcal{I}$

If cycles exist $\Rightarrow$ cyclic dependencies $\Rightarrow$

some requests must be interrupted

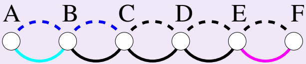


interrupt DF-connection

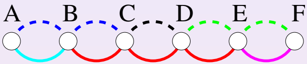
(Break-before-Make)

Tool: the Dependency Digraph (Jose & Somani, 03)

initial routing $\mathcal{I}$ + request BE



final routing $\mathcal{F}$ (pre-computed)

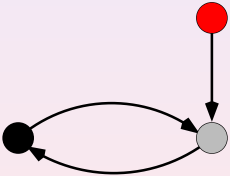


Dependancy Digraph

- one vertex per connection with different routes in I and F
- arc from u to v if ressources needed by u in $\mathcal{F}$ are used by v in $\mathcal{I}$

If cycles exist $\Rightarrow$ cyclic dependencies $\Rightarrow$

some requests must be interrupted

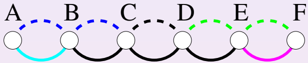


reroute EF-connection

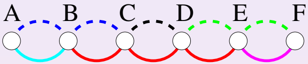
(Make-before-Break)

Tool: the Dependency Digraph (Jose & Somani, 03)

initial routing $\mathcal{I}$ + request BE



final routing $\mathcal{F}$ (pre-computed)

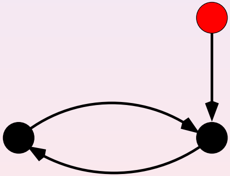


Dependancy Digraph

- one vertex per connection with different routes in I and F
- arc from u to v if ressources needed by u in $\mathcal{F}$ are used by v in $\mathcal{I}$

If cycles exist $\Rightarrow$ cyclic dependencies $\Rightarrow$

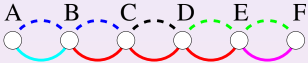
some requests must be interrupted



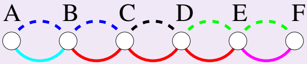
reroute DF-connection

Tool: the Dependency Digraph (Jose & Somani, 03)

initial routing $\mathcal{I}$ + request BE



final routing $\mathcal{F}$ (pre-computed)

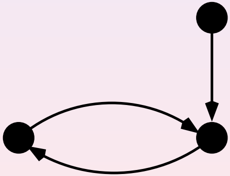


Dependancy Digraph

- one vertex per connection with different routes in I and F
- arc from u to v if ressources needed by u in $\mathcal{F}$ are used by v in $\mathcal{I}$

If cycles exist $\Rightarrow$ cyclic dependencies $\Rightarrow$

some requests must be interrupted

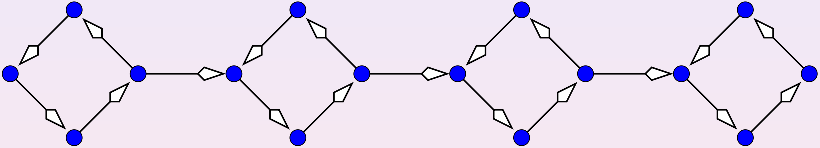


route BE-connection

Possible objectives

Minimize overall number of break-before-make

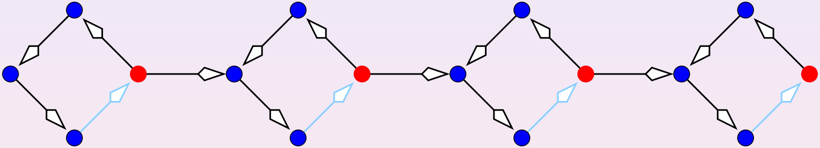
Minimum Feedback Vertex Set (MFVS), here $N/4$



Possible objectives

Minimize overall number of break-before-make

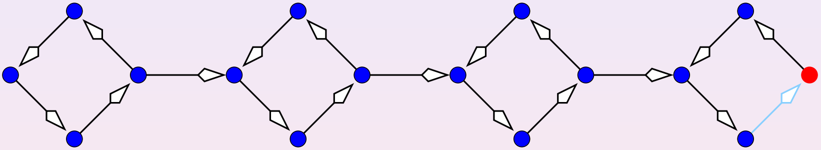
Minimum Feedback Vertex Set (MFVS), here $N/4$



Possible objectives

Minimize overall number of break-before-make

Minimum Feedback Vertex Set (MFVS), here $N/4$



Minimize number of simultaneous break-before-make

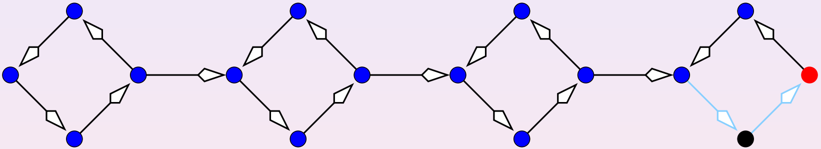
Process Number, pn = smallest number of requests that have to be **simultaneously** interrupted.
Here, $pn = 1 \Rightarrow$ Gap with MFVS up to $N/2$

It is a graph searching formulation of the problem...

Possible objectives

Minimize overall number of break-before-make

Minimum Feedback Vertex Set (MFVS), here $N/4$



Minimize number of simultaneous break-before-make

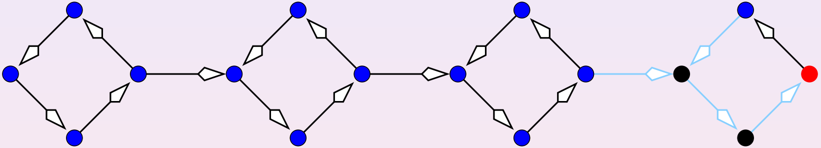
Process Number, pn = smallest number of requests that have to be **simultaneously** interrupted.
Here, $pn = 1 \Rightarrow$ Gap with MFVS up to $N/2$

It is a graph searching formulation of the problem...

Possible objectives

Minimize overall number of break-before-make

Minimum Feedback Vertex Set (MFVS), here $N/4$



Minimize number of simultaneous break-before-make

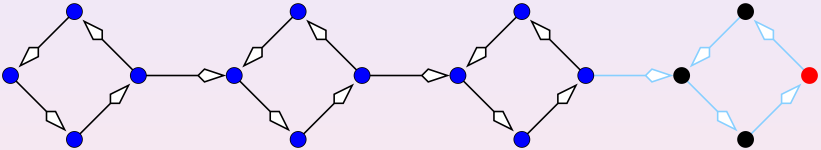
Process Number, pn = smallest number of requests that have to be **simultaneously** interrupted.
Here, $pn = 1 \Rightarrow$ Gap with MFVS up to $N/2$

It is a graph searching formulation of the problem...

Possible objectives

Minimize overall number of break-before-make

Minimum Feedback Vertex Set (MFVS), here $N/4$



Minimize number of simultaneous break-before-make

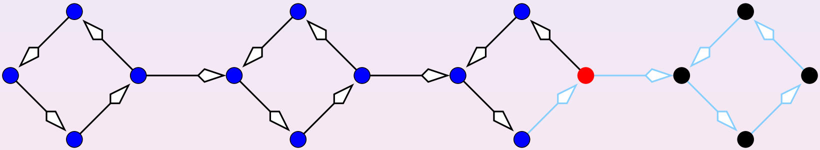
Process Number, pn = smallest number of requests that have to be **simultaneously** interrupted.
Here, $pn = 1 \Rightarrow$ Gap with MFVS up to $N/2$

It is a graph searching formulation of the problem...

Possible objectives

Minimize overall number of break-before-make

Minimum Feedback Vertex Set (MFVS), here $N/4$



Minimize number of simultaneous break-before-make

Process Number, pn = smallest number of requests that have to be **simultaneously** interrupted.
Here, $pn = 1 \Rightarrow$ Gap with MFVS up to $N/2$

It is a graph searching formulation of the problem...

Routing Reconfiguration, Process number

Given a digraph D (Dependency digraph)

Sequence of three basic operations,...

1. Place a searcher at a node = **interrupt the request**;
2. Process a node if all its out-neighbors are either processed or occupied by an agent = **(Re)route a connection when final resources are available**;
3. Remove an agent from a node, after having processed it.

...that must result in processing all nodes

Process number $pn(G)$= min p | G can be processed with p agents

Remarks: Graph Searching game when capture = surround
It is monotone by definition

Routing Reconfiguration, Process number

Given a digraph D (Dependency digraph)

Sequence of three basic operations,. . .

1. Place a searcher at a node = **interrupt the request**;
2. Process a node if all its out-neighbors are either processed or occupied by an agent = **(Re)route a connection when final resources are available**;
3. Remove an agent from a node, after having processed it.

. . . that must result in processing all nodes

Process number $pn(G) = \min p \mid G$ can be processed with p agents

Remarks: Graph Searching game when capture = surround

It is monotone by definition

Example: DAG

Only one operation is used

1. Place a searcher at a node = interrupt the request;
2. Process a node if all its out-neighbors are either processed or occupied by an agent = **(Re)route a connection when final resources are available**;
3. Remove an agent from a node, after having processed it.

Direct path, DAG

Theorem

$pn(D) = 0$ iff D is a DAG

Example: DAG

Only one operation is used

1. Place a searcher at a node = interrupt the request;
2. Process a node if all its out-neighbors are either processed or occupied by an agent = **(Re)route a connection when final resources are available**;
3. Remove an agent from a node, after having processed it.

Direct path, DAG

Theorem

$pn(D) = 0$ iff D is a DAG

Example: DAG

Only one operation is used

1. Place a searcher at a node = interrupt the request;
2. Process a node if all its out-neighbors are either processed or occupied by an agent = **(Re)route a connection when final resources are available**;
3. Remove an agent from a node, after having processed it.

Direct path, DAG

Theorem

$pn(D) = 0$ iff D is a DAG

Example: DAG

Only one operation is used

1. Place a searcher at a node = interrupt the request;
2. Process a node if all its out-neighbors are either processed or occupied by an agent = **(Re)route a connection when final resources are available**;
3. Remove an agent from a node, after having processed it.

Direct path, DAG

Theorem

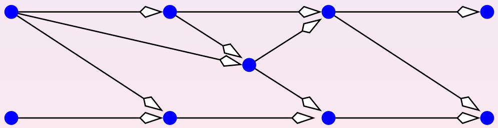
$pn(D) = 0$ iff D is a DAG

Example: DAG

Only one operation is used

1. Place a searcher at a node = interrupt the request;
2. Process a node if all its out-neighbors are either processed or occupied by an agent = **(Re)route a connection when final resources are available**;
3. Remove an agent from a node, after having processed it.

Direct path, DAG



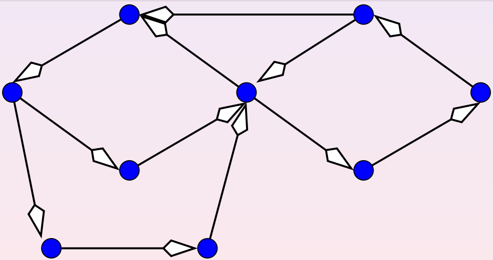
Theorem

$pn(D) = 0$ iff D is a DAG

Digraphs with process number 1

One agent is used

1. Place a searcher at a node = **interrupt the request**;
2. Process a node if all its out-neighbors are either processed or occupied by an agent = **(Re)route a connection when final resources are available**;
3. Remove an agent from a node, after having processed it.



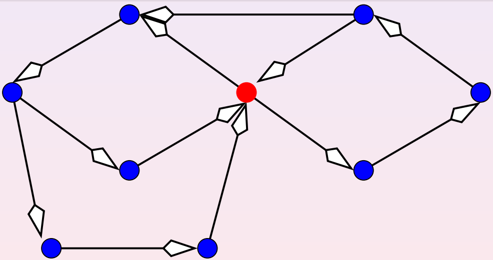
Theorem

$pn(D) = 1 \Leftrightarrow \forall SCC$, MFVS$(SCC) = 1$ $\qquad O(N + M)$

Digraphs with process number 1

One agent is used

1. Place a searcher at a node = **interrupt the request**;
2. Process a node if all its out-neighbors are either processed or occupied by an agent = **(Re)route a connection when final resources are available**;
3. Remove an agent from a node, after having processed it.



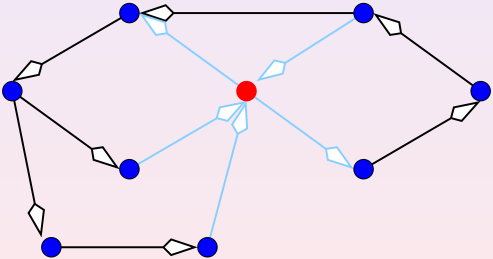
Theorem

$pn(D) = 1 \Leftrightarrow \forall SCC,\ \mathrm{MFVS}(SCC) = 1 \qquad O(N + M)$

Digraphs with process number 1

One agent is used

1. Place a searcher at a node = **interrupt the request**;
2. Process a node if all its out-neighbors are either processed or occupied by an agent = **(Re)route a connection when final resources are available**;
3. Remove an agent from a node, after having processed it.



Theorem

$pn(D) = 1 \Leftrightarrow \forall SCC,\ \text{MFVS}(SCC) = 1$ $\qquad O(N + M)$

Process number versus Search number

a parameter of (un)directed graphs

vs, vertex separation

Recall that the pathwidth $pw(G) = s(G) - 1$ (search number)

in undirected graph or symetric digraph

$pw(G) = vs(G)$ **(Kinnersley [IPL 92])**. Thus, $vs(G) = s(G) - 1$

Theorem **(Coudert & Sereni, 2007)**

$vs(D) \leq pn(D) \leq vs(D) + 1$

Complexity: NP-Hard, Not APX

- Characterization of digraphs with process number 0, 1, 2 **(Coudert & Sereni, 2007)**
- distributed $O(n \log n)$-time algorithm in trees **(Coudert, Huc, Mazauric [DISC 2008])**

What we delt with

Joint work with D. Coudert, F. Huc, D. Mazauric and J-S. Sereni [ONDM 09]

How to handle priority connections?

connections that cannot be interrupted

Heuristic and simulations

to compute upper bounds on process number

Joint work with D. Coudert and D. Mazauric [AGT 09]

What if requests can share links?

requests share a constant fraction of the bandwidth of a link

Two classes of services

Priority connections

- Refuse *by contract* (SLA) break-before-make
  $\Rightarrow$ vertex of the dependency digraph that cannot host agent

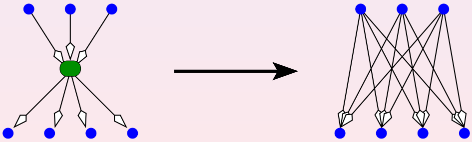
Impossibility

- Direct cycle of priority connections in the dependency digraph

$\Rightarrow$ *Small* number of such connections

- Recognition in time $O(N + M)$

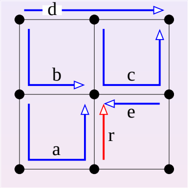
Transformation



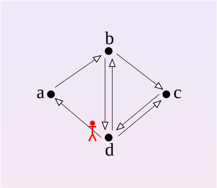
$\Rightarrow$ Same problem to solve

Example with priority connection d

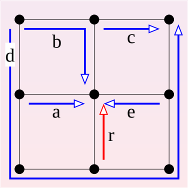
Symmetric grid, with 1 wavelength per arc.



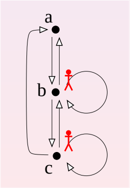
Routing 1



Dependency digraph, $pn = 1$



Routing 2



Without d, $pn = 2$

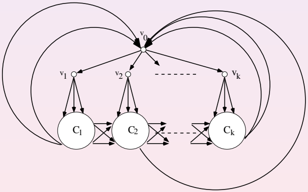
Cost of Priority Connections

D a digraph, and a set $P \subseteq V$ of priority connections

Generalize process number $pn(D,P)$= min p s.t. G can be processed with p agents without placing an agent on nodes of P

Lemma

For any digraph D and $P \subseteq V$
$pn(D,P)$ is defined iff P is a DAG, and in this case
$pn(D,P) \leq pn(D) + |N^+(P)|$ (assymptotically tight)



C_i symmetric clique of size k, $pn(D) = k+1$
$pn(D,P) = \sum_{i \leq k} |C_i|$ with $P = \{v_0, \cdots, v_k\}$.

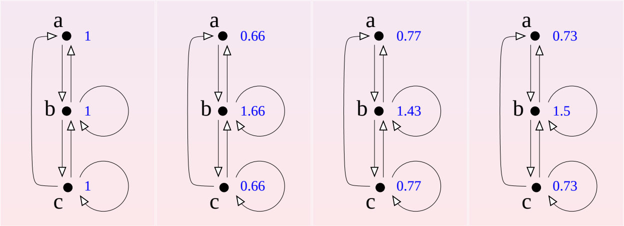
Previous heuristic (Jose and Somani [DRCN 03])

1. Compute all directed cycles using Johnson's algorithm
2. Choose the vertex that belongs to the maximum number of cycles
3. Remove that vertex and update set of cycles
4. Repeat 2-3 until remaining digraph is a DAG
5. Process DAG
6. Process removed vertices

- Heuristic for MFVS
- Complexity in $O((n+m)(c+1))$
- Exponential number of cycles $\Rightarrow$ only for small digraphs

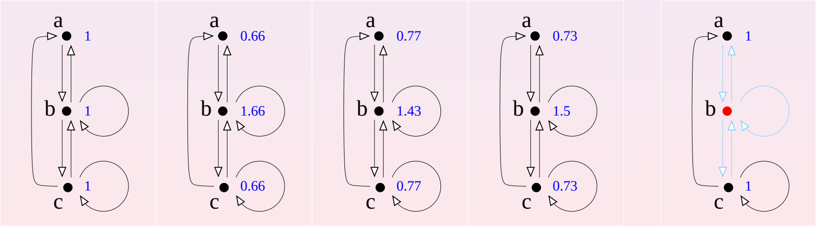
Our heuristic / process number

1. Priority connections: impossibility and transformation
2. Choose of a candidate vertex to receive an agent (to be removed) using a *flow circulation method*
3. Remove that vertex and process all possible vertices including removed vertices and priority connections
4. Repeat 2-3 until processing of all vertices



Our heuristic / process number

1. Priority connections: impossibility and transformation
2. Choose of a candidate vertex to receive an agent (to be removed) using a *flow circulation method*
3. Remove that vertex and process all possible vertices including removed vertices and priority connections
4. Repeat 2-3 until processing of all vertices

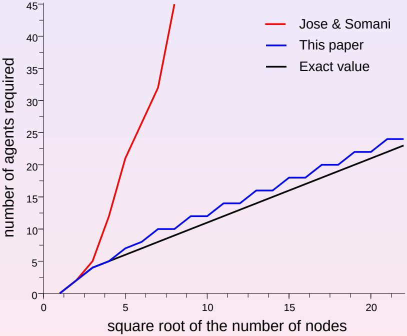


Our heuristic / process number

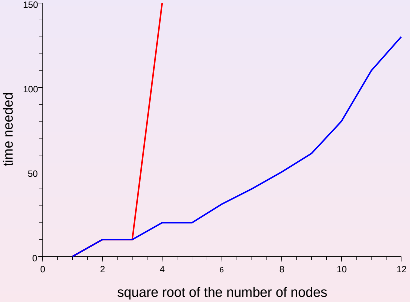
1. Priority connections: impossibility and transformation
2. Choose of a candidate vertex to receive an agent (to be removed) using a *flow circulation method*
3. Remove that vertex and process all possible vertices including removed vertices and priority connections
4. Repeat 2-3 until processing of all vertices

- Heuristic for the process number
- Complexity in $O(n^2(n+m)) \Rightarrow$ large digraphs

Simulation results: $n \times n$ grids

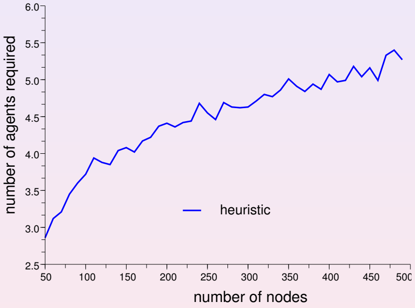


Number of simultaneous agents (break-before-make)

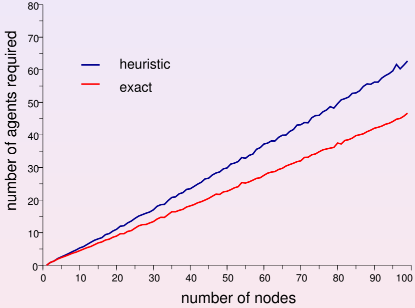


Computation time

Simulation results



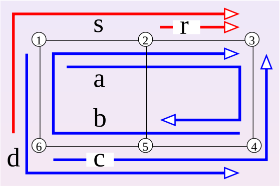
2-digraphs



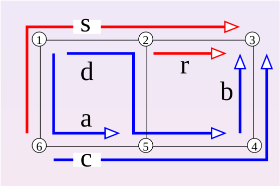
Circular arc graphs

When connections can share Bandwidth

Example: Symmetric grid, where each arc has capacity 2.



Routing 1,
r and *s* cannot be accepted



Routing 2

Theorem **(Coudert, Mazauric, N. [AGT 09])**

When arcs have capacity more than 1, to decide whether the reconfiguration can be done without interruptions is NP-complete.
This is true even if capacities are at most 3.

Recall that if capacities equal 1, this problem is equivalent to recognize a DAG

Further work

Lot of questions remain:

- More simulations, more realistic scenarios
- Multiple classes of services ?
- Time dependent penalities
- Other objectives
  Compromise: simultaneous interruptions / interruption time
- Distributed algorithms
- Heuristics when shared bandwidth
- complexity when ≤ 2 requests can share a link?
- ...