Graph Searching and Routing Reconfiguration in WDM Networks

Nicolas Nisse

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


GRASTA, Banff, Canada, October 11th, 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 \( (u, v) \) \iff 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
⇒ 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
⇒ Online processes to route all requests: e.g., greedy routing

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

Request d : 1 → 3
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 → 3
Request e : 6 → 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)

```
1 2 3
4 5 6
7 8 9
```

Request d : 1 → 3
Request e : 6 → 5
Request c : 2 → 3
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 → 3
Request e : 6 → 5
Request c : 2 → 3
Failure of link {8, 9}
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 → 3
Request e : 6 → 5
Request c : 2 → 3
Rerouting of request e
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 → 3
Request e : 6 → 5
Request c : 2 → 3
Request b : 1 → 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 resources

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 resources

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

⇒ Destination resources must be available

Break-before-make:

Break connection before establishing the new path

⇒ 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
Objective: Number of Interruptions (detailed later)
Motivations  Processing game  Problems  Variants

Dependency digraph

$u \rightarrow v$
if $u$ needs resources of $v$

if $v$ must be rerouted/interrupted before $u$

$b$ needs resources used by $d$ and $c$
Dependancy Digraph

- one vertex per connection with different routes in $\mathcal{I}$ and $\mathcal{F}$
- arc from $u$ to $v$ if resources needed by $u$ in $\mathcal{F}$ are used by $v$ in $\mathcal{I}$
A game on dependency digraph

Routing Reconfiguration & Processing Game

Cyclic dependencies
⇒ 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 $a$

eroute request $a$
A game on dependency digraph

process node \( d \) and remove agent route request \( d \)
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

\[ |V| + 2 \]

\[ 2 |E| \]
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 \textit{simultaneous} interrupted requests

\textbf{Process Number}, $pn = \text{smallest number of requests that have to be \textit{simultaneously} interrupted.}$

Here, $pn = 1 \Rightarrow \text{Gap with MFVS up to } N/2$
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, $pn = 1 \implies$ 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 \textit{simultaneous} interrupted requests

Process Number, $pn =$ smallest number of requests that have to be \textit{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 \textit{simultaneous} interrupted requests

\textbf{Process Number}, $pn =$ smallest number of requests that have to be \textit{simultaneously} interrupted.
Here, $pn = 1 \implies$ 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 \textbf{simultaneous} interrupted requests

\textbf{Process Number}, $pn =$ smallest number of requests that have to be \textit{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 = \text{min. number of simultaneous interruptions} \\
$pn(D) = \text{min number of agents used during the strategy}$

Min. Feedback Vertex Set = \text{min. total number of interruptions} \\
$mfvs(D) = \text{min total number of occupied vertices during the strategy}$
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
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

![DAG Diagram]

Theorem

\[ pn(D) = 0 \text{ iff } D \text{ is a DAG} \]
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

\[ pn(D) = 1 \iff \forall SCC, \text{ MFVS}(SCC) = 1 \]

\[ O(N + M) \]
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

\[ pn(D) = 1 \iff \forall SCC, \text{MFVS}(SCC) = 1 \]

\[ O(N + M) \]
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

\[ pn(D) = 1 \iff \forall \text{SCC}, \text{MFVS}(\text{SCC}) = 1 \]

\[ O(N + M) \]
Processing Game vs. Graph Searching

In undirected graphs or “symmetric” digraphs

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

Invisible fugitive moves along edges

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

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. 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) \( \iff \) 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 $\iff$ 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**  ⇔  **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 **backwards** 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** ↔ **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

<table>
<thead>
<tr>
<th>Related parameter of directed (and undirected) graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>vertex separation vs = (directed) pathwidth</td>
</tr>
<tr>
<td>Kinnersley [IPL 92]</td>
</tr>
</tbody>
</table>

### Theorem

(Coudert & Sereni, 2007)

\[
\begin{align*}
\text{vs}(D) & \leq \text{monotone } \text{pn}(D) \leq \text{vs}(D) + 1 \\
pw(G) & \leq \text{monotone-}pn(G) \leq pw(G) + 1 & \text{undirected graph } G \\
dpw(D) & \leq \text{monotone-}pn(D) \leq pw(D) + 1 & \text{directed graph } D
\end{align*}
\]

### 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 $D$, $pn(D) = \text{monotone-}pn(D)$

Process-decomposition

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

$pn(D) = \min. \text{ width among all decompositions}$

$\Rightarrow pn(D) = pn(\bar{D})$ \hspace{1cm} ($\bar{D}$ is $D$ where arcs have been reversed)
To summarize

<table>
<thead>
<tr>
<th></th>
<th>undirected graphs</th>
<th>directed graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fugitive</strong></td>
<td>move along edges</td>
<td>move along arcs</td>
</tr>
<tr>
<td></td>
<td>must move</td>
<td>must be able to move</td>
</tr>
<tr>
<td></td>
<td></td>
<td>must move in SCC</td>
</tr>
<tr>
<td><strong>invisible</strong></td>
<td>pw</td>
<td>dpw</td>
</tr>
<tr>
<td></td>
<td>pn</td>
<td>pn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>?</td>
</tr>
<tr>
<td><strong>visible</strong></td>
<td>tw</td>
<td>DAG-width</td>
</tr>
<tr>
<td></td>
<td>monotone</td>
<td>visible-pn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>≈ dtw</td>
</tr>
<tr>
<td></td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td>no ratio</td>
<td>dtw ≤ 3br</td>
</tr>
</tbody>
</table>

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** = \( \text{min. number of simultaneous interruptions} \)

\[
pn(D) = \text{min number of agents used during the strategy}
\]

**Min. Feedback Vertex Set** = \( \text{min. total number of interruptions} \)

\[
mfvs(D) = \text{min total number of occupied vertices during the strategy}
\]
Tradeoff: total/ max simultaneous interruptions

#occupied vertices

- $mfvs_{\{pn\}}$
- $mfvs$
- $pn$
- $pn_{\{mfvs\}}$
- $mfvs$

#agents
**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 $pn_{mfvs}(D)$, $mfvs_{pn}(D)$, $\mu$, and $\lambda$ are NP-Complete and not APX.*
∃ digraphs with arbitrary large ratio: $\mu = \frac{pn_{mfvs}(D)}{pn(D)}$.

$mfvs(D) = n$

$pn(D) = 2$

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

$mfvs(D) = n$

$pn(D) = 2$

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

$mfvs(D) = n$

$pn(D) = 2$

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

\( mfvs(D) = n \)

\( pn(D) = 2 \)

\( pn_{mfvs}(D) = n \)
∃ digraphs with arbitrary large ratio: \( \mu = \frac{p_{n_{mfvs}}(D)}{p_n(D)} \).

\[ mfvs(D) = n \]
\[ p_n(D) = 2 \]
\[ p_{n_{mfvs}}(D) = n \]
∃ digraphs with arbitrary large ratio: $\mu = \frac{pn_{mfvs}(D)}{pn(D)}$.

$mfvs(D) = n$

$pn(D) = 2$

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

$mfvs(D) = n$

$pn(D) = 2$

$pn_{mfvs}(D) = n$
∃ digraphs with arbitrary large ratio: \( \lambda = \frac{mfv_{spn}(D)}{mfv(D)}. \)

\( mfv(D) = 4 \)

\( pn(D) = 3 \)

\( mfv_{spn}(D) = n + 4 \)
∃ digraphs with arbitrary large ratio: \( \lambda = \frac{mfvs_{pn}(D)}{mfvs(D)} \).

\( mfvs(D) = 4 \)

\( pn(D) = 3 \)

\( mfvs_{pn}(D) = n + 4 \)
∃ digraphs with arbitrary large ratio: \[ \lambda = \frac{mf_{vs_{pn}}(D)}{mf_{vs}(D)}. \]

\[ mf_{vs}(D) = 4 \]
\[ pn(D) = 3 \]
\[ mf_{vs_{pn}}(D) = n + 4 \]
∃ digraphs with arbitrary large ratio: \( \lambda = \frac{mfvs_{pn}(D)}{mfvs(D)} \).

\[ mfvs(D) = 4 \]
\[ pn(D) = 3 \]
\[ mfvs_{pn}(D) = n + 4 \]
∃ digraphs with arbitrary large ratio: \( \lambda = \frac{mfvs_{pn}(D)}{mfvs(D)} \).

\( mfvs(D) = 4 \)

\( pn(D) = 3 \)

\( mfvs_{pn}(D) = n + 4 \)
∃ digraphs with arbitrary large ratio: \( \lambda = \frac{mfv_{pn}(D)}{mfv(D)} \).

\[
mfv(D) = 4 \\
 pn(D) = 3 \\
mfv_{pn}(D) = n + 4
\]
∃ digraphs with arbitrary large ratio: \( \lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}. \)

\[
mfvs(D) = 4 \\
pn(D) = 3 \\
mfvs_{pn}(D) = n + 4
\]
∃ digraphs with arbitrary large ratio: \( \lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}. \)

\[ mfvs(D) = 4 \]

\[ pn(D) = 3 \]

\[ mfvs_{pn}(D) = n + 4 \]
∃ digraphs with arbitrary large ratio: \( \lambda = \frac{mfvs_{pn}(D)}{mfvs(D)} \).

\[ \begin{align*}
mfvs(D) &= 4 \\
fn(D) &= 3 \\
mfvs_{pn}(D) &= n + 4
\end{align*} \]
∃ digraphs with arbitrary large ratio: \( \lambda = \frac{mfv_{sp} (D)}{mfv (D)} \).

- \( mfv (D) = 4 \)
- \( pn (D) = 3 \)
- \( mfv_{sp} (D) = n + 4 \)
∃ digraphs with arbitrary large ratio: \[ \lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}. \]

mfvs(D) = 4

pn(D) = 3

mfvs_{pn}(D) = n + 4
∃ digraphs with arbitrary large ratio: \( \lambda = \frac{mfv_{spn}(D)}{mfv(D)} \).

- \( mfv(D) = 4 \)
- \( pn(D) = 3 \)
- \( mfv_{spn}(D) = n + 4 \)
∃ digraphs with arbitrary large ratio: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}$.

$mfvs(D) = 4$

$pn(D) = 3$

$mfvs_{pn}(D) = n + 4$
∃ digraphs with arbitrary large ratio: \( \lambda = \frac{mfvs_{pn}(D)}{mfvs(D)} \).

\[
\begin{align*}
mfvs(D) &= 4 \\
pn(D) &= 3 \\
mfvs_{pn}(D) &= n + 4
\end{align*}
\]
∃ digraphs with arbitrary large ratio: \( \lambda = \frac{mfvs_{pn}(D)}{mfvs(D)} \).

\[
\begin{align*}
\text{mfvs}(D) &= 4 \\
\text{pn}(D) &= 3 \\
\text{mfvs}_{pn}(D) &= n + 4
\end{align*}
\]
∃ digraphs with arbitrary large ratio: \( \lambda = \frac{mfvs_{pn}(D)}{mfvs(D)} \).

\( mfvs(D) = 4 \)

\( pn(D) = 3 \)

\( mfvs_{pn}(D) = n + 4 \)
∃ digraphs with arbitrary large ratio: \( \lambda = \frac{mfvs_{pn}(D)}{mfvs(D)}. \)

- \( mfvs(D) = 4 \)
- \( pn(D) = 3 \)
- \( mfvs_{pn}(D) = n + 4 \)
∃ digraphs with arbitrary large ratio: \( \lambda = \frac{mfvs_{pn}(D)}{mfvs(D)} \).

\[ mfvs(D) = 4 \]
\[ pn(D) = 3 \]
\[ mfvs_{pn}(D) = n + 4 \]
Directed graphs with BOUNDED Process Number:
\( \lambda = \frac{\text{occupied vertices}}{\text{mfvs}} \) UNBOUNDED

What if \( G \) is undirected ??

Let \( G \) be a symmetric directed/undirected graph,
\[
\lambda = \frac{\text{mfvs}_{pn}(G)}{\text{mfvs}(G)} \leq \text{pn}(G)
\]
Directed graphs with BOUNDED Process Number:
\[ \lambda = \text{occupied vertices} / \text{mfvs UNBOUNDED} \]

What if \( G \) is undirected ??

Let \( G \) be a symmetric directed/undirected graph,
\[ \lambda = \frac{\text{mfvs}_{pn}(G)}{\text{mfvs}(G)} \leq pn(G) \]
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.
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.

<table>
<thead>
<tr>
<th>MFVS</th>
<th>$V \setminus$ MFVS</th>
</tr>
</thead>
<tbody>
<tr>
<td>occupied vertices</td>
<td>$Y$</td>
</tr>
<tr>
<td>unoccupied vertices</td>
<td>$W$</td>
</tr>
</tbody>
</table>

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

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

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

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

$$|N(v_2) \setminus N(v_1)| \leq pn(G) - 1, \ |N(v_i) \setminus \bigcup_{j<i} N(v_j)| \leq pn(G) - 1$$
Consider a MFVS of G. Using \( pn(G) \) agents and occupying \( mfvs_{pn}(G) \) vertices, such that occupies the minimum number of vertices in MFVS.

\[
\lambda = \frac{mfvs_{pn}(G)}{mfvs(G)} = \frac{|Y| + |X|}{|Y| + |W|} \leq \frac{|Y| + |W| \cdot pn(G) + |R|}{|Y| + |W|}
\]

so \( |R| \leq |N(R)|(pn(G) - 1) \leq |Y|(pn(G) - 1) \)
Consider a MFVS of $G$. $S$ using $pn(G)$ agents and occupying $mfv_{pn}(G)$ vertices, such that occupies the minimum number of vertices in MFVS

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

$$\lambda \leq \frac{|Y|+|W|.pn(G)+|Y|(pn(G)-1)}{|Y|+|W|} = pn(G)$$
\[
\forall \epsilon, \exists \text{ symmetric digraphs } D: \lambda = \frac{mfvs_{pn}(D)}{mfvs(D)} > 3 - \epsilon.
\]
\[ \forall \epsilon, \exists \text{ symmetric digraphs } D: \lambda = \frac{mfv_{sp}(D)}{mfv(D)} > 3 - \epsilon. \]
∀ε, ∃ symmetric digraphs $D$: $\lambda = \frac{mfv_{spn}(D)}{mfv(D)} > 3 - \epsilon$.

$mfv(D) = n + 4$
∀ε, ∃ symmetric digraphs $D$: $\lambda = \frac{mfvs_{pn}(D)}{mfvs(D)} > 3 - \epsilon$.

$mfvs(D) = n + 4$

$pn(D) = n + 1$
\[ \forall \epsilon, \exists \text{symmetric digraphs } D: \lambda = \frac{mfv_{pn}(D)}{mfv(D)} > 3 - \epsilon. \]

mfv(D) = n + 4

pn(D) = n + 1
∀ \epsilon, \exists \text{ symmetric digraphs } D: \lambda = \frac{mfv_{spn}(D)}{mfv(D)} > 3 - \epsilon.

mfv(D) = n + 4

pn(D) = n + 1
∀\(\epsilon\), \(\exists\) symmetric digraphs \(D\): \(\lambda = \frac{mfv_{spn}(D)}{mfv(D)} > 3 - \epsilon\).

\[mfv(D) = n + 4\]
\[pn(D) = n + 1\]
∀ε, ∃ symmetric digraphs D: \( \lambda = \frac{mfvs_{pn}(D)}{mfvs(D)} > 3 - \epsilon. \)

\( mfvs(D) = n + 4 \)

\( pn(D) = n + 1 \)
Motivations

Processing game

Problems

Variants

Tradeoff

Computation

∀ε, ∃ symmetric digraphs D: \( \lambda = \frac{mfv_{spn}(D)}{mfv(D)} > 3 - \epsilon \).
∀ε, ∃ symmetric digraphs D: \( \lambda = \frac{mfv_{spn}(D)}{mfvs(D)} > 3 - \epsilon \).

\[mfvs(D) = n + 4\]
\[pn(D) = n + 1\]
\[mfv_{spn}(D) = 3n + 2\]
Some open questions on Tradeoff

A lot of “bad” news… No tradeoff?

Conjecture

Let $G$ be a symmetric directed/undirected graph,

$$\lambda = \frac{mfvs_{sp}(G)}{mfvs(G)} \leq 3$$

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?

<table>
<thead>
<tr>
<th>Few approximation algorithms (as far as I know):</th>
</tr>
</thead>
<tbody>
<tr>
<td>• treewidth : $O(\sqrt{\log tw})$</td>
</tr>
<tr>
<td>• treewidth of planar: $O(1)$</td>
</tr>
<tr>
<td>• heuristics for treewidth</td>
</tr>
</tbody>
</table>

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])
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.
Motivations Processing game Problems Variants

Tradeoff Computation

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

- Jose & Somani
- This paper
- Exact value
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