Graph Searching and Routing Reconfiguration in WDM Networks

Nicolas Nisse

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

joint works with: N. Cohen, D. Coudert, F. Huc, D. Mazauric, N. Nepomuceno, J.-S. Sereni, R.P. Soares

GRASTA, Banff, Canada, October 11*th*, 2012

Outline

1. "Practical" motivations
2. Processing and Graph Searching Games
3. "New" problems
   - Tradeoff: # agents vs. # occupied vertices
   - Computation: approximation and heuristic
4. Variants and Open questions

Routing in WDM Networks

Physical Network, Links provide several wavelengths

multi-digraph $G = (V, E)$

an arc $(u, v) \Leftrightarrow$ one wavelength on the link (u, v)

Routing of a set of requests/connections

set of requests $\mathcal{R} \subseteq V \times V$

routing: for each request (u, v),

a path from u to v and 1 wavelength.

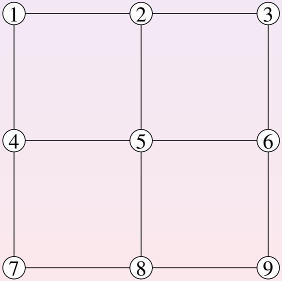
Problem: due to dynamicity of traffic, failures

how to maintain an efficient routing?

What happens in "real" world

Variation of traffic + dynamicity induced by failures
$\Rightarrow$ Online processes to route all requests: e.g., greedy routing

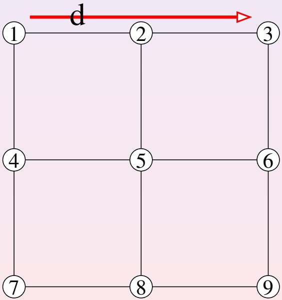
Example of a grid network with directed symmetric links (capacity 1)



What happens in "real" world

Variation of traffic + dynamicity induced by failures
$\Rightarrow$ Online processes to route all requests: e.g., greedy routing

Example of a grid network with directed symmetric links (capacity 1)

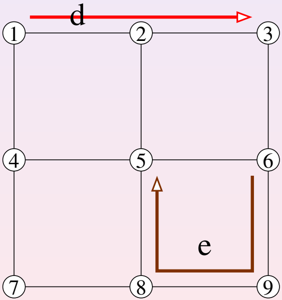


Request d : $1 \rightarrow 3$

What happens in "real" world

Variation of traffic + dynamicity induced by failures
$\Rightarrow$ Online processes to route all requests: e.g., greedy routing

Example of a grid network with directed symmetric links (capacity 1)

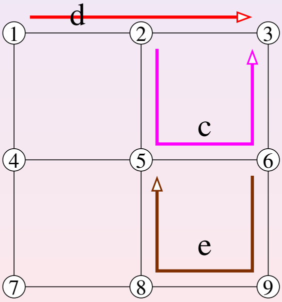


Request d : $1 \rightarrow 3$
Request e : $6 \rightarrow 5$

What happens in "real" world

Variation of traffic + dynamicity induced by failures
⇒ Online processes to route all requests: e.g., greedy routing

Example of a grid network with directed symmetric links (capacity 1)

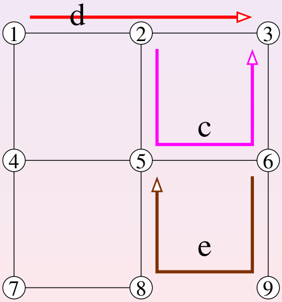


Request d : $1 \rightarrow 3$
Request e : $6 \rightarrow 5$
Request c : $2 \rightarrow 3$

What happens in "real" world

Variation of traffic + dynamicity induced by failures
$\Rightarrow$ Online processes to route all requests: e.g., greedy routing

Example of a grid network with directed symmetric links (capacity 1)

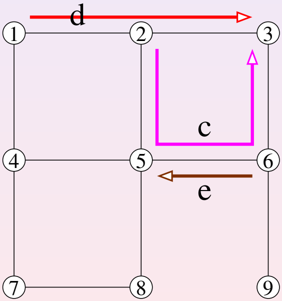


Request d : $1 \rightarrow 3$
Request e : $6 \rightarrow 5$
Request c : $2 \rightarrow 3$
Failure of link $\{8, 9\}$

What happens in "real" world

Variation of traffic + dynamicity induced by failures
$\Rightarrow$ Online processes to route all requests: e.g., greedy routing

Example of a grid network with directed symmetric links (capacity 1)

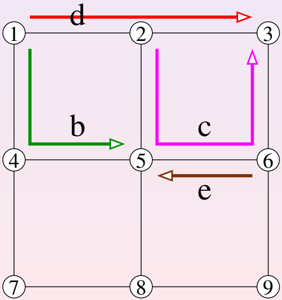


Request d : $1 \to 3$
Request e : $6 \to 5$
Request c : $2 \to 3$
Rerouting of request e

What happens in "real" world

Variation of traffic + dynamicity induced by failures
$\Rightarrow$ Online processes to route all requests: e.g., greedy routing

Example of a grid network with directed symmetric links (capacity 1)

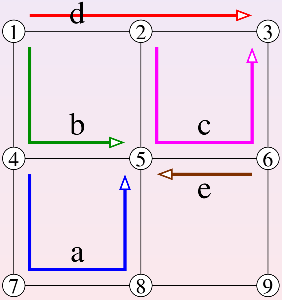


Request d : $1 \rightarrow 3$
Request e : $6 \rightarrow 5$
Request c : $2 \rightarrow 3$
Request b : $1 \rightarrow 5$
New link $\{8, 9\}$

What happens in "real" world

Variation of traffic + dynamicity induced by failures
$\Rightarrow$ Online processes to route all requests: e.g., greedy routing

Example of a grid network with directed symmetric links (capacity 1)

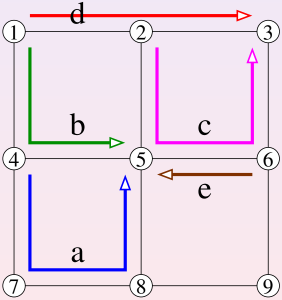


Request d : $1 \rightarrow 3$
Request e : $6 \rightarrow 5$
Request c : $2 \rightarrow 3$
Request b : $1 \rightarrow 5$
Request a : $4 \rightarrow 5$

What happens in "real" world

Variation of traffic + dynamicity induced by failures
⇒ Online processes to route all requests: e.g., greedy routing

Example of a grid network with directed symmetric links (capacity 1)



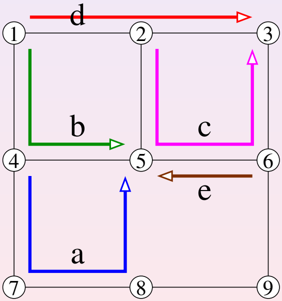
Leads to a poor usage of ressources

Sometimes greedy routing is impossible
even if several requests are allowed to be moved

What happens in "real" world

Variation of traffic + dynamicity induced by failures
⇒ Online processes to route all requests: e.g., greedy routing

Example of a grid network with directed symmetric links (capacity 1)



Leads to a poor usage of ressources

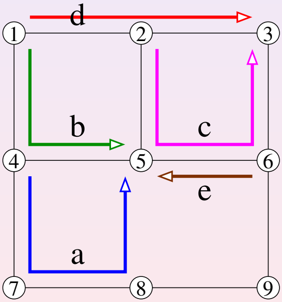
Sometimes greedy routing is impossible
even if several requests are allowed to be moved

If $\{5, 8\}$ fails:
Move-to-Vacant impossible

What happens in "real" world

Variation of traffic + dynamicity induced by failures
⇒ Online processes to route all requests: e.g., greedy routing

Example of a grid network with directed symmetric links (capacity 1)



2 questions arise:

1. Compute new routing
2. Switch from initial routing to final one

We focus on 2

Two ways of switching one request

Make-before-break:

Establish new path before switching the connection

$\Longrightarrow$ Destination resources must be available

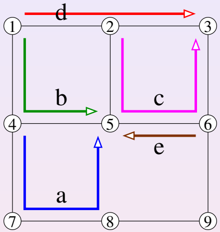
Break-before-make:

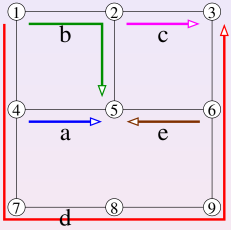
Break connection before establishing the new path

$\Longrightarrow$ Traffic stopped $=$ interruption

The Routing Reconfiguration Problem

How to go from the initial routing (left) to the final one (right)?





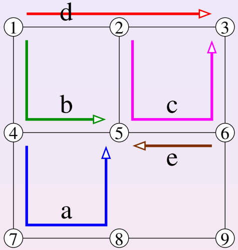
Inputs: Set of connection requests + current & new routing

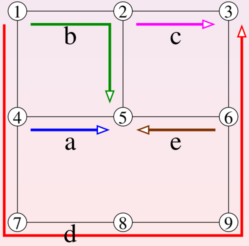
Output: Scheduling for switching connection requests from current to new routes

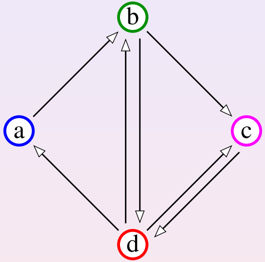
Constraint: A connection is switched **only once**

ObjectiveS Number of Interruptions (detailled later)

Dependency digraph



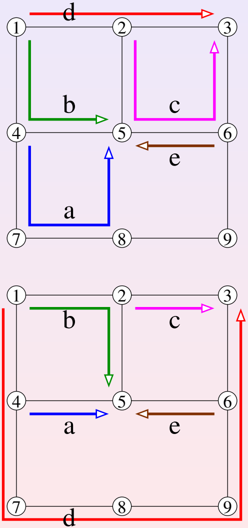


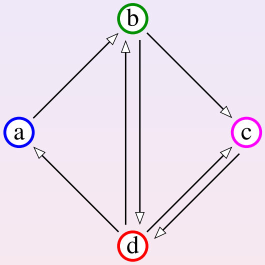


$u \rightarrow v$
if u needs ressources of v
if v must be rerouted/interrupted before u

b needs ressources used by d and c

Dependency digraph

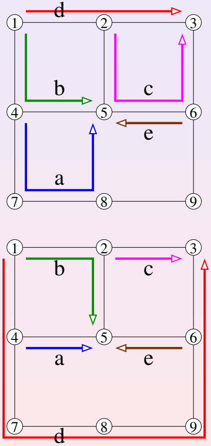


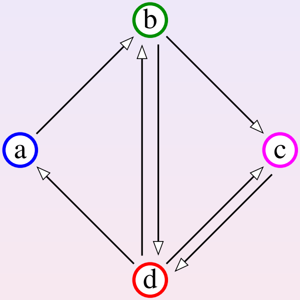


Dependancy Digraph

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

A game on dependency digraph

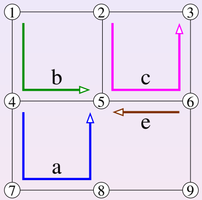


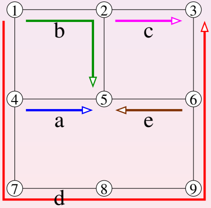


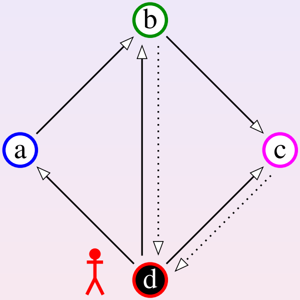
cyclic dependancies

$\Rightarrow$ Interruption required

A game on dependency digraph

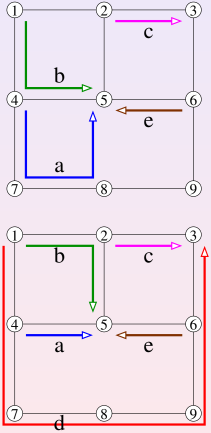


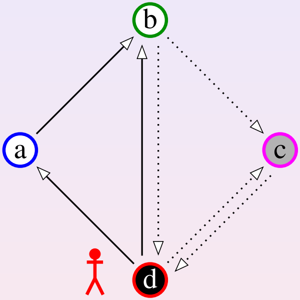




put an agent on node *d*
break request *d*

A game on dependency digraph

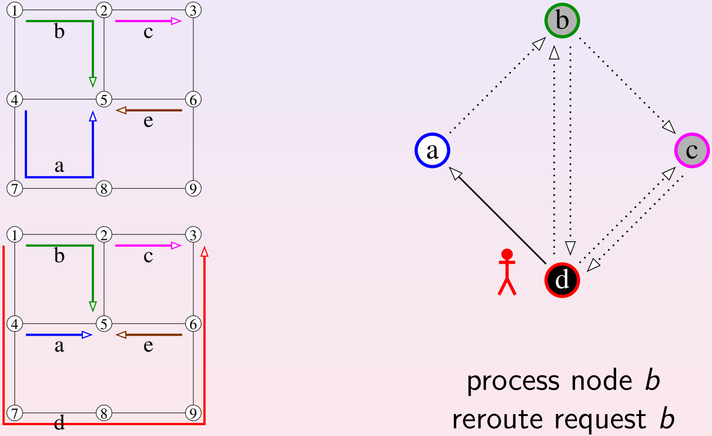




process node c

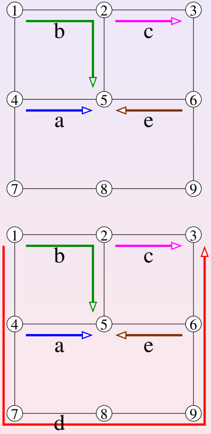
reroute request c

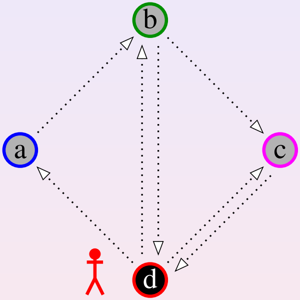
A game on dependency digraph



process node b
reroute request b

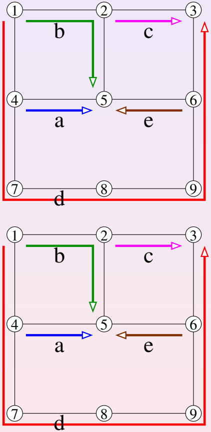
A game on dependency digraph

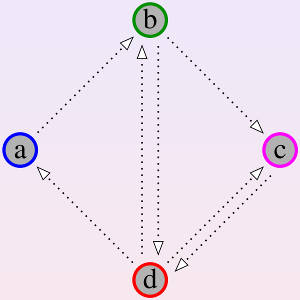




process node a
reroute request a

A game on dependency digraph





process node *d*
and remove agent
route request *d*

Processing Game

Game with Agents on the Dependency digraph D

Sequence of three basic operations,. . .

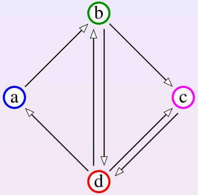
1. Place a searcher at a node = **interrupt the request**;
2. Process a node if all its out-neighbors are either processed or occupied by an agent = **(Re)route a connection when final resources are available**;

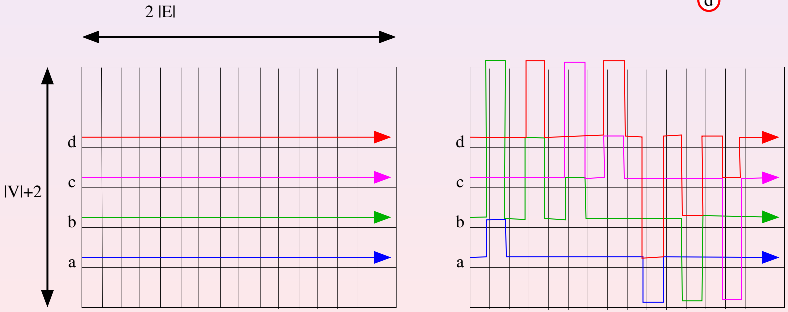
   A processed node is removed from the dependency digraph.
3. Remove an agent from a node, only if it has been processed.

. . . that must result in processing all nodes

From now on: problem on digraphs

Any directed graph is a dependency digraph

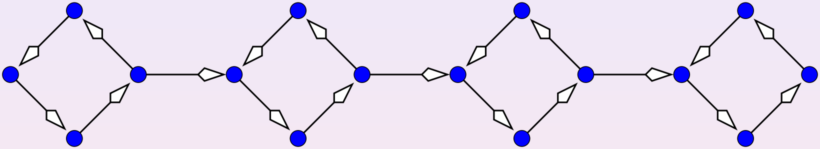




Two possible objectives

Minimize overall number of interrupted requests

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

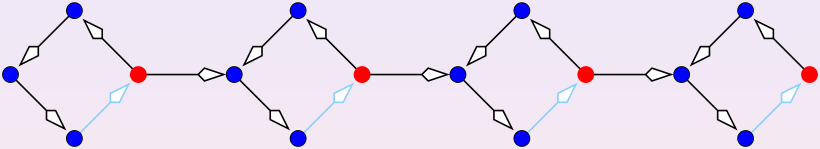


Remarks: MFVS is NP-complete and non APX in digraphs
2-approx in undirected (directed symmetric) graphs

Two possible objectives

Minimize overall number of interrupted requests

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

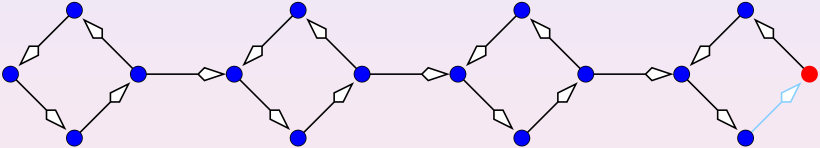


Remarks: MFVS is NP-complete and non APX in digraphs
2-approx in undirected (directed symmetric) graphs

Two possible objectives

Minimize overall number of interrupted requests

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



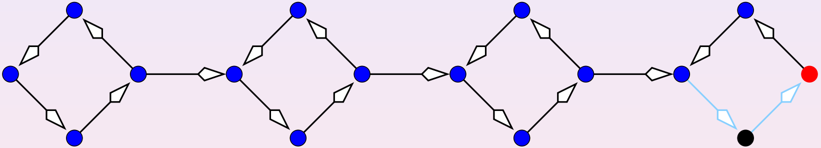
Minimize number of simultaneous interrupted requests

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

Two possible objectives

Minimize overall number of interrupted requests

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



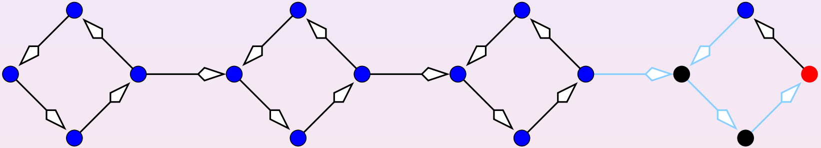
Minimize number of simultaneous interrupted requests

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

Two possible objectives

Minimize overall number of interrupted requests

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



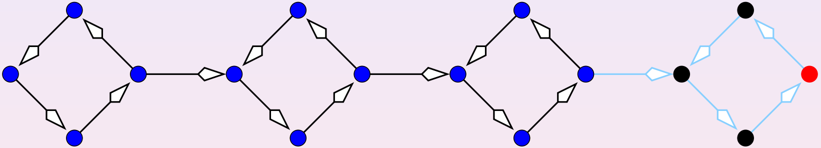
Minimize number of simultaneous interrupted requests

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

Two possible objectives

Minimize overall number of interrupted requests

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



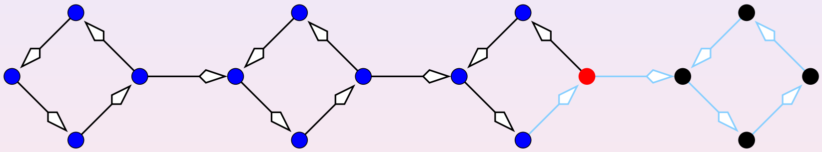
Minimize number of simultaneous interrupted requests

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

Two possible objectives

Minimize overall number of interrupted requests

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



Minimize number of **simultaneous** interrupted requests

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

Routing Reconfiguration, Process number

Game with Agents on the Dependency digraph D

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**;

   A processed node is removed from the dependency digraph.
3. Remove an agent from a node, after having processed it.

...that must result in processing all nodes

Process number = **min. number of simultaneous interruptions**

$pn(D)$=min number of agents used during the strategy

Min. Feedback Vertex Set = **min. total number of interruptions**

$mfvs(D)$= min total number of occupied vertices during the strategy

Outline

1. "Practical" motivations
2. Processing and Graph Searching Games
3. "New" problems
   - Tradeoff: # agents vs. # occupied vertices
   - Computation: approximation and heuristic
4. Variants and Open questions

Processing Game

Let D be a digraph

Sequence of three basic operations,...

1. Place a searcher at a node;
2. Process a node if all its out-neighbors are either processed or occupied by an agent; the node does not need to be occupied!
3. Remove an agent from a node, only if it has been processed.

...that must result in processing all nodes

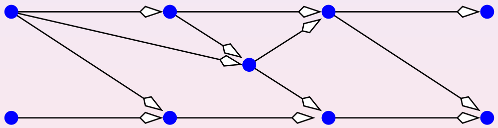
Process number, $pn(D)$=min number of agents used during the strategy

Simple Example 1: DAG

Only one operation is used

1. Place a searcher at a node;
2. Process a node if all its out-neighbors are either processed or occupied by an agent;
3. Remove an agent from a node, only if it has been processed.

DAG



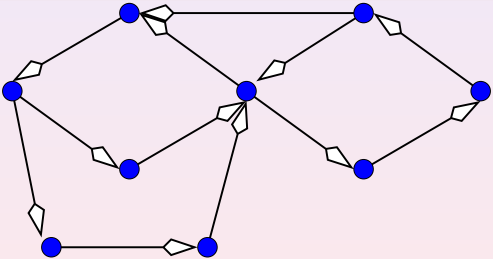
Theorem

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

Simple Example 2: process number 1

One agent is used

1. Place a searcher at a node;
2. Process a node if all its out-neighbors are either processed or occupied by an agent;
3. Remove an agent from a node, only if it has been processed.



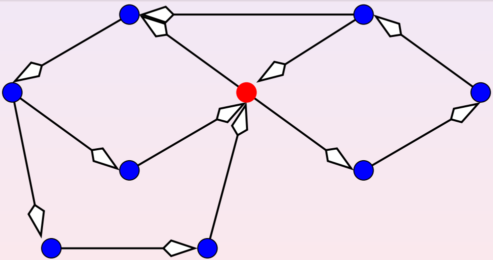
Theorem

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

Simple Example 2: process number 1

One agent is used

1. Place a searcher at a node;
2. Process a node if all its out-neighbors are either processed or occupied by an agent;
3. Remove an agent from a node, only if it has been processed.



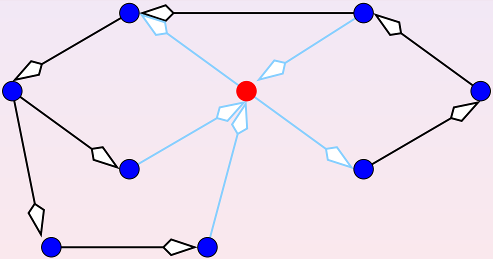
Theorem

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

Simple Example 2: process number 1

One agent is used

1. Place a searcher at a node;
2. Process a node if all its out-neighbors are either processed or occupied by an agent;
3. Remove an agent from a node, only if it has been processed.



Theorem

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

Processing Game vs. Graph Searching

In undirected graphs or "symmetric" digraphs

Node-search (a.k.a. helicopter game) $\Leftrightarrow$ pathwidth

Invisible fugitive moves along edges

1. Place a searcher at a node;
2. Remove an agent from a node (if no recontamination). (monotone)

Capture if a cop lands on the fugitive and it cannot flee (it is surrounded)

Monotone Process Number

Invisible fugitive moves along edges

1. Place a searcher at a node;

3. Remove an agent from a node if no recontamination.

Processing Game vs. Graph Searching

In undirected graphs or "symmetric" digraphs

Node-search (a.k.a. helicopter game) $\Leftrightarrow$ pathwidth

Invisible fugitive moves along edges

1. Place a searcher at a node;
2. Remove an agent from a node (if no recontamination). (monotone)

Capture if a cop lands on the fugitive and it cannot flee (it is surrounded)

Monotone Process Number

Invisible fugitive moves along edges

1. Place a searcher at a node;
2. Process a node (capture) if its neighbors are occupied;
3. Remove an agent from a node if no recontamination.

Processing Game vs. Graph Searching

In undirected graphs or "symmetric" digraphs

Node-search (a.k.a. helicopter game) $\Leftrightarrow$ pathwidth

Invisible fugitive moves along edges

1. Place a searcher at a node;
2. Remove an agent from a node (if no recontamination). (monotone)

Capture if a cop lands on the fugitive and it cannot flee (it is surrounded)

Monotone Process Number

Invisible fugitive moves along edges and must always move

1. Place a searcher at a node;

3. Remove an agent from a node if no recontamination.

Capture if a cop lands on the fugitive and it cannot flee (it is surrounded)

OR if it is surrounded

Processing Game vs. Graph Searching

In directed graphs

directed node-search $\Leftrightarrow$ directed pathwidth

Invisible fugitive moves **along arcs**

1. Place a searcher at a node;
2. Remove an agent from a node (if no recontamination). (monotone)

Capture if a cop lands on the fugitive and it cannot flee (its out-neighbor is occupied)

Monotone Process Number

Invisible fugitive moves backward arcs

1. Place a searcher at a node;

3. Remove an agent from a node if no recontamination.

Processing Game vs. Graph Searching

In directed graphs

directed node-search $\Leftrightarrow$ directed pathwidth

Invisible fugitive moves **along arcs**

1. Place a searcher at a node;
2. Remove an agent from a node (if no recontamination). (monotone)

Capture if a cop lands on the fugitive and it cannot flee (its out-neighbor is occupied)

Monotone Process Number

Invisible fugitive moves backward arcs

1. Place a searcher at a node;
2. Process a node (capture) if its out-neighbors are occupied or processed;
3. Remove an agent from a node if no recontamination.

Processing Game vs. Graph Searching

In directed graphs

directed node-search $\Leftrightarrow$ directed pathwidth

Invisible fugitive moves **along arcs**

1. Place a searcher at a node;
2. Remove an agent from a node (if no recontamination). (monotone)

Capture if a cop lands on the fugitive and it cannot flee (its out-neighbor is occupied)

Monotone Process Number

Invisible fugitive moves backward arcs and must always move

1. Place a searcher at a node;

3. Remove an agent from a node if no recontamination.

Capture if a cop lands on the fugitive and it cannot flee (it is surrounded)

OR if it is surrounded or stuck in a not strongly-connected component

Monotone process number

Related parameter of directed (and undirected) graphs

vertex separation vs = (directed) pathwidth **Kinnersley [IPL 92]**

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

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

$pw(G) \leq monotone\text{-}pn(G) \leq pw(G) + 1$ undirected graph G
$dpw(D) \leq monotone\text{-}pn(D) \leq pw(D) + 1$ directed graph D

Complexity

- NP-Complete, Not APX **(Coudert & Sereni, 2007)**
- Characterization of digraphs with process number 0, 1, 2 **(Coudert & Sereni, 2007)**
- distributed $O(n \log n)$-time exact algorithm in trees **(Coudert, Huc, Mazauric [Algorithmica 12])**

Monotonicity

Theorem **(N., Soares, 2012)**

For any digraph D, $pn(D) = monotone\text{-}pn(D)$

Process-decompositon

Sequence of pairs $P = ((W_1, X_1), \cdots, (W_t, X_t))$ such that:

- $(X_1, \cdots, X_t)$ is a partition of $V \setminus \bigcup_{i=1}^{t} W_i$;
- $\forall i \leq j \leq t$, $W_i \cap W_k \subseteq W_j$;
- X_i induces a Directed Acyclic Graph (DAG), for any $1 \leq i \leq t$;
- $\forall (u, v) \in A$, $\exists j \leq i$ such that $v \in W_j \cup X_j$ and $u \in W_i \cup X_i$.

Width $= \max_{1 \leq i \leq n} |W_i|$

$pn(D) =$ min. width among all decompositions

$\Rightarrow pn(D) = pn(\bar{D})$ ($\bar{D}$ is D where arcs have been reversed)

To summarize

	undirected graphs		**directed graphs**		
fugitive	move along edges		move along arcs		
		must move		must be able to move	must move in SCC
invisible	pw	pn	dpw	pn	?
	monotone				
visible	*tw*		DAG-width	visible-*pn*	$\approx$ *dtw*
	monotone	?	*not monotone*		
			no ratio	?	$dtw \leq 3br$

Table: Classification of the graph searching games

Outline

1. "Practical" motivations
2. Processing and Graph Searching Games
3. "New" problems
   - Tradeoff: # agents vs. # occupied vertices
   - Computation: approximation and heuristic
4. Variants and Open questions

Routing Reconfiguration, Process number

Game with Agents on the Dependency digraph D

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**;

   A processed node is removed from the dependency digraph.
3. Remove an agent from a node, after having processed it.

...that must result in processing all nodes

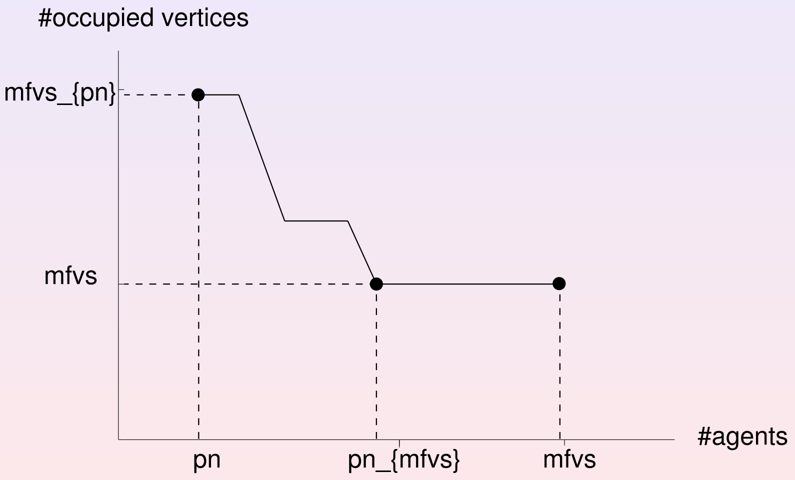
Process number = **min. number of simultaneous interruptions**

$pn(D)$=min number of agents used during the strategy

Min. Feedback Vertex Set = **min. total number of interruptions**

$mfvs(D)$= min total number of occupied vertices during the strategy

Tradeoff: total/ max simultaneous interruptions



Complexity

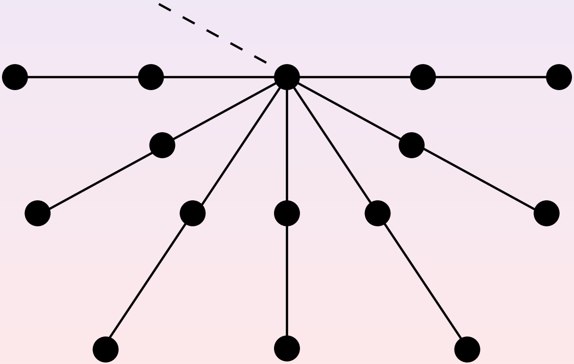
- Smallest number of agents such that the number of occupied vertices is minimum $= pn_{mfvs}(D)$
- $\mu = \frac{pn_{mfvs}(D)}{pn(D)}$
- Smallest total number of occupied vertices such that the number of agents is minimum $= mfvs_{pn}(D)$
- $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}$

Theorem

The problems of determining $pn_{mfvs}(D)$, $mfvs_{pn}(D)$, μ, and λ are NP-Complete and not APX.

agents for minimizing # occupied vertices

$\exists$ digraphs with arbitrary large ratio: $\mu = \frac{pn_{mfvs}(D)}{pn(D)}$.



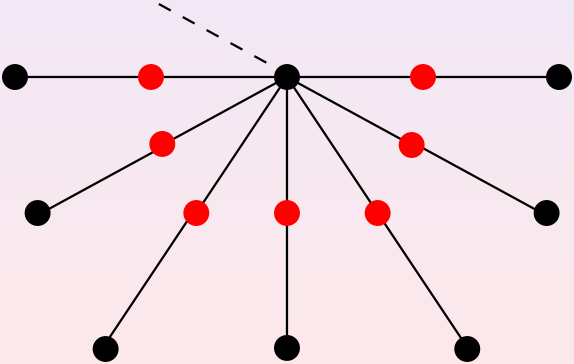
$mfvs(D) = n$

$pn(D) = 2$

$pn_{mfvs}(D) = n$

agents for minimizing # occupied vertices

$\exists$ digraphs with arbitrary large ratio: $\mu = \frac{pn_{mfvs}(D)}{pn(D)}$.



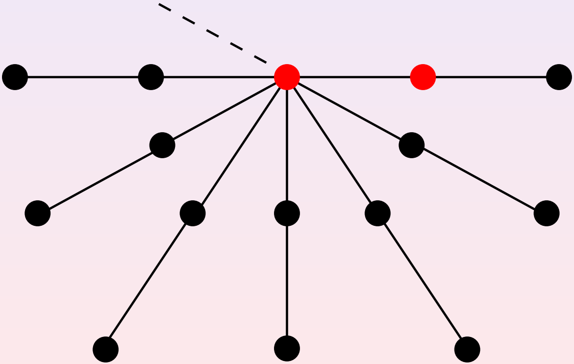
$mfvs(D) = n$

$pn(D) = 2$

$pn_{mfvs}(D) = n$

agents for minimizing # occupied vertices

$\exists$ digraphs with arbitrary large ratio: $\mu = \frac{pn_{mfvs}(D)}{pn(D)}$.



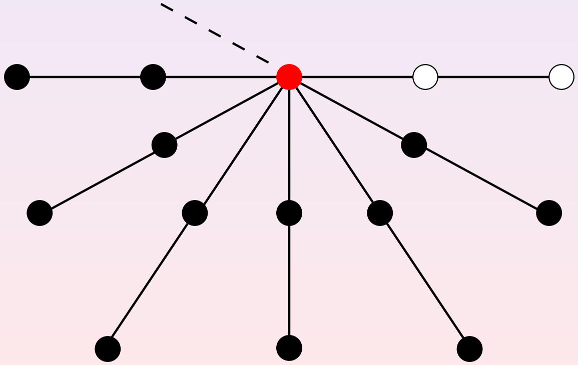
$mfvs(D) = n$

$pn(D) = 2$

$pn_{mfvs}(D) = n$

agents for minimizing # occupied vertices

$\exists$ digraphs with arbitrary large ratio: $\mu = \frac{pn_{mfvs}(D)}{pn(D)}$.



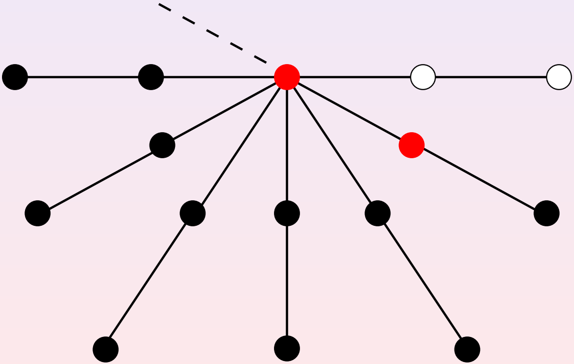
$mfvs(D) = n$

$pn(D) = 2$

$pn_{mfvs}(D) = n$

agents for minimizing # occupied vertices

$\exists$ digraphs with arbitrary large ratio: $\mu = \frac{pn_{mfvs}(D)}{pn(D)}$.



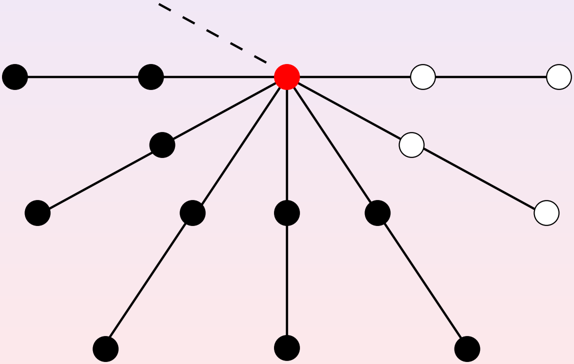
$mfvs(D) = n$

$pn(D) = 2$

$pn_{mfvs}(D) = n$

agents for minimizing # occupied vertices

$\exists$ digraphs with arbitrary large ratio: $\mu = \frac{pn_{mfvs}(D)}{pn(D)}$.



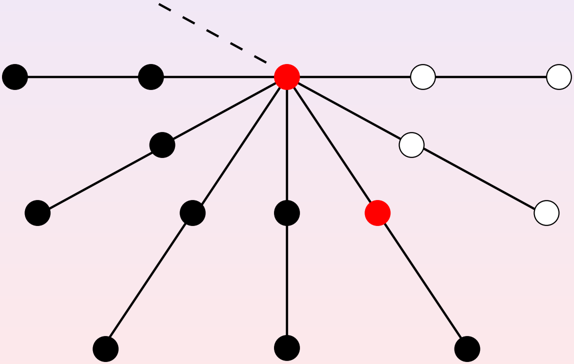
$mfvs(D) = n$

$pn(D) = 2$

$pn_{mfvs}(D) = n$

agents for minimizing # occupied vertices

$\exists$ digraphs with arbitrary large ratio: $\mu = \frac{pn_{mfvs}(D)}{pn(D)}$.



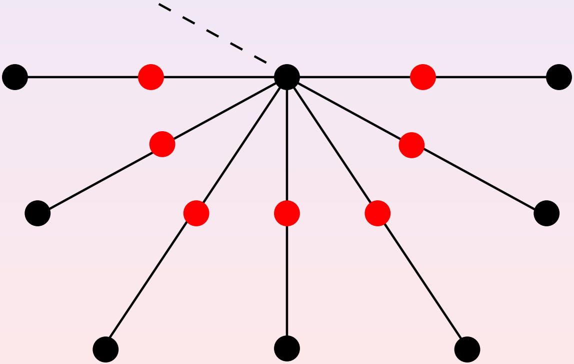
$mfvs(D) = n$

$pn(D) = 2$

$pn_{mfvs}(D) = n$

agents for minimizing # occupied vertices

$\exists$ digraphs with arbitrary large ratio: $\mu = \frac{pn_{mfvs}(D)}{pn(D)}$.



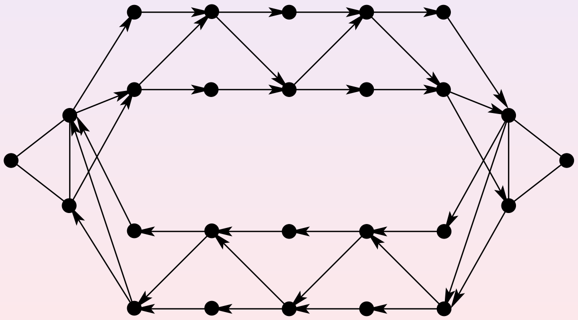
$$mfvs(D) = n$$

$$pn(D) = 2$$

$$pn_{mfvs}(D) = n$$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}$.



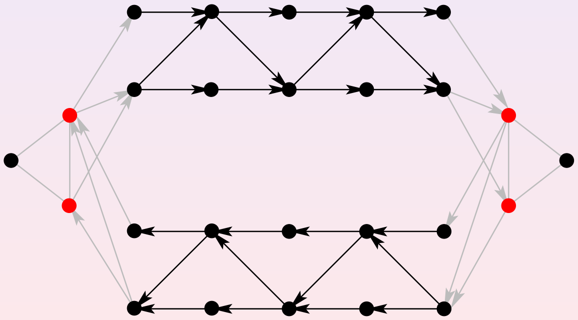
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}$.



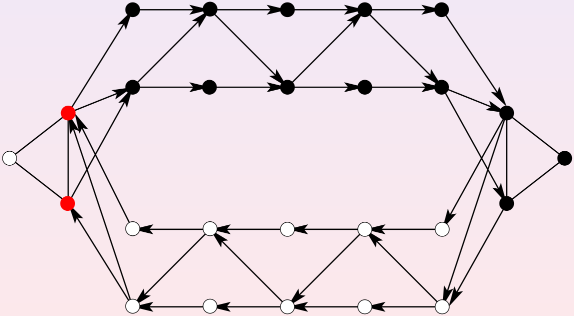
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}$.



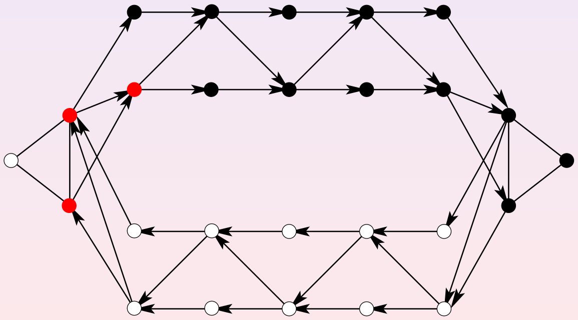
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}$.



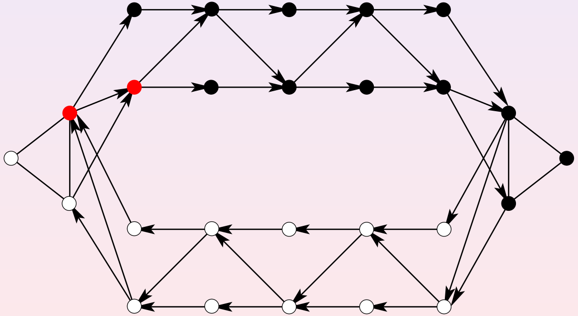
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}$.



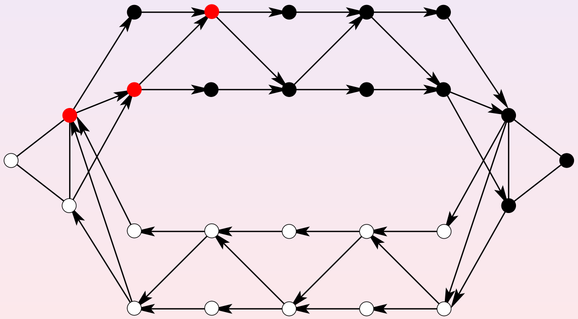
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}$.



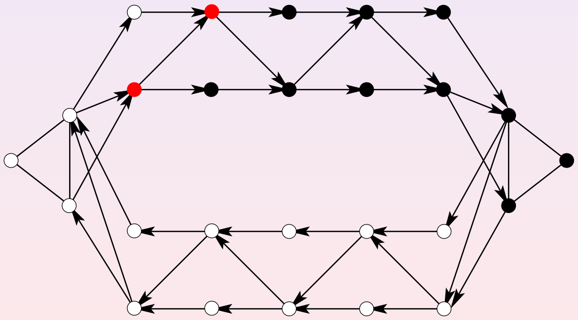
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}$.



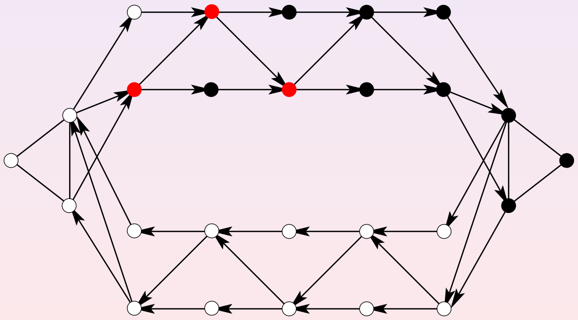
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}$.



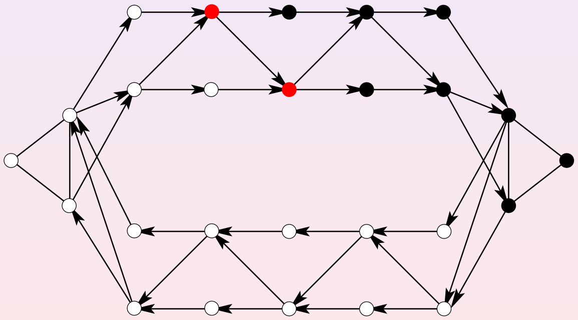
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}$.



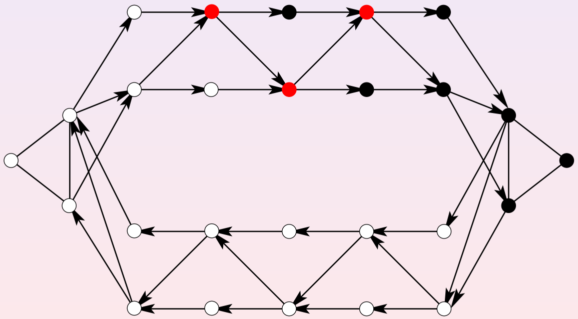
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

$\#$ occupied vertices by the minimum $\#$ agents

$\exists$ digraphs with arbitrary large ratio: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}$.



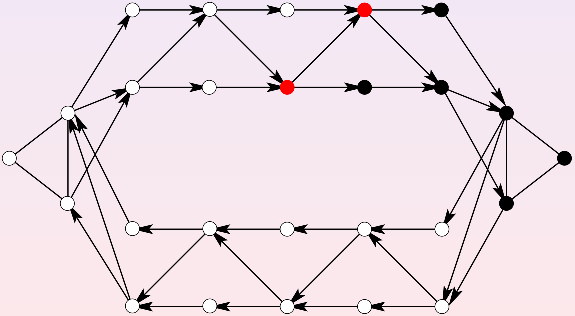
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}$.



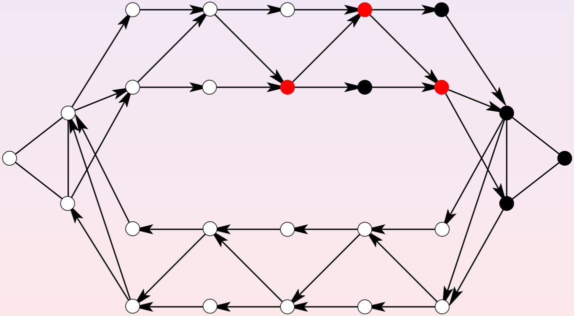
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}$.



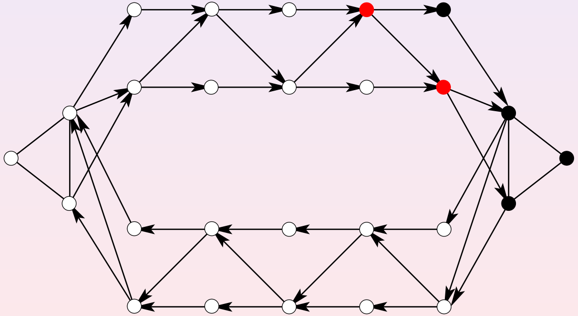
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

$\#$ occupied vertices by the minimum $\#$ agents

$\exists$ digraphs with arbitrary large ratio: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}$.



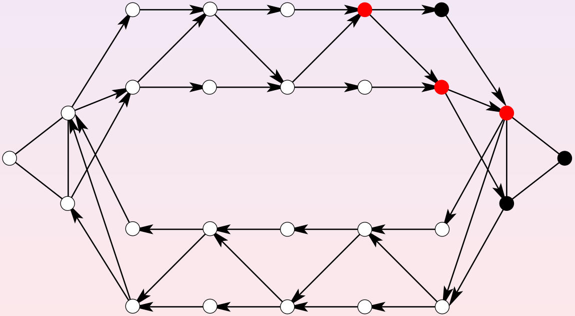
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}$.



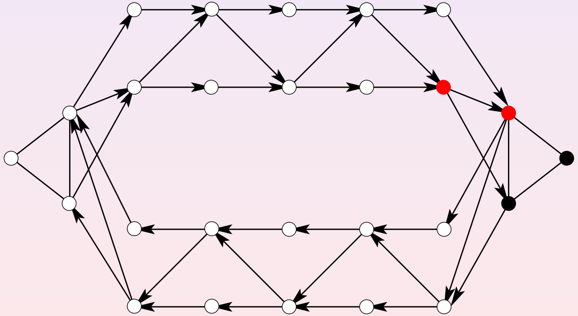
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}$.



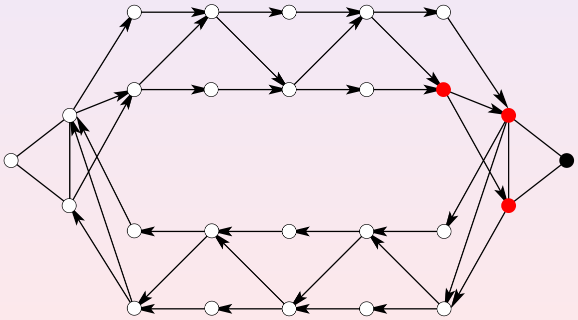
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}$.



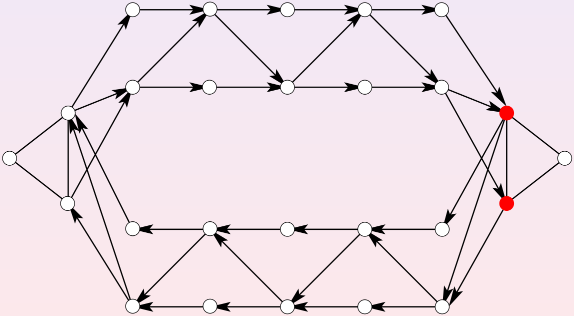
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}$.



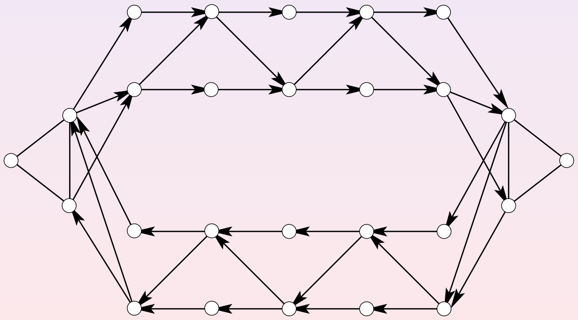
$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

occupied vertices by the minimum # agents

$\exists$ digraphs with arbitrary large ratio: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}$.



$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$

$\#$ occupied vertices by the minimum $\#$ agents

Directed graphs with BOUNDED Process Number:
$\lambda =$ occupied vertices / mfvs UNBOUNDED

What if G is undirected ??

Let G be a symmetric directed/undirected graph,
$\lambda = \frac{mfvs_{pn}(G)}{mfvs(G)} \leq pn(G)$

$\#$ occupied vertices by the minimum $\#$ agents

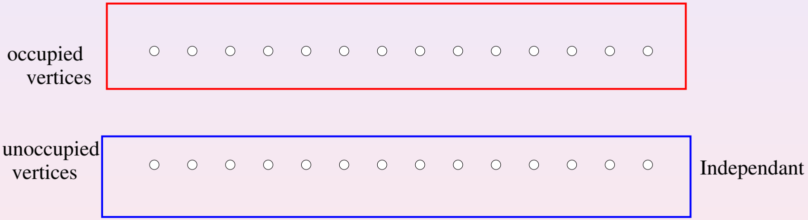
Directed graphs with BOUNDED Process Number:
$\lambda =$ occupied vertices / mfvs UNBOUNDED

What if G is undirected ??

Let G be a symmetric directed/undirected graph,
$\lambda = \frac{mfvs_{pn}(G)}{mfvs(G)} \leq pn(G)$

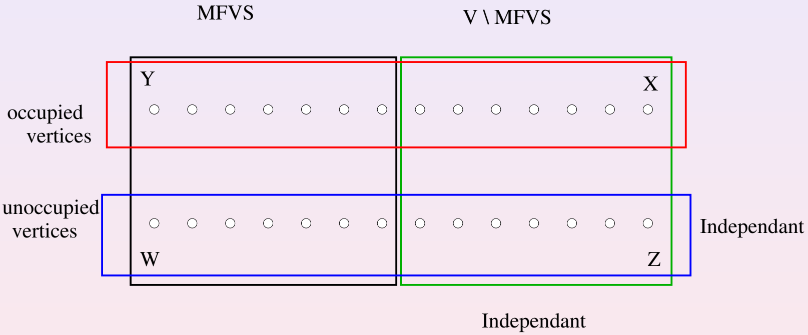
occupied vertices by the minimum # agents

Consider a MFVS of G. S using $pn(G)$ agents and occupying $mfvs_{pn}(G)$ vertices, such that occupies the minimum number of vertices in MFVS



occupied vertices by the minimum # agents

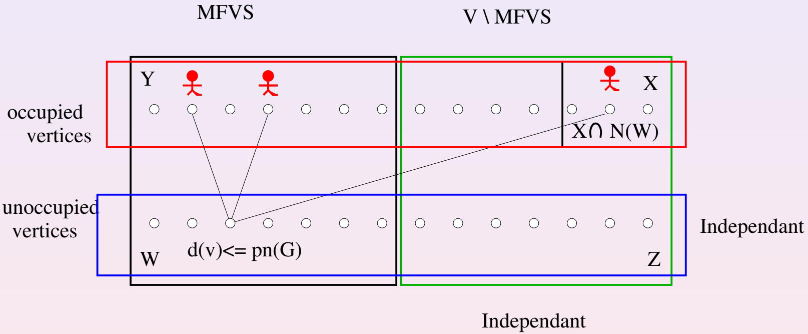
Consider a MFVS of G. S using $pn(G)$ agents and occupying $mfvs_{pn}(G)$ vertices, such that occupies the minimum number of vertices in MFVS



$$\lambda = \frac{mfvs_{pn}(G)}{mfvs(G)} = \frac{Y+X}{Y+W}$$

occupied vertices by the minimum # agents

Consider a MFVS of G. S using $pn(G)$ agents and occupying $mfvs_{pn}(G)$ vertices, such that occupies the minimum number of vertices in MFVS

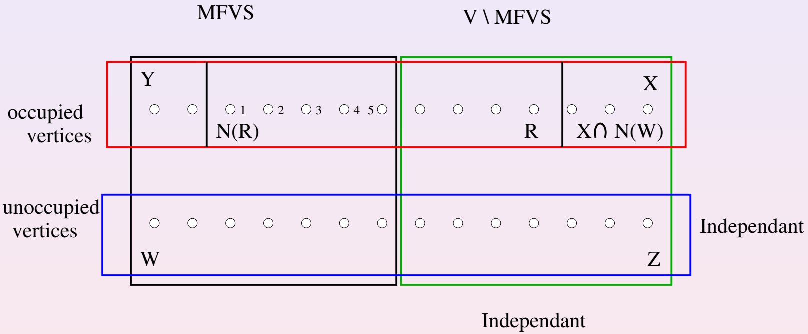


$$\lambda = \frac{mfvs_{pn}(G)}{mfvs(G)} = \frac{|Y|+|X|}{|Y|+|W|}$$

$$|X| = |X \cap N(W)| + |R| \leq |W|.pn(G) + |R|$$

occupied vertices by the minimum # agents

Consider a MFVS of G. S using $pn(G)$ agents and occupying $mfvs_{pn}(G)$ vertices, such that occupies the minimum number of vertices in MFVS

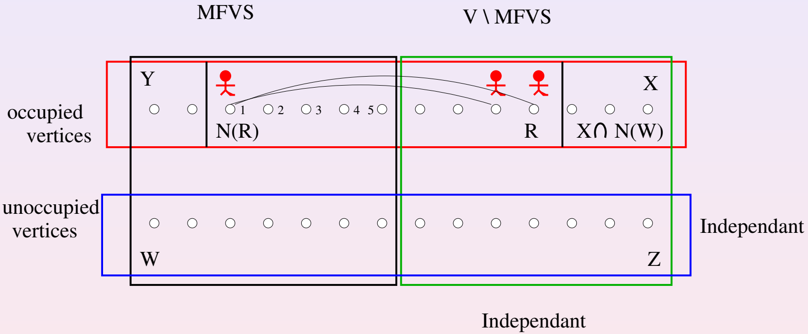


$$\lambda = \frac{mfvs_{pn}(G)}{mfvs(G)} = \frac{|Y|+|X|}{|Y|+|W|} \leq \frac{|Y|+|W|.pn(G)+|R|}{|Y|+|W|}$$

$N(R) = \{v_1, \cdots, v_r\} \subseteq Y$: ordering in which agents are removed

occupied vertices by the minimum # agents

Consider a MFVS of G. S using $pn(G)$ agents and occupying $mfvs_{pn}(G)$ vertices, such that occupies the minimum number of vertices in MFVS

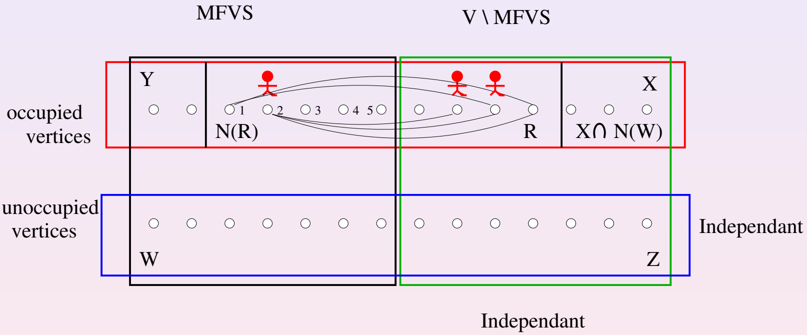


$$\lambda = \frac{mfvs_{pn}(G)}{mfvs(G)} = \frac{|Y|+|X|}{|Y|+|W|} \leq \frac{|Y|+|W|.pn(G)+|R|}{|Y|+|W|}$$

$|N(v_1)| \leq pn(G) - 1$

occupied vertices by the minimum # agents

Consider a MFVS of G. S using $pn(G)$ agents and occupying $mfvs_{pn}(G)$ vertices, such that occupies the minimum number of vertices in MFVS

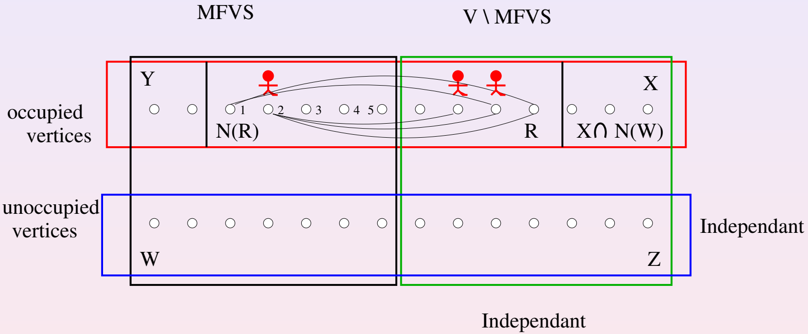


$$\lambda = \frac{mfvs_{pn}(G)}{mfvs(G)} = \frac{|Y|+|X|}{|Y|+|W|} \leq \frac{|Y|+|W|.pn(G)+|R|}{|Y|+|W|}$$

$|N(v_2) \setminus N(v_1)| \leq pn(G) - 1$, $|N(v_i) \setminus \bigcup_{j<i} N(v_j)| \leq pn(G) - 1$

occupied vertices by the minimum # agents

Consider a MFVS of G. S using $pn(G)$ agents and occupying $mfvs_{pn}(G)$ vertices, such that occupies the minimum number of vertices in MFVS

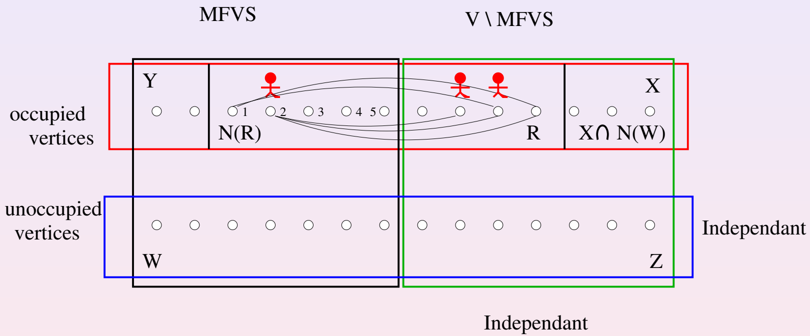


$$\lambda = \frac{mfvs_{pn}(G)}{mfvs(G)} = \frac{|Y|+|X|}{|Y|+|W|} \leq \frac{|Y|+|W|.pn(G)+|R|}{|Y|+|W|}$$

so $|R| \leq |N(R)|(pn(G)-1) \leq |Y|(pn(G)-1)$

occupied vertices by the minimum # agents

Consider a MFVS of G. S using $pn(G)$ agents and occupying $mfvs_{pn}(G)$ vertices, such that occupies the minimum number of vertices in MFVS

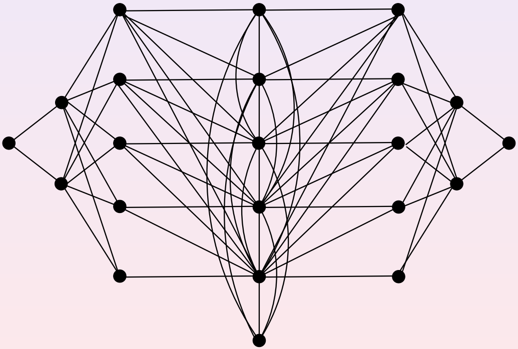


$$\lambda = \frac{mfvs_{pn}(G)}{mfvs(G)} = \frac{|Y|+|X|}{|Y|+|W|} \leq \frac{|Y|+|W|.pn(G)+|R|}{|Y|+|W|}$$

$$\lambda \leq \frac{|Y|+|W|.pn(G)+|Y|(pn(G)-1)}{|Y|+|W|} = pn(G)$$

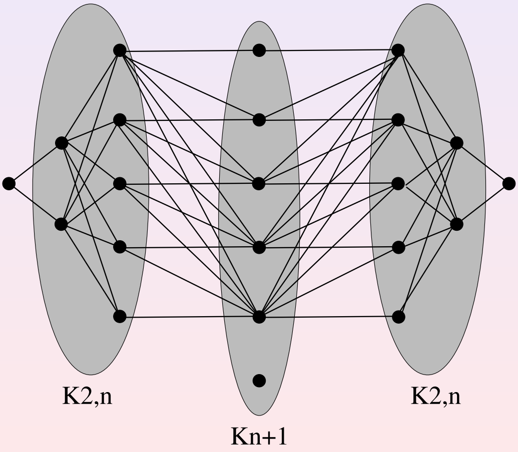
occupied vertices by the minimum # agents

$\forall \epsilon$, $\exists$ symmetric digraphs D: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)} > 3 - \epsilon.$



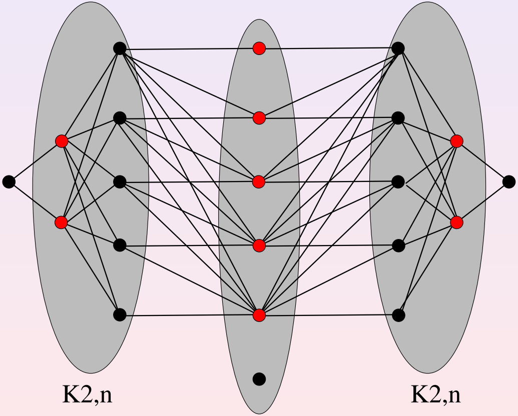
occupied vertices by the minimum # agents

$\forall \epsilon$, $\exists$ symmetric digraphs D: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)} > 3 - \epsilon$.



occupied vertices by the minimum # agents

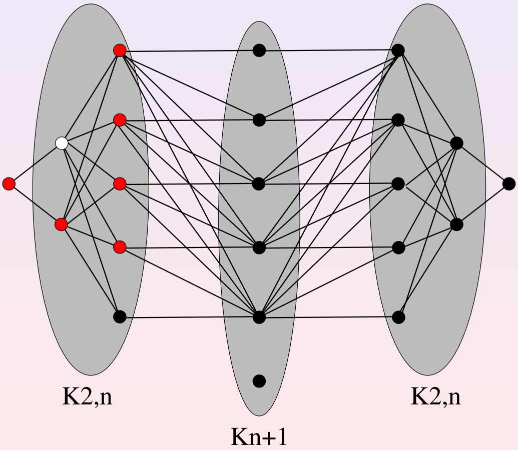
$\forall \epsilon$, $\exists$ symmetric digraphs D: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)} > 3 - \epsilon$.



$mfvs(D) = n + 4$

$\#$ occupied vertices by the minimum $\#$ agents

$\forall \epsilon$, $\exists$ symmetric digraphs D: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)} > 3 - \epsilon$.

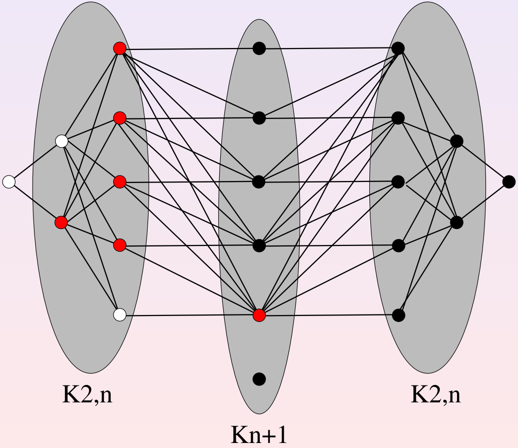


$mfvs(D) = n + 4$

$pn(D) = n + 1$

occupied vertices by the minimum # agents

$\forall \epsilon$, $\exists$ symmetric digraphs D: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)} > 3 - \epsilon$.

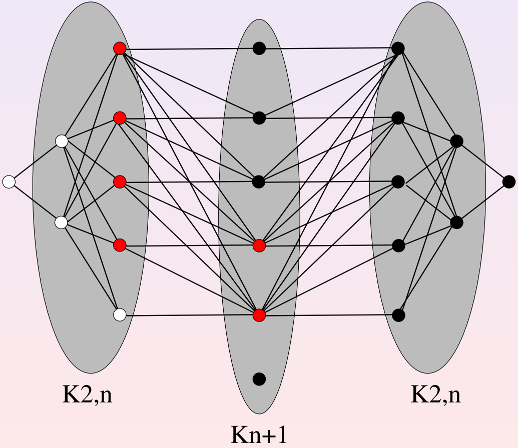


$mfvs(D) = n + 4$

$pn(D) = n + 1$

occupied vertices by the minimum # agents

$\forall \epsilon$, $\exists$ symmetric digraphs D: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)} > 3 - \epsilon$.

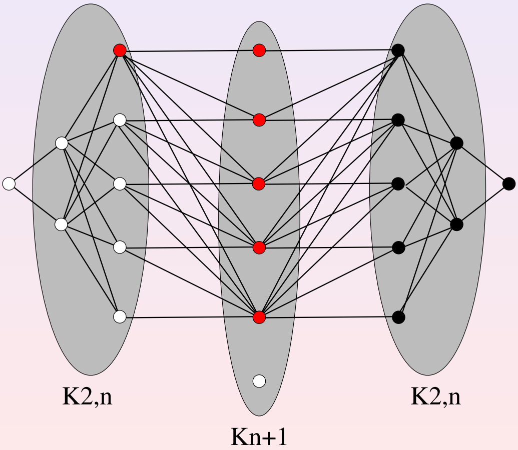


$mfvs(D) = n + 4$

$pn(D) = n + 1$

$\#$ occupied vertices by the minimum $\#$ agents

$\forall \epsilon$, $\exists$ symmetric digraphs D: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)} > 3 - \epsilon$.

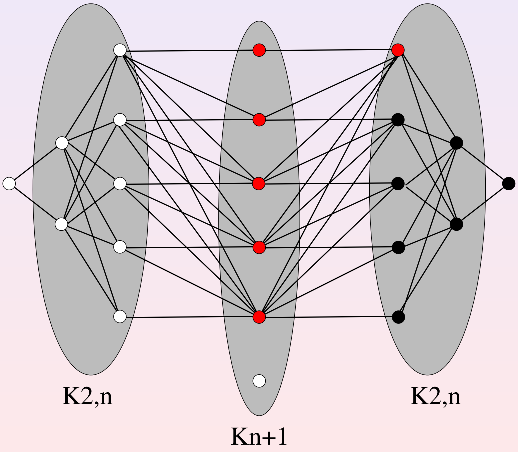


$mfvs(D) = n + 4$

$pn(D) = n + 1$

occupied vertices by the minimum # agents

$\forall \epsilon, \exists$ symmetric digraphs D: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)} > 3 - \epsilon$.

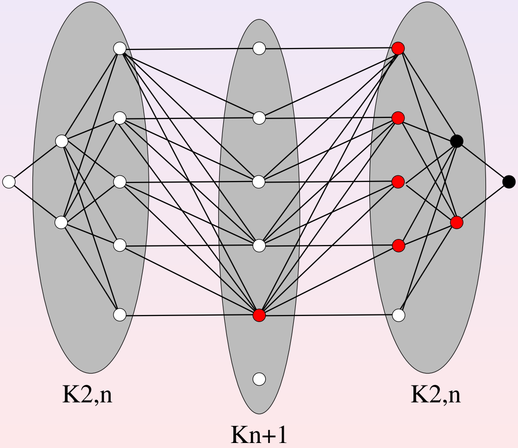


$mfvs(D) = n + 4$

$pn(D) = n + 1$

occupied vertices by the minimum # agents

$\forall \epsilon$, $\exists$ symmetric digraphs D: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)} > 3 - \epsilon$.

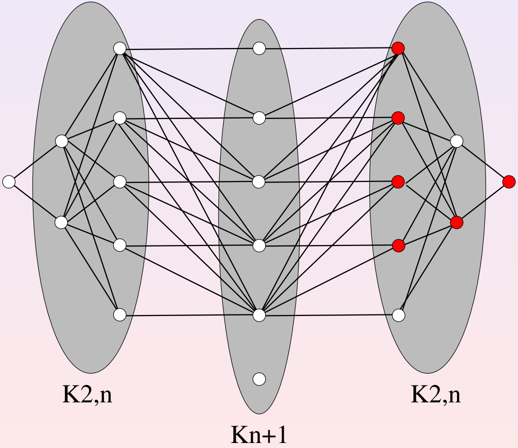


$mfvs(D) = n + 4$

$pn(D) = n + 1$

$\#$ occupied vertices by the minimum $\#$ agents

$\forall \epsilon$, $\exists$ symmetric digraphs D: $\lambda = \dfrac{mfvs_{pn}(D)}{mfvs(D)} > 3 - \epsilon$.



$mfvs(D) = n + 4$

$pn(D) = n + 1$

$mfvs_{pn}(D) = 3n + 2$

Some open questions on Tradeoff

A lot of "bad" news... No tradeoff ?

Conjecture

Let G be a symmetric directed/undirected graph,
$\lambda = \frac{mfvs_{pn}(G)}{mfvs(G)} \leq 3$

i.e.,
even using "few" searchers, can we occupy "few" nodes?

What about computation?

In theory: everything is NP-complete :(
What if we want to compute anyway?

Few approximation algorithms (as far as I know):

- treewidth : $O(\sqrt{\log tw})$ **[Feige et al. 2005]**
- treewidth of planar: $O(1)$ **[Seymour & Thomas 94]**
- heuristics for treewidth **[Bodlaender, Koster et al.]**

Nothing for pathwidth !?

Heuristic and simulations **[Coudert, Huc, Mazauric, N., Sereni'09]**

to compute upper bounds on process number

heuristic using LP **(Solano [JOCN 09])**

What about computation?

In theory: everything is NP-complete :(
What if we want to compute anyway?

Few approximation algorithms (as far as I know):

- treewidth : $O(\sqrt{\log tw})$ **[Feige et al. 2005]**
- treewidth of planar: $O(1)$ **[Seymour & Thomas 94]**
- heuristics for treewidth **[Bodlaender, Koster et al.]**

Nothing for pathwidth !?

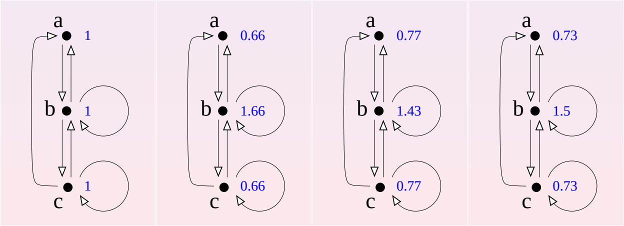
Heuristic and simulations **[Coudert, Huc, Mazauric, N., Sereni'09]**

to compute upper bounds on process number

heuristic using LP **(Solano [JOCN 09])**

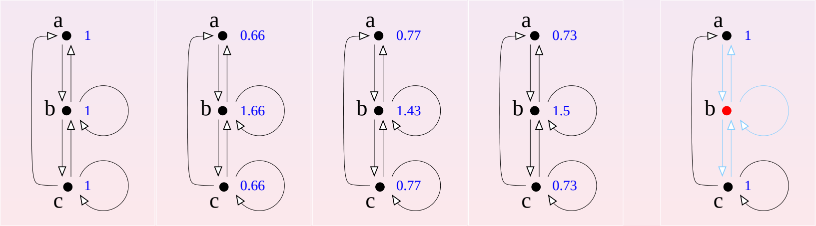
Heuristic / process number

1. Process nodes with all out-neighbors occupied or processed
2. If one non-occupied and non-processed out-neighbor, "slide" the agent
3. Choose of a candidate vertex to receive an agent (to be removed) using a *flow circulation method*
4. Remove that vertex and process all possible vertices including removed vertices and priority connections
5. Repeat 1-4 until processing of all vertices



Heuristic / process number

1. Process nodes with all out-neighbors occupied or processed
2. If one non-occupied and non-processed out-neighbor, "slide" the agent
3. Choose of a candidate vertex to receive an agent (to be removed) using a *flow circulation method*
4. Remove that vertex and process all possible vertices including removed vertices and priority connections
5. Repeat 1-4 until processing of all vertices

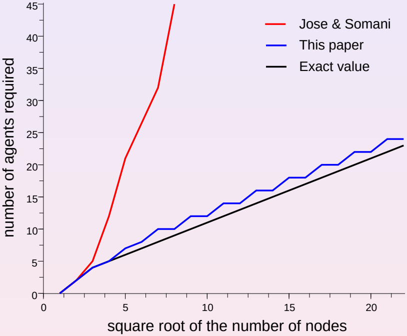


Heuristic / process number

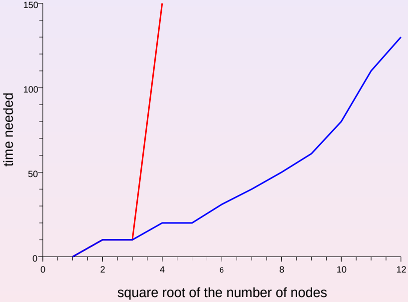
1. Process nodes with all out-neighbors occupied or processed
2. If one non-occupied and non-processed out-neighbor, "slide" the agent
3. Choose of a candidate vertex to receive an agent (to be removed) using a *flow circulation method*
4. Remove that vertex and process all possible vertices including removed vertices and priority connections
5. Repeat 1-4 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

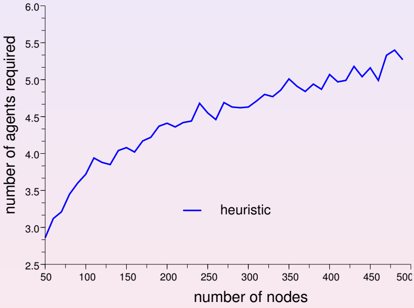


Number of simultaneous agents (break-before-make)

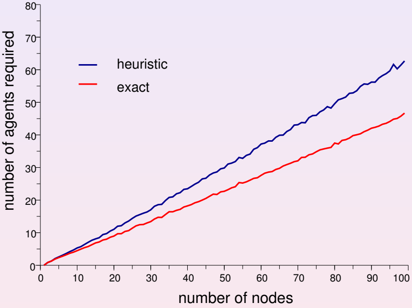


Computation time

Simulation results



2-digraphs



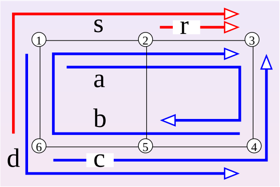
Circular arc graphs

Outline

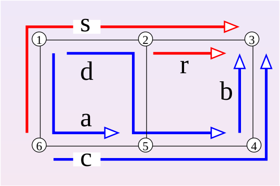
1. "Practical" motivations
2. Processing and Graph Searching Games
3. "New" problems
   - Tradeoff: # agents vs. # occupied vertices
   - Computation: approximation and heuristic
4. Variants and Open questions

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

Three questions to remember (and solve?)

Is the process-number a "good" directed width?

Visible fugitive: decomposition/bramble/ cost of monotonicity?

Can we efficiently compute pathwidth?

Approximation, heuristics?

Can graph searching help to study other problems?

related to scheduling

Thank you