Reconfiguration dans les réseaux optiques

D. Coudert¹, F. Huc¹,², D. Mazauric¹, N. Nisse¹ and J-S. Sereni³,⁴

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
Example: maintenance operation

- Symmetric links, capacity 1
- Maintenance on link 5-8
Example: maintenance operation

- Symmetric links, capacity 1
- Maintenance on link 5-8
Example: maintenance operation

- Symmetric links, capacity 1
- Maintenance on link 5-8
Context

Circuit-switched networks
- Telephone: call repacking (70’s)
- ATM
- WDM, MPLS

Motivation
- Optimize usage of resources (reduce blocking probability)
- Fault tolerance
- Maintenance operations
Related problems

Compute new "optimal" routing
- Minimizing number of changes
- NP-hard
- ILP, heuristics
- How to switch from current to new routing

Sequence of rerouting to converge to a valid routing
- Convergence time
- Existence

Compute new routing + sequence of rerouting
- Very hard problems
- Existence
Our problem

Inputs: Set of connection requests
+ current and new routing

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

Objective: Minimizing the number of simultaneous interrupted requests

Constraint: Reroute requests one by one
Dependency digraph

b before a

c and d before b
d before c
a, b, c before d
Dependency digraph

b before a
c and d before b
d before c
a, b, c before d
Dependency digraph

b before a

a, b, c before d

c and d before b

d before c

D. Mazaric et al.
Dependency digraph

\[ \text{b before a} \]
\[ \text{c and d before b} \]
\[ \text{d before c} \]
\[ \text{a, b, c before d} \]
Dependency digraph

D. Mazaric et al.

b before a
c and d before b

a, b, c before d
Dependency digraph

- b before a
- c and d before b
- d before c
- a, b, c before d
Process number, $pn$

Modeling in terms of cops-and-robber game
[Coudert & Pérennes & Pham & Sereni, AlgoTel, 2005]

- Fugitive in a graph: invisible, infinite speed
- Agents: with teleportation
- The goal is to capture the fugitive surely with the minimum number of agents
- Captured $=$ caught or surrounded
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

\[= \text{break/interrupt/route on temporary resources a connection}\]

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

\[= \text{(Re)route a connection when final resources are available}\]

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

Direct path

\[\text{DAG}\]

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

\[ \text{Th: If } D \text{ is a DAG, then } pn(D) = 0 \]
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

\[ \text{= break/interrupt/route on temporary resources a connection} \]

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

\[ \text{= (Re)route a connection when final resources are available} \]

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

Direct path, DAG

\text{Th: If } D \text{ 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
   $= \text{break/interrupt/route on temporary resources a connection}$

$R_2$ Process a vertex if all its out-neighbors are either processed or occupied by an agent
   $= \text{(Re)route a connection when final resources are available}$

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

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

\[ O(N + M) \]
Digraphs with process number 1

Rules

$R_1$ Put an agent on a vertex

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

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

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

$O(N + M)$
Digraphs with process number 1

Rules

$R_1$ Put an agent on a vertex
   $= \text{break/interrupt/route on temporary resources a connection}$

$R_2$ Process a vertex if all its out-neighbors are either processed or occupied by an agent
   $= (\text{Re})\text{route a connection when final resources are available}$

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

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

break request \(d\)

put an agent on node \(d\)
Example

reroute request c

process node c
Example

reroute request $b$

process node $b$
Example

reroute request $a$

process node $a$
Example

route request $d$

process node $d$
and remove agent
Process number: what is known

Related parameters

- Pathwidth, $pw$ [Robertson & Seymour, JCTB, 1983]
- Node search number, $ns$ [Kirousis & Papadimitriou, TCS, 1986]
- Vertex separation, $vs$ [Kinnersley, IPL, 1992]

Relations

- $pw(G) = vs(G) = ns(G) - 1$
- $vs(D) \leq pn(D) \leq vs(D) + 1$ [Coudert & Sereni, 2007]

Complexity

- NP-Complete
- Not APX
  - No polynomial time constant factor approximation algorithm

- Characterization of digraphs with process number 0, 1, 2
- Heuristic algorithms (MFVS) [Jose & Somani, DRCN, 2003]
Previous heuristic [Jose & Somani, DRCN, 2003]

1. Compute all directed cycles using Johnson’s algorithm
2. Choose $u^*$ that belongs to the maximum number of cycles
3. Remove $u^*$ 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 $c \Rightarrow$ only for small digraphs
Previous heuristic [Jose & Somani, DRCN, 2003]

1. Compute all directed cycles using Johnson’s algorithm
2. Choose $u^*$ that belongs to the maximum number of cycles
3. Remove $u^*$ 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 $c \Rightarrow$ only for small digraphs
Heuristic algorithms

Flow circulation method

Dependency Digraph

\[ t = 0 \]
Heuristic algorithms

Flow circulation method

Dependency Digraph

\[ t = 1 \]
Heuristic algorithms

Flow circulation method

Dependency Digraph

t = 2
Heuristic algorithms

Flow circulation method

Dependency Digraph

$t = k$
Heuristic algorithms

Given a strongly connected digraph $D$:

1. Apply flow circulation method
2. Let $u^*$ be one the nodes of maximum weight
3. Place an agent on $u^*$
4. Process all the nodes whose can be processed
5. Decompose remaining digraph into SCCs
6. Apply sequentially the heuristic on each SCC

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

2-digraphs

Circular arc graphs
Two classes of services

Priority connections
- Refuse *by contract* (SLA) any interruption

Impossibility
- Direct cycle of priority connections in the dependency digraph
  ⇒ *Small* number of such connections
- Partition into strongly connected components, $O(N + M)$

Transformation

⇒ Same problem to solve
Example with priority connection $d$

Routing 1

Routing 2

Dependency digraph, $pn = 1$

Without $d$, $pn = 2$
Conclusion and future works

**Conclusion:**
- Heuristic algorithms
- Two classes of services

**Future works:**
- Compromise simultaneous interruptions / duration of interruption
  - Multiple classes of services
  - Time dependent penalties
- Allow to reroute more than once
Merci