Reconfiguration

Some open problems

D. Coudert[1], F. Huc[1,2], D. Mazauric[1], N. Nisse[1], J-S. Sereni[3,4], and R. Soares[5]

1- MASCOTTE, INRIA, I3S, CNRS, Univ. Nice Sophia, Sophia Antipolis, France

2- TCS-sensor lab, Centre Universitaire d'Informatique, Univ. Genève, Suisse

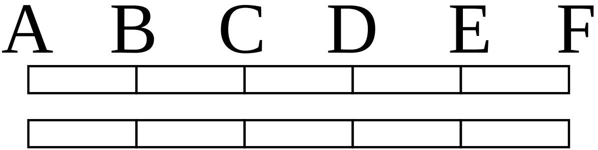
3- LIAFA, CNRS, Univ. D. Diderot, Paris, France

4- KAM, Faculty of Math. and Physics, Charles Univ., Prague, Czech Republic

5- Univ. Fortaleza, Brazil

WDM networks with dynamic traffic

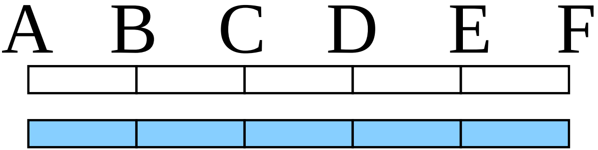
How to handle traffic changes ?



- $+ \; A \rightarrow F$
- $+ \; A \rightarrow C$
- $+ \; E \rightarrow F$
- $- \; A \rightarrow F$
- $+ \; D \rightarrow F$
- $+ \; A \rightarrow B$
- $+ \; B \rightarrow E$

WDM networks with dynamic traffic

How to handle traffic changes ?

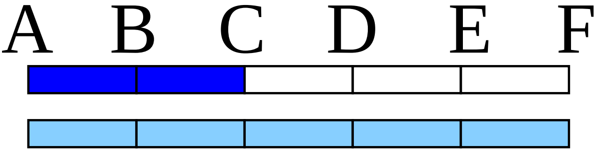


$+\ A \rightarrow F$
$+\ A \rightarrow C$
$+\ E \rightarrow F$
$-\ A \rightarrow F$
$+\ D \rightarrow F$
$+\ A \rightarrow B$
$+\ B \rightarrow E$

Routing of request: $A \rightarrow F$

WDM networks with dynamic traffic

How to handle traffic changes ?

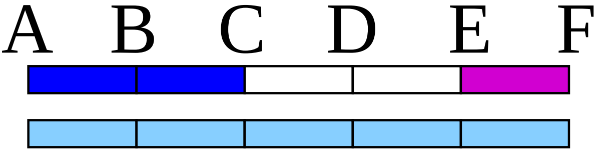


$+ \quad A \rightarrow F$
$+ \quad A \rightarrow C$
$+ \quad E \rightarrow F$
$- \quad A \rightarrow F$
$+ \quad D \rightarrow F$
$+ \quad A \rightarrow B$
$+ \quad B \rightarrow E$

Routing of request: $A \rightarrow C$

WDM networks with dynamic traffic

How to handle traffic changes ?

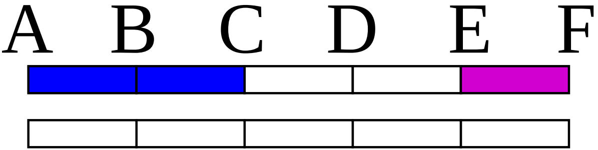


$+ \quad A \rightarrow F$
$+ \quad A \rightarrow C$
$+ \quad E \rightarrow F$
$- \quad A \rightarrow F$
$+ \quad D \rightarrow F$
$+ \quad A \rightarrow B$
$+ \quad B \rightarrow E$

Routing of request: $E \rightarrow F$

WDM networks with dynamic traffic

How to handle traffic changes ?

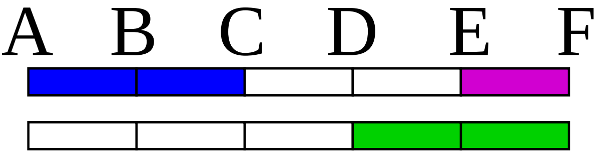


$+\ A \rightarrow F$
$+\ A \rightarrow C$
$+\ E \rightarrow F$
$-\ A \rightarrow F$
$+\ D \rightarrow F$
$+\ A \rightarrow B$
$+\ B \rightarrow E$

Removal of request: $A \rightarrow F$

WDM networks with dynamic traffic

How to handle traffic changes ?

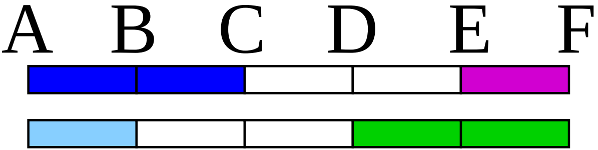


$+ \ A \to F$
$+ \ A \to C$
$+ \ E \to F$
$- \ A \to F$
$+ \ D \to F$
$+ \ A \to B$
$+ \ B \to E$

Routing of request: $D \to F$

WDM networks with dynamic traffic

How to handle traffic changes ?

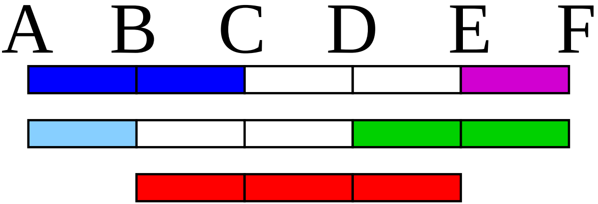


$+\ A \to F$
$+\ A \to C$
$+\ E \to F$
$-\ A \to F$
$+\ D \to F$
$+\ A \to B$
$+\ B \to E$

Routing of request: $A \to B$

WDM networks with dynamic traffic

How to handle traffic changes ?

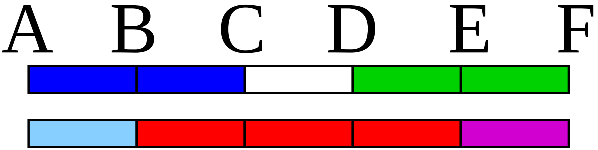


$+ \quad A \rightarrow F$
$+ \quad A \rightarrow C$
$+ \quad E \rightarrow F$
$- \quad A \rightarrow F$
$+ \quad D \rightarrow F$
$+ \quad A \rightarrow B$
$+ \quad B \rightarrow E$

Routing of request: $B \rightarrow E$? ?

WDM networks with dynamic traffic

How to handle traffic changes ?



$+ \quad A \rightarrow F$
$+ \quad A \rightarrow C$
$+ \quad E \rightarrow F$
$- \quad A \rightarrow F$
$+ \quad D \rightarrow F$
$+ \quad A \rightarrow B$
$+ \quad B \rightarrow E$

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 taquin
- 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 lightpaths (route+wavelength)

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

Constraint: A connection is switched only once

What can we do ?

- Reject the new request $\rightarrow$ *blocking probabilities*
- Stop all requests and restart with new "optimal" routing
- Sequence of switching to converge to new routing taquin
- 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 lightpaths (route+wavelength)

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

Constraint: A connection is switched only once

GMPLS

Make-before-break:

Establish new path before switching the connection

$\Longrightarrow$ Destination resources must be available

Break-before-make:

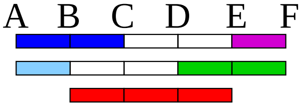
Break connection before establishing the new path

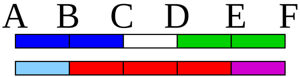
$\Longrightarrow$ Traffic stopped while new path not established

Reconfiguration in WDM networks

Example

Dependency digraph



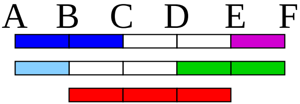


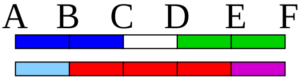
Processing using 1 break-before-make and 1 make-before-break

Reconfiguration in WDM networks

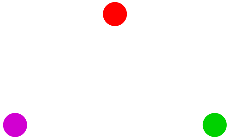
Example

Dependency digraph





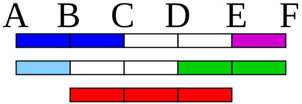
Processing using 1 break-before-make and 1 make-before-break

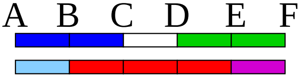


Reconfiguration in WDM networks

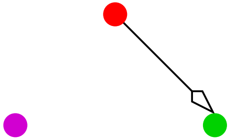
Example

Dependency digraph





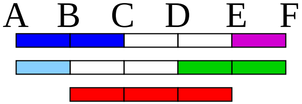
Processing using 1 break-before-make and 1 make-before-break

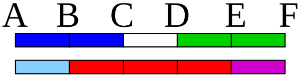


Reconfiguration in WDM networks

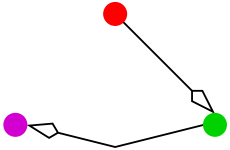
Example

Dependency digraph





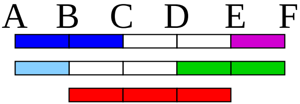
Processing using 1 break-before-make and 1 make-before-break

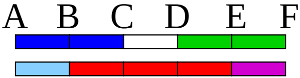


Reconfiguration in WDM networks

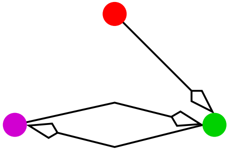
Example

Dependency digraph

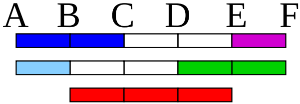


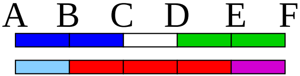


Processing using 1 break-before-make and 1 make-before-break

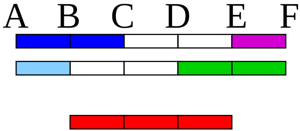


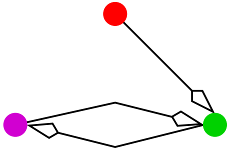
Dependency digraph





Processing using 1 break-before-make and 1 make-before-break

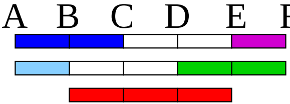


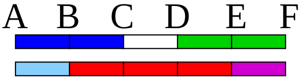


Reconfiguration in WDM networks

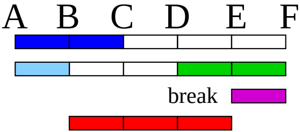
Example

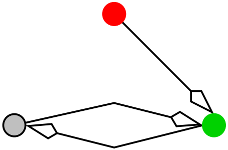
Dependency digraph





Processing using 1 break-before-make and 1 make-before-break

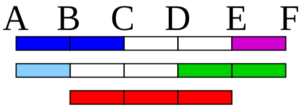


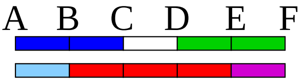


Reconfiguration in WDM networks

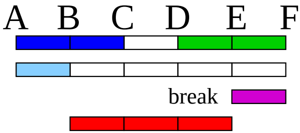
Example

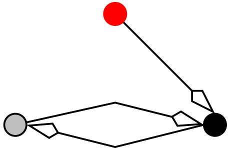
Dependency digraph





Processing using 1 break-before-make and 1 make-before-break

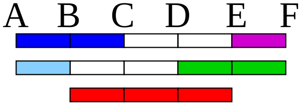


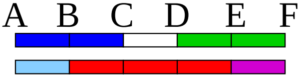


Reconfiguration in WDM networks

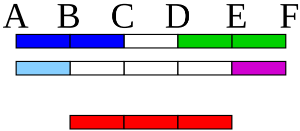
Example

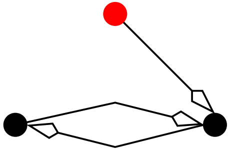
Dependency digraph





Processing using 1 break-before-make and 1 make-before-break

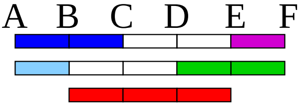


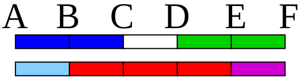


Reconfiguration in WDM networks

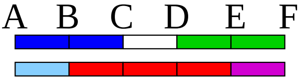
Example

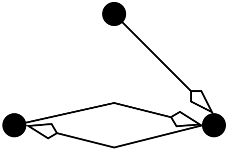
Dependency digraph





Processing using 1 break-before-make and 1 make-before-break

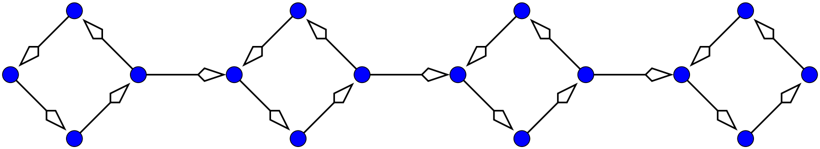




Possible objectives

Minimize overall number of break-before-make

$=$ Minimum Feedback Vertex Set (MFVS), here 4



Minimize number of simultaneous break-before-make

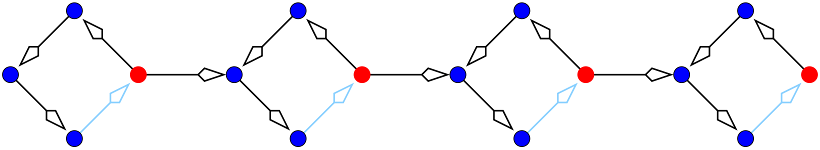
$\sim$ Graph searching problem, cops-and-robber game, pursuit,. . .

- Process number

Possible objectives

Minimize overall number of break-before-make

= Minimum Feedback Vertex Set (MFVS), here 4



Minimize number of simultaneous break-before-make

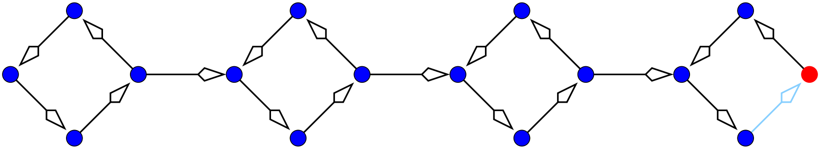
∼ Graph searching problem, cops-and-robber game, pursuit,...

- Process number, here 1
- Gap with MFVS up to $N/2$

Possible objectives

Minimize overall number of break-before-make

$=$ Minimum Feedback Vertex Set (MFVS), here 4



Minimize number of simultaneous break-before-make

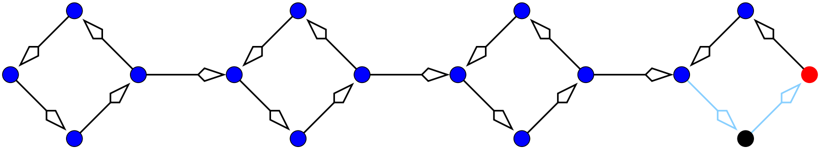
$\sim$ Graph searching problem, cops-and-robber game, pursuit,...

- Process number, here 1
- Gap with MFVS up to $N/2$

Possible objectives

Minimize overall number of break-before-make

$=$ Minimum Feedback Vertex Set (MFVS), here 4



Minimize number of simultaneous break-before-make

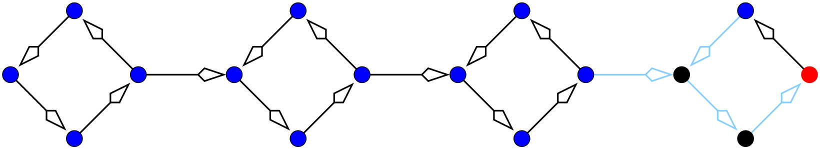
$\sim$ Graph searching problem, cops-and-robber game, pursuit,...

- Process number, here 1
- Gap with MFVS up to $N/2$

Possible objectives

Minimize overall number of break-before-make

$=$ Minimum Feedback Vertex Set (MFVS), here 4



Minimize number of simultaneous break-before-make

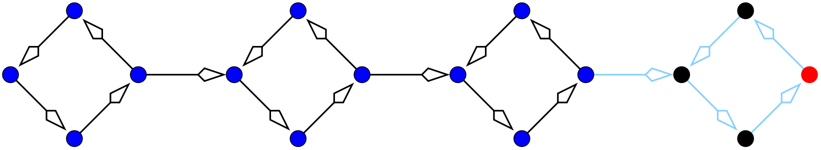
$\sim$ Graph searching problem, cops-and-robber game, pursuit,. . .

- Process number, here 1
- Gap with MFVS up to $N/2$

Possible objectives

Minimize overall number of break-before-make

$=$ Minimum Feedback Vertex Set (MFVS), here 4



Minimize number of simultaneous break-before-make

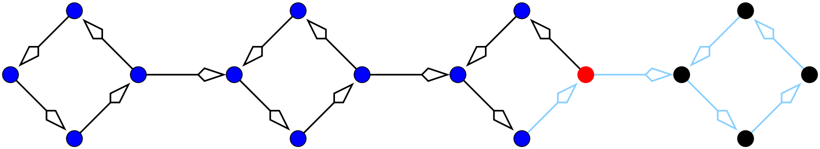
$\sim$ Graph searching problem, cops-and-robber game, pursuit,...

- Process number, here 1
- Gap with MFVS up to $N/2$

Possible objectives

Minimize overall number of break-before-make

$=$ Minimum Feedback Vertex Set (MFVS), here 4



Minimize number of simultaneous break-before-make

$\sim$ Graph searching problem, cops-and-robber game, pursuit,...

- Process number, here 1
- Gap with MFVS up to $N/2$

Process number, *pn*

Rules

R_1 Put an agent on a vertex

= break/interrupt/route on temporary resources a connection

R_2 Process a vertex if all its out-neighbors are either processed or occupied by an agent

= (Re)route a connection when final resources are available

R_3 An agent can be re-used after the processing of the vertex

p-process strategy = strategy to process a (di)graph using at most p agents

Process number = smallest p s.t. G can be p-processed, $pn(G)$

Example: DAG

Rules

R_1 Put an agent on a vertex

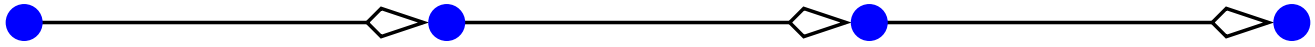
= break/interrupt/route on temporary resources a connection

R_2 Process a vertex if all its out-neighbors are either processed or occupied by an agent

= (Re)route a connection when final resources are available

R_3 An agent can be re-used after the processing of the vertex

Direct path, DAG



Th: If D is a DAG, then $pn(D) = 0$

Example: DAG

Rules

R_1 Put an agent on a vertex

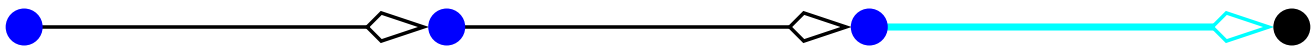
- $=$ break/interrupt/route on temporary resources a connection

R_2 Process a vertex if all its out-neighbors are either processed or occupied by an agent

- $=$ (Re)route a connection when final resources are available

R_3 An agent can be re-used after the processing of the vertex

Direct path, DAG



Th: If D is a DAG, then $pn(D) = 0$

Example: DAG

Rules

R_1 Put an agent on a vertex

= break/interrupt/route on temporary resources a connection

R_2 Process a vertex if all its out-neighbors are either processed or occupied by an agent

= (Re)route a connection when final resources are available

R_3 An agent can be re-used after the processing of the vertex

Direct path, DAG

Th: If D is a DAG, then $pn(D) = 0$

Example: DAG

Rules

R_1 Put an agent on a vertex

= break/interrupt/route on temporary resources a connection

R_2 Process a vertex if all its out-neighbors are either processed or occupied by an agent

= (Re)route a connection when final resources are available

R_3 An agent can be re-used after the processing of the vertex

Direct path, DAG

Th: If D is a DAG, then $pn(D) = 0$

Example: DAG

Rules

R_1 Put an agent on a vertex

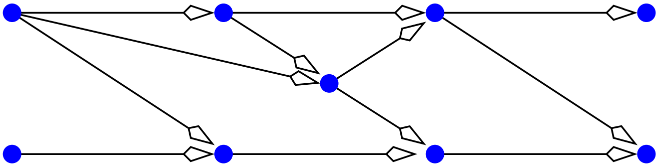
= break/interrupt/route on temporary resources a connection

R_2 Process a vertex if all its out-neighbors are either processed or occupied by an agent

= (Re)route a connection when final resources are available

R_3 An agent can be re-used after the processing of the vertex

Direct path, DAG



Th: If D is a DAG, then $pn(D) = 0$

Digraphs with process number 1

Rules

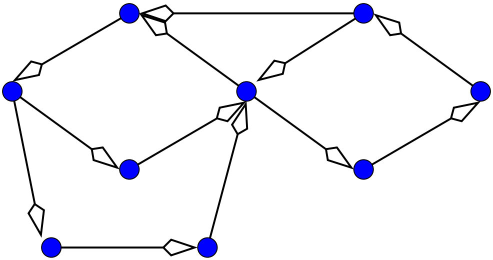
R_1 Put an agent on a vertex

$=$ break/interrupt/route on temporary resources a connection

R_2 Process a vertex if all its out-neighbors are either processed or occupied by an agent

$=$ (Re)route a connection when final resources are available

R_3 An agent can be re-used after the processing of the vertex



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

Digraphs with process number 1

Rules

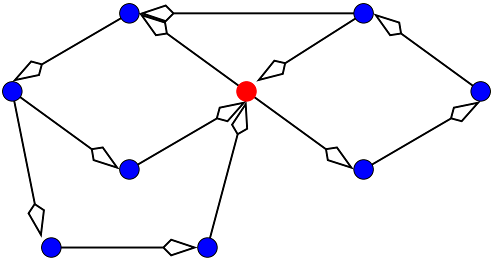
R_1 Put an agent on a vertex

$=$ break/interrupt/route on temporary resources a connection

R_2 Process a vertex if all its out-neighbors are either processed or occupied by an agent

$=$ (Re)route a connection when final resources are available

R_3 An agent can be re-used after the processing of the vertex



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

Digraphs with process number 1

Rules

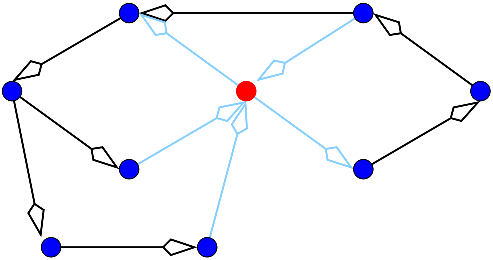
R_1 Put an agent on a vertex

$=$ break/interrupt/route on temporary resources a connection

R_2 Process a vertex if all its out-neighbors are either processed or occupied by an agent

$=$ (Re)route a connection when final resources are available

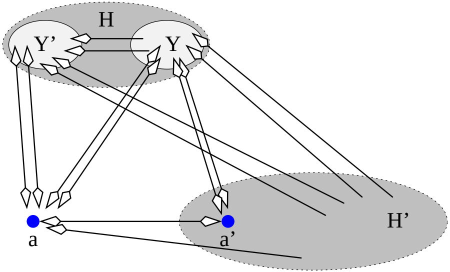
R_3 An agent can be re-used after the processing of the vertex



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

Digraphs with process number 2

$(2, a)$-digraph $=$ digraph that can be 2-processed starting from a



Y DAG

digraph contraction

Y' 1-digraph

Strongly Connected Components + test 1-digraph

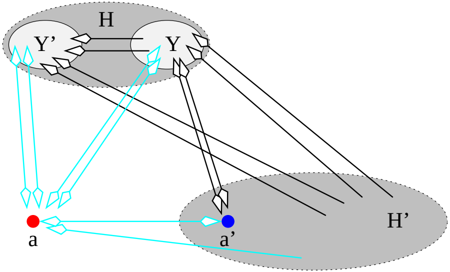
H' $(2, a')$-digraph

Th: $pn(D) = 2$ iff $\exists$ a s.t. D is a $(2, a)$-digraph

Complexity: $(2, a)$-digraph in time $O(N(N + M))$, so $O(N^2(N + M))$

Digraphs with process number 2

$(2, a)$-digraph $=$ digraph that can be 2-processed starting from a



Y DAG

digraph contraction

Y' 1-digraph

Strongly Connected Components

$+$ test 1-digraph

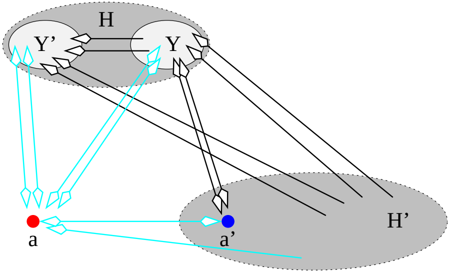
H' $(2, a')$-digraph

Th: $pn(D) = 2$ iff $\exists$ a s.t. D is a $(2, a)$-digraph

Complexity: $(2, a)$-digraph in time $O(N(N + M))$, so $O(N^2(N + M))$

Digraphs with process number 2

$(2, a)$-digraph $=$ digraph that can be 2-processed starting from a



Y DAG
digraph contraction

Y' 1-digraph
Strongly Connected Components
$+$ test 1-digraph

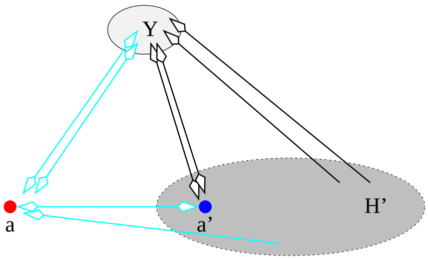
H' $(2, a')$-digraph

Th: $pn(D) = 2$ iff $\exists$ a s.t. D is a $(2, a)$-digraph

Complexity: $(2, a)$-digraph in time $O(N(N + M))$, so $O(N^2(N + M))$

Digraphs with process number 2

$(2, a)$-digraph $=$ digraph that can be 2-processed starting from a



Y DAG

digraph contraction

Y' 1-digraph
Strongly Connected
Components
$+$ test 1-digraph

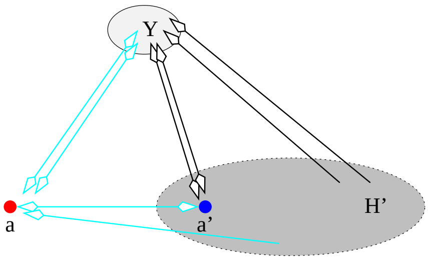
H' $(2, a')$-digraph

Th: $pn(D) = 2$ iff $\exists$ a s.t. D is a $(2, a)$-digraph

Complexity: $(2, a)$-digraph in time $O(N(N + M))$, so $O(N^2(N + M))$

Digraphs with process number 2

(2, a)-digraph $=$ digraph that can be 2-processed starting from a



Y DAG
digraph contraction

Y' 1-digraph
Strongly Connected
Components
$+$ test 1-digraph

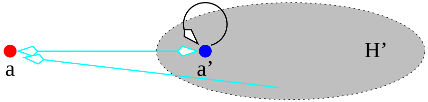
H' $(2, a')$-digraph

Th: $pn(D) = 2$ iff $\exists$ a s.t. D is a $(2, a)$-digraph

Complexity: $(2, a)$-digraph in time $O(N(N+M))$, so
$O(N^2(N+M))$

Digraphs with process number 2

$(2, a)$-digraph $=$ digraph that can be 2-processed starting from a



Y DAG
digraph contraction

Y' 1-digraph
Strongly Connected Components
$+$ test 1-digraph

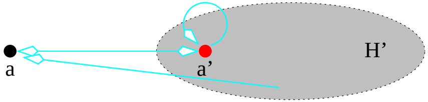
H' $(2, a')$-digraph

Th: $pn(D) = 2$ iff $\exists$ a s.t. D is a $(2, a)$-digraph

Complexity: $(2, a)$-digraph in time $O(N(N + M))$, so $O(N^2(N + M))$

Digraphs with process number 2

$(2, a)$-digraph $=$ digraph that can be 2-processed starting from a



Y DAG
digraph contraction

Y' 1-digraph
Strongly Connected Components
$+$ test 1-digraph

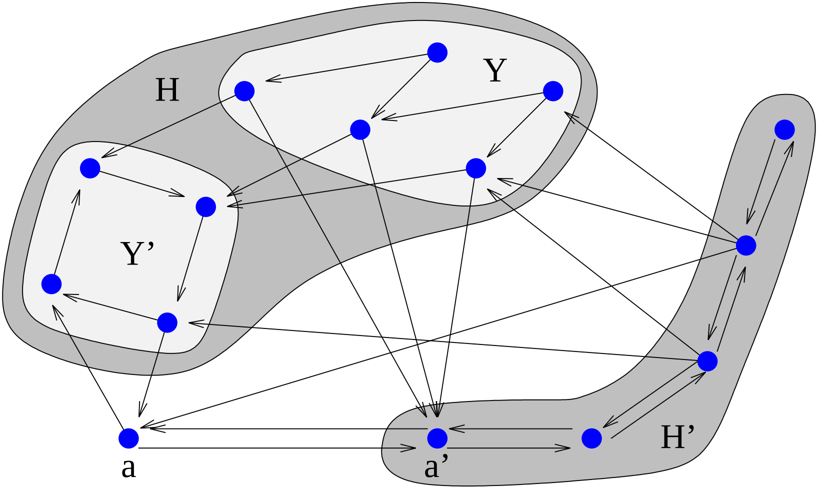
H' $(2, a')$-digraph

Th: $pn(D) = 2$ iff $\exists$ a s.t. D is a $(2, a)$-digraph

Complexity: $(2, a)$-digraph in time $O(N(N + M))$, so $O(N^2(N + M))$

Digraphs with process number 2

Example



Process number: what is known

Related parameters

- Pathwidth, *pw* [Robertson & Seymour, JCTB, 1983]
- Node search number, *ns* [Kirousis & Papadimitriou, TCS, 1986]
- Vertex separation, *vs*

Relations

- $pw(G) = vs(G) = ns(G) - 1$
- $vs(D) \leq pn(D) \leq vs(D) + 1$ [CPPS05, CoSe07]

Complexity

- NP-Hard
- Not APX
  - $=$ No polynomial time constant factor approximation algorithm

- Characterization of (di)graphs with process number 0, 1, 2
- Distributed algorithm for trees

Two classes of services

Priority connections

- Refuse *by contract* (SLA) break-before-make

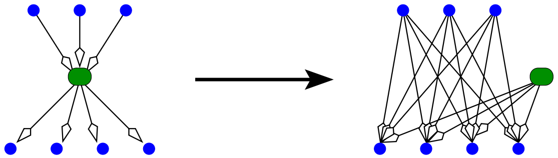
Impossibility

- Direct cycle of priority connections in the dependency digraph

$\Rightarrow$ *Small* number of such connections

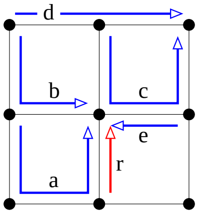
- Partition into strongly connected components, $O(N + M)$

Transformation

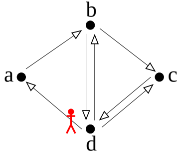


$\Rightarrow$ Same problem to solve

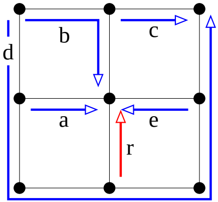
Example with priority connection d



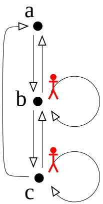
Routing 1



Dependency digraph, $pn = 1$



Routing 2



Without d, $pn = 2$

Previous heuristic

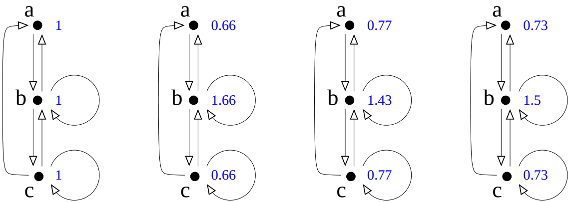
[Jose and Somani, DRCN 03]

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

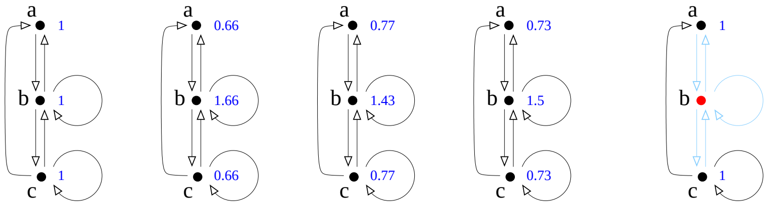
Our heuristic / process number

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

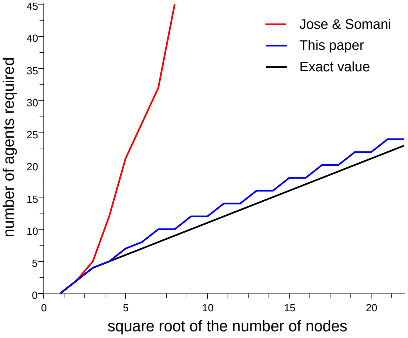


Our heuristic / process number

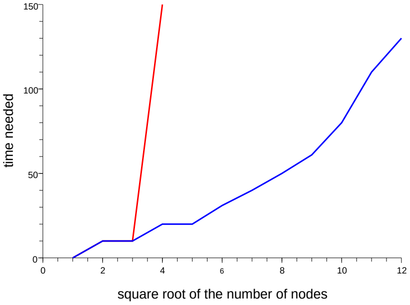
1. Priority connections: impossibility and transformation
2. Choose of a candidate vertex to receive an agent (to be removed) using a *flow circulation method*
3. Remove that vertex and process all possible vertices including removed vertices and priority connections
4. Repeat 2-3 until processing of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

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

Simulation results: $n \times n$ grids

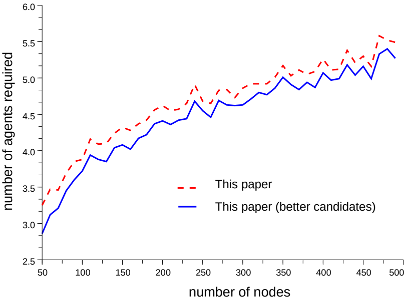


Number of simultaneous agents (break-before-make)

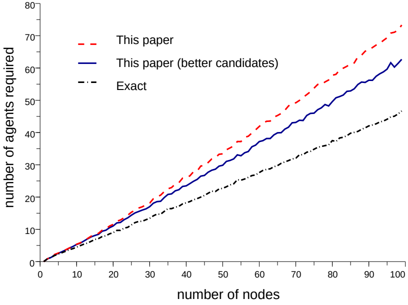


Computation time

Simulation results



2-digraphs



Circular arc graphs

Open problems

Directed graphs

- Number of steps (Ronan Soares)
  - Fixed number of agents, minimize # steps
  - Fixed number of steps, minimize # agents
  - SLA with fixed or time dependent penalities
- Variable number of agents
  - Use available temporary resources (lightpath).
  - Use protection resources: dedicated, shared, path, segment,...
- More general dependency digraphs
  - Sub-wavelength (grooming), LSP
- Parallel operations on a cycle ?
- Smooth reconfiguration ?
- How to compute best destination configuration ?
- Extension to SOA ?

Open problems

Directed graphs

- Number of steps (Ronan Soares)
  - Fixed number of agents, minimize # steps
  - Fixed number of steps, minimize # agents
  - SLA with fixed or time dependent penalities
- Variable number of agents
  - Use available temporary resources (lightpath).
  - Use protection resources: dedicated, shared, path, segment,...
- More general dependency digraphs
  - Sub-wavelength (grooming), LSP
- Parallel operations on a cycle ?
- Smooth reconfiguration ?
- How to compute best destination configuration ?
- Extension to SOA ?

Open problems

Directed graphs

- Number of steps (Ronan Soares)
  - Fixed number of agents, minimize # steps
  - Fixed number of steps, minimize # agents
  - SLA with fixed or time dependent penalities
- Variable number of agents
  - Use available temporary resources (lightpath).
  - Use protection resources: dedicated, shared, path, segment,...
- More general dependency digraphs
  - Sub-wavelength (grooming), LSP
- Parallel operations on a cycle ?
- Smooth reconfiguration ?
- How to compute best destination configuration ?
- Extension to SOA ?

Open problems

Directed graphs

- Number of steps (Ronan Soares)
  - Fixed number of agents, minimize # steps
  - Fixed number of steps, minimize # agents
  - SLA with fixed or time dependent penalities
- Variable number of agents
  - Use available temporary resources (lightpath).
  - Use protection resources: dedicated, shared, path, segment,...
- More general dependency digraphs
  - Sub-wavelength (grooming), LSP
- Parallel operations on a cycle ?
- Smooth reconfiguration ?
- How to compute best destination configuration ?
- Extension to SOA ?

Open problems

Directed graphs

- Number of steps (Ronan Soares)
  - Fixed number of agents, minimize # steps
  - Fixed number of steps, minimize # agents
  - SLA with fixed or time dependent penalities
- Variable number of agents
  - Use available temporary resources (lightpath).
  - Use protection resources: dedicated, shared, path, segment,...
- More general dependency digraphs
  - Sub-wavelength (grooming), LSP
- Parallel operations on a cycle ?
- Smooth reconfiguration ?
- How to compute best destination configuration ?
- Extension to SOA ?

Open problems

Directed graphs

- Number of steps (Ronan Soares)
  - Fixed number of agents, minimize # steps
  - Fixed number of steps, minimize # agents
  - SLA with fixed or time dependent penalities
- Variable number of agents
  - Use available temporary resources (lightpath).
  - Use protection resources: dedicated, shared, path, segment,...
- More general dependency digraphs
  - Sub-wavelength (grooming), LSP
- Parallel operations on a cycle ?
- Smooth reconfiguration ?
- How to compute best destination configuration ?
- Extension to SOA ?

Open problems

Directed graphs

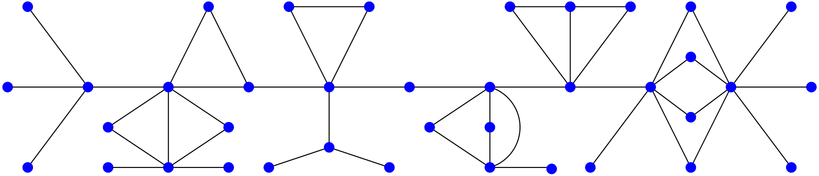
- Number of steps (Ronan Soares)
  - Fixed number of agents, minimize # steps
  - Fixed number of steps, minimize # agents
  - SLA with fixed or time dependent penalities
- Variable number of agents
  - Use available temporary resources (lightpath).
  - Use protection resources: dedicated, shared, path, segment,...
- More general dependency digraphs
  - Sub-wavelength (grooming), LSP
- Parallel operations on a cycle ?
- Smooth reconfiguration ?
- How to compute best destination configuration ?
- Extension to SOA ?

Open problems

Undirected graphs

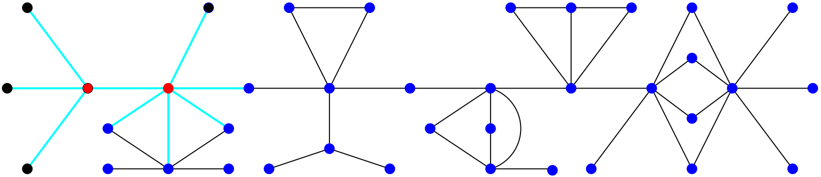
- Characterization of graphs with process number 3
- Relation/difference with node search number
    - Done for trees

Graphs with process number 2



- G minus a path = stars
- Characterization: 15 excluded minors or a simple linear algorithm
- G s.t. $pn(G) = 3$: at least 185.266 excluded minors

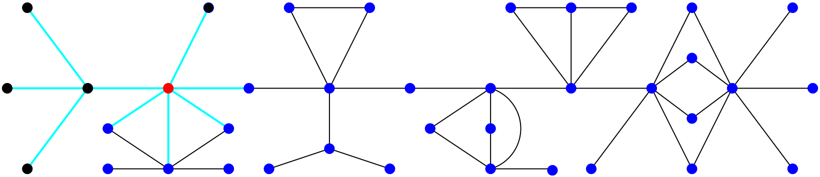
Graphs with process number 2



- G minus a path = stars
- Characterization: 15 excluded minors or a simple linear algorithm

- G s.t. $pn(G) = 3$: at least 185.266 excluded minors

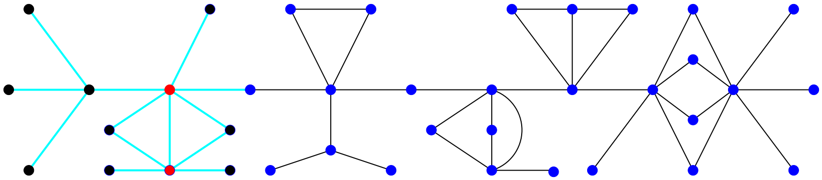
Graphs with process number 2



- G minus a path = stars
- Characterization: 15 excluded minors or a simple linear algorithm

- G s.t. $pn(G) = 3$: at least 185.266 excluded minors

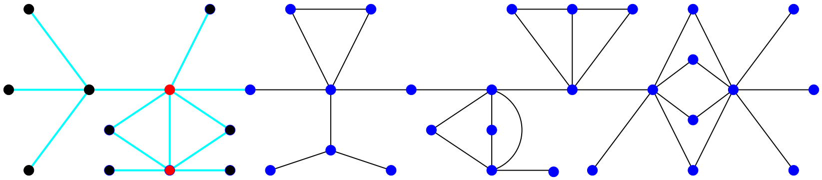
Graphs with process number 2



- G minus a path = stars
- Characterization: 15 excluded minors or a simple linear algorithm

- G s.t. $pn(G) = 3$: at least 185.266 excluded minors

Graphs with process number 2

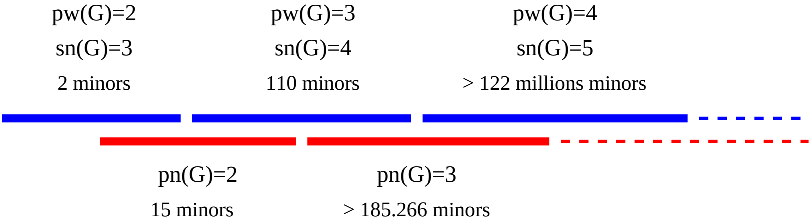


- G minus a path = stars
- Characterization: 15 excluded minors or a simple linear algorithm

- G s.t. $pn(G) = 3$: at least 185.266 excluded minors

Process number *vs* node search number

vertex separation	$vs(D) \leq pn(D) \leq vs(D) + 1$
	$vs(G) \leq pn(G) \leq vs(G) + 1$
pathwidth	$pw(G) \leq pn(G) \leq pw(G) + 1$
node search number	$sn(G) - 1 \leq pn(G) \leq sn(G)$



Process number *vs* node search number

vertex separation	$vs(D) \leq pn(D) \leq vs(D) + 1$
	$vs(G) \leq pn(G) \leq vs(G) + 1$
pathwidth	$pw(G) \leq pn(G) \leq pw(G) + 1$
node search number	$sn(G) - 1 \leq pn(G) \leq sn(G)$

pw(G)=2
sn(G)=3
2 minors

pw(G)=3
sn(G)=4
110 minors

pw(G)=4
sn(G)=5
> 122 millions minors

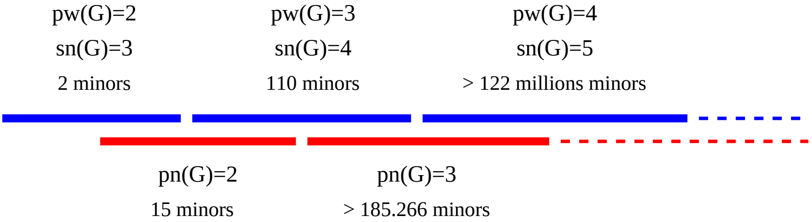
pn(G)=2
15 minors

pn(G)=3
> 185.266 minors

Open question: Characterize G s.t. $pn(G) = sn(G)$

Process number *vs* node search number

vertex separation	$vs(D) \leq pn(D) \leq vs(D) + 1$
	$vs(G) \leq pn(G) \leq vs(G) + 1$
pathwidth	$pw(G) \leq pn(G) \leq pw(G) + 1$
node search number	$sn(G) - 1 \leq pn(G) \leq sn(G)$



Open question: Characterize G s.t. $pn(G) = sn(G)$

$\longrightarrow$ done for trees (with Huc & Mazauric)


NON
AUX CADENCES
INFERNALES
ZRRÔÔÔ
ZZ
RRÔÔÔ
Z
NON
NON