Routing Reconfiguration/Process Number: Coping wih Two Classes of Services

David Coudert, Florian Huc, Dorian Mazauric, Nicolas Nisse, Jean-Sébastien Sereni

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Routing Reconfiguration/Process Number: Coping wih Two Classes of Services

David Coudert — Florian Huc — Dorian Mazauric — Nicolas Nisse —

Jean-Sébastien Sereni

N° 6698

October 2008

Thème COM

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--6698--FR+ENG

Routing Reconfiguration/Process Number: Coping wih Two Classes of Services

David Coudert*†, Florian Huc*† , Dorian Mazauric*† , Nicolas Nisse*† , Jean-Sébastien Sereni‡§¶

Thème COM — Systèmes communicants
Projets Mascotte

Rapport de recherche n° 6698 — October 2008 — 15 pages

Abstract:

In WDM backbone networks, the traffic pattern evolves constantly due to the nature of the demand itself or because of equipment failures leading to reroute affected connections. In this context, requests are routed greedily using available resources without changing the routing of pre-established connections. However, such a policy leads to a poor usage of resources and so higher blocking probability: new connection requests might be rejected while network resources are sufficient to serve all the traffic. Therefore, it is important to regularly reconfigure the network by rerouting established connections in order to optimize the usage of network resources.

In this paper, we consider the network reconfiguration problem that consists in switching existing connections one after the other from the current routing to a new pre-computed routing. Due to cyclic dependencies between connections, some requests may have to be temporarily interrupted during this process. Clearly, the number of requests simultaneously interrupted has to be minimized. Furthermore, it might be impossible for the network operator to interrupt some connections because of the contract signed with the corresponding clients. In this setting, the network reconfiguration problem consists in going from a routing to another one given that some priority connections cannot be interrupted.

The network reconfiguration problem without priority connections has previously been modeled as a cops-and-robber game in [5, 6]. Here, we first extend this model to handle priority connections. Then we identify cases where no solution exists. Using a simple

This work was partially funded by Région PACA and by European project IST FET Aeolus.

* MASCOTTE, INRIA, I3S, CNRS, Univ. Nice Sophia, Sophia Antipolis, France.
† firstname.lastname@sophia.inria.fr
‡ CNRS, LIAFA, Univ. Paris Diderot, Paris, France.
§ Department of Applied Mathematics (KAM), Charles Univ., Pragues, Czech Republic.
¶ sereni@kam.mff.cuni.cz

transformation, we prove that the reconfiguration problem with priority connections can be reduced to the problem without this constraint. Finally, we propose a new heuristic algorithm that improves upon previous proposals.

Key-words: Rerouting, process number, vertex separation, pathwidth.

Reconfiguration de routage/process number: prise en compte de deux classes de services

Résumé : Dans les réseaux de cœur WDM, le trafic évolue de manière continue. Cela est due aux variations des demandes mais aussi aux pannes d'équipements qui nécessitent le reroutage de certaines connections. Dans ce contexte, les connections sont routées en utilisant les ressources disponibles et sans modifier le routage des connections préalablement établies. Cependant, une telle politique peut amener à des blocages dus à une mauvaise utilisation des ressources. Ainsi, il est important de reconfigurer régulièrement le routage des connections présentes afin d'optimiser l'utilisation des ressources du réseau.

Dans cet article, nous étudions le problème de reconfiguration qui consiste à modifier des connections de manière successives pour passer du routage courant à un autre routage précalculé. Seulement, les redondances entre les ressources nécessaires à l'établissement de certaines connections et les ressources utilisées par d'autres peuvent nécessiter l'interruption temporaires de certaines connections. Il est dans l'intérêt de l'opérateur de minimiser le nombre de ressources à interrompre simultanément. De plus, dus à des contrats spécifiques avec des clients, il peut être impossible à l'opérateur d'en interrompre certaines. Dans ce contexte, le problème de reconfiguration consiste à ordonnancer les connections à rerouter tout en minimisant le nombre de connections interrompues et en respectant certaines contraintes imposées par des connections prioritaires.

Le problème de reroutage en l'absence de connections prioritaires a déjà été étudié et modélisé par un jeux de capture dans [5, 6]. Nous généralisons ici ce modèle pour la première fois dans l'optique de tenir compte des connections prioritaires. Nous caractérisons ensuite les cas dans lesquels un reroutage n'est pas possible et, à l'aide d'une transformation simple, nous montrons que le problème avec connections prioritaires peu se ramener au problème sans connections prioritaires. Enfin, nous proposons une nouvelle heuristique améliorant les heuristiques précédemment proposées.

Mots-clés : Reroutage, process number, vertex separation, largeur de chemins

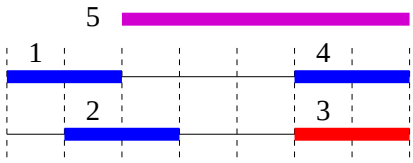
1 Introduction

Optimizing the usage of network resources is a critical issue for telecom companies operating high speed WDM backbone networks [14, 8]. The traffic demand increases rapidly (e.g. 6 millions new Internet users per month in China) and is subject to constant variation with the deployment of new services (mobile Internet, peer-to-peer, on-demand TV). Therefore, the routing of the traffic pattern has to be updated regularly to ensure an efficient usage of network resources and to preserve enough flexibility to accomodate new connection requests rapidly. For example, Fig. 1(a) represents a path network using two wavelengths where four connections are initially established. After the termination of request 3, it is not possible to accept request 5 although the routing depicted in Fig. 1(b) is possible. Hence, to reduce the blocking probability [16, 12], some requests have to be switched on other routes, and so the routing has to be reconfigured [10]. Fault tolerance is another critical concern of the networks. Indeed, failures in the backbone network may have an important impact with huge financial repercussions. Therefore, when a failure occurs, traffic restauration or protection mechanisms are used to ensure the continuation of affected connections. Such mechanisms are fast and ensure low traffic perturbation, but the resulting routing of connections requests may be inefficient. Since the reparing time of the network could be long (days), it is interesting to optimize the usage of resources and so to reconfigure the routing.

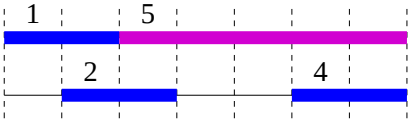
In this paper, we concentrate on routing reconfiguration problem. It consists in switching a set of connection requests from current routing to another pre-determined routing under the constraint that connections are switched one by one. Our study is independent of the destination routing and its computation is not considered here.

To switch an established connection from a lightpath to another, one has to ensure that destination resources are available. For instance, in Fig. 1(a), connection 4 must be switched before the establishment of connection 5. To model all dependencies between connections in the reconfiguration phase, we use the notion of dependency digraph [10]. Given the initial routing R and the new routing R' we want to achieve, the dependency digraph contains a node per connection and there is an arc from node u to node v if connection v uses resources in R that will be used by u in R'. So an arc from u to v indicates that connection v must be switched before connection u.

Clearly, when the dependency digraph is a DAG, the scheduling of the switchings is straightforward, starting from the leaves. However, cyclic dependencies may exist and so the dependency digraph may contains cycles. In such cases, reconfiguration is feasible only if we allow to temporarily interrupt some connections in order to break the dependency cycles. The objective is thus to minimize the number of connections that will be simultaneously interrupted. Initially, this problem has been studied in [10] with an heuristic algorithm. Later, the network reconfiguration problem has been defined in terms of *process number* [5, 6], an analogous of cops-and-robber games [11, 9]. Using this formulation, this problem has been shown NP-complete [5, 6]. More precisely, a game is defined where agents are successively placed to and removed from the vertices of the dependency digraph [5, 6]. In this setting, the minimum number of agents used during the game equals the smallest number of connections simultaneously interrupted during the reconfiguration process. In this



(a) Initial routing R_1 of requests $1, 2, 3, 4$. Request 3 finishes and a new request starts.



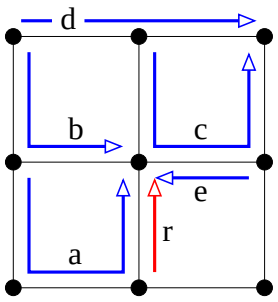
(b) The new routing R_2 satisfying requests $1, 2, 4, 5$.

Figure 1: Example of reconfiguration.

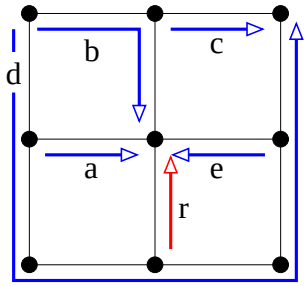
game, an agent placed on a node models the interruption of the corresponding connection, and a connection can be switched, or *processed*, if all its outneighbors in the dependency digraph are either covered by an agent or have already been processed. In other words, a connection can be switched if all of the connections using the same resources have either been interrupted, or have already been switched. Furthermore, an agent can be reused as soon as corresponding connection is switched. The process number is the minimum number of agents needed in the game.

The reconfiguration of the routing allows the network operator to optimize the usage of network resources. However, this process forces to interrupt temporarily some connections, and so induces some traffic perturbations that clients may not accept. Moreover, some clients may sign a specific contract forbidding interruptions, and so the operator offers two classes of services. To cope with these two classes, we introduce the new constraint that imposed some particular connections, called the *priority connections*, not to be interrupted. In the process number game formulation, this constraint is modeled by a particular class of nodes of the dependency digraph, called the `black` nodes, that are those nodes corresponding to the priority connections. During the game, the `black` nodes cannot host agents, i.e., the single way to process them is to deal with all their outneighbors first. It is worth-mentioning that a direct cycle of `black` nodes in the dependency digraph makes the reconfiguration impossible, thus the number of such nodes (and so clients) must be small. Furthermore, due to `black` nodes the process number of the dependency digraph may increased (see Section 3).

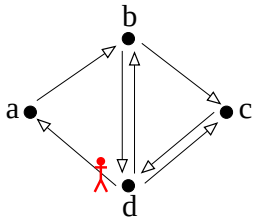
Section 2 is devoted to the formal definition of the process number and the game on the dependency digraph without `black` nodes. Then, in Section 3 we introduce the modeling with `black` nodes. We give impossibility results. Namely, we prove that reconfiguration can be performed if and only if no subset of `black` nodes induces a strongly connected component of the dependency digraph. Then, we show that both problems, with and without `black` nodes are equivalent. More precisely, when reconfiguration is possible, we present a simple transformation of any digraph D with `black` nodes (without strongly connected component induced by the `black` nodes) into another digraph D^* without `black` nodes with same process number and such that it is straightforward to adapt the rerouting strategy for D^* into a strategy for D. In Section 4, we recall the basic principles of the heuristic designed in [4] and propose some improvements to solve more efficiently our problem. Finally, in Section 5, we evaluate the respective efficiency of heuristic algorithms through simulations.



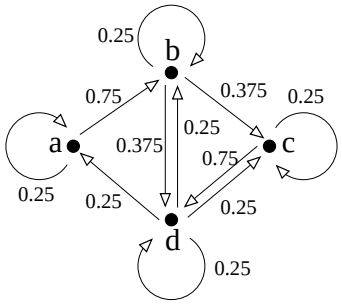
(a) Routing R_1 of requests a, b, c, d, e. The new request r cannot be satisfied.



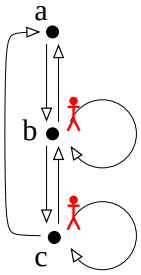
(b) Routing R_2 satisfying requests a, b, c, d, e and r.



(c) Dependency digraph D for switching from routing R_1 to R_2.



(d) D_μ from D to apply *flow circulation* algorithm.



(e) $D^*(d)$ from D if d is a `black` node.

Figure 2: Fig. 2(a) is a grid where links have a single wavelength in each direction. New request r can not be accepted, although the routing of Fig. 2(b) is possible. Figs. 2(c) and 2(d) represents dependency digraph D and corresponding D_μ to switch from routing R_1 to routing R_2. If d is a `black` node, digraph $D^*(d)$ of Fig. 2(e) is obtained from D by removing d and adding a complete bipartite digraph from $N^-(d)$ to $N^+(d)$.

2 Modeling

In this section, we recall the modeling proposed in [5, 6] to solve the routing reconfiguration problem when it is allow to interrupt any connection request during a reconfiguration phase, if needed.

Following [5, 6], the routing reconfiguration problem can be expressed as a cops-and-robber game, similar to the game-theoretical model of the *pathwidth* of a graph [15, 11]. Hence, it can be seen as an exploration of the digraph by agents using the following rules. Interrupting a request r is represented by placing an agent on the corresponding vertex u_r

in D. A vertex is said *processed* when the corresponding request has been rerouted, and we call a *process strategy* a sequence of the three following actions allowing to reroute all requests with respect to the constraints represented by the dependency digraph.

R_1. Put an agent on a vertex (*interrupt a connection*).

R_2. Remove an agent from a vertex if all its out-neighbours are either processed or occupied by an agent (*release a connection to its final route when destination resources are available*). The vertex is now processed (*connection has been rerouted*).

R_3. Process a vertex if all its out-neighbours are occupied by an agent (*destination resources are available, and so the connection can be rerouted*).

A *p-process strategy* is a strategy which process the digraph using p agents and the *process number*, pn(D), is the smallest p such that a p-process strategy exists. For example, a star has process number one and the strategy consists in positioning an agent on the central vertex (R_1). Thus all leaves can be processed (R_3) and finally the agent can be removed (R_2). A path of four vertices or more has process number two, a cycle of size five or more has process number three, and a $n \times n$ grid, $n \geq 3$, has process number $n + 1$.

In the example of Fig. 2, the network is a 3×3 grid where links have a single wavelength in each direction. With the routing of Fig. 2(a), the new request r can not be accepted but the routing of Fig. 2(b) is possible. Fig. 2(c) represents the dependency digraph for switching from routing R_1 to routing R_2. The process strategy is first to place an agent of node d (R_1). The remaining digraph is a direct path a, b, c, and so we process c, b and then a (R_3). Finally, the agent can be removed to process d (R_2). So the process number of this digraph is 1 and a single connection has to be temporarily interrupted.

Although digraphs with process number 1 and 2 have been characterized [6], computing pn(D) is NP-complete for general digraphs and the same result holds when D is a symmetric digraph and so for the underlying graph G. It has also been proved that $\mathrm{pw}(G) \leq \mathrm{pn(G)} \leq \mathrm{pw(G)} + 1$, where $\mathrm{pw}(G)$ is the pathwidth of G. Because the problem of computing the pathwidth is NP-complete in general [13], numerous works have been directed toward particular graphs' classes. For instance, several polynomial-time, or even linear-time, algorithms have been proposed that compute the pathwidth of trees [17], cograph [2], permutation graphs [1], circular arc graphs [18] (in particular, this class contains interval graphs), etc. Due to the relation between the pathwidth of a graph and its process number, all aforementioned results compute the process of number up to one for any of these graphs' classes. In [3], Bodlaender *et al.* proposed a $O(\log^2 n)$-approximation for the pathwidth problem in arbitrary graph. Furthermore, the pathwidth problem is not in APX [7], meaning that there is no polynomial time algorithm that can ensure a constant approximation factor k unless $P = NP$, for any constant k. This motivate us to develop efficient heuristic algorithms.

3 Coping with multiple classes

We now extend the model to the case where some requests, the *priority* connections, cannot be interrupted during a reconfiguration phase. In our setting, the priority connections are modeled by a specific class of vertices of the dependency digraph, called the `black` nodes. The constraint over the priority connections can be reformulated by the fact that a `black` node cannot host an agent during the process strategy. So rule R_1 can be applied only on other nodes. For example, suppose that node d of Fig. 2(c) is a `black` node. In this situation the process strategy consists in putting agents on both nodes b and c (R_1) which allow to process node a (R_2), then to process node d (R_2) and finally to process nodes b and c. Such strategy requires 2 agents while the process number of the digraph of Fig. 2(c), without `black` node, is 1. Note that the presence of a single `black` node may increase drastically the number of agents. For instance, consider a star with a large number of leaves where the central node is a `black` node.

Let $\mathrm{pn}^*(\mathrm{D}, \mathrm{Q})$ denote the process number of a digraph $D = (V, A)$ with a set $Q \subseteq V$ of `black` nodes. We will show in Sec. 3.1 that in some cases it is not possible to perform the reconfiguration and thus $\mathrm{pn}^*(\mathrm{D}, \mathrm{Q})$ is not defined. Then, we will show in Sec. 3.2 that, when the reconfiguration is possible, we have $\mathrm{pn}^*(\mathrm{D}, \mathrm{Q}) = \mathrm{pn}(\mathrm{D}^*)$, where D^* is a digraph without `black` nodes obtained from D by a simple transformation.

3.1 Impossibility results

When the dependency digraph contains a direct cycle of `black` nodes, it is impossible to break it and so no feasible process strategy exists. So we have:

Lemma 1. *Let $D = (V, A)$ be a dependency digraph and let $Q \subseteq V$ be the set of `black` nodes. If D contains a direct cycle of `black` nodes then no process strategy exists.*

Proposition 2. *Given a dependency digraph $D = (V, A)$ and a set of `black` nodes $Q \subseteq V$, we can decide in time $O(|V| + |A|)$ if a process strategy exists.*

Proof. The existence of a process strategy relies on the existence of a direct cycle of `black` nodes. So, it suffices to test if the digraph D_Q, obtained by removing all nodes of $V - Q$ from D, has a strongly connected component. □

As a consequence of Lemma. 1, it is always possible to compute a process strategy when $|Q| \leq 1$. Now, when $|Q| \geq 2$ an interesting question is to determine the probability that the dependency digraph contains a strongly connected component of `black` nodes. Clearly, this probability will increase with $|Q|$, and so we have to determine the size of Q for which this probability will be 1 with high probability.

Indeed, there is a tradeoff to determine for operators between the number of clients for which no traffic interruption is allowed and the blocking probability. When the number of such client is large, then it is almost impossible to reconfigure the routing and so the blocking probability will be high.

3.2 Transformation

To compute the number of agents needed to process a digraph D containing `black` nodes, we can construct a digraph D^* from D without `black` nodes such that the process number of D^* equals the number of agents needed to process the initial digraph D with black nodes. Moreover, the process strategy for D directly follows the process strategy of D^*.

Definition 3. *Let $D = (V, A)$ be a digraph and let $v \in V$. The digraph $D^*(v)$ is obtained from D by removing v and adding an arc from every vertex of $N^-(v)$ to every vertex of $N^+(v)$. Note that the preceding transformation can yield loops if $N^-(v) \cap N^+(v) \neq \emptyset$.*

For example, let D be the digraph depicted in Fig. 2(c) and let d be a `black` node. Using above transformation, we obtain the digraph D^* represented in Fig. 2(e). Since we have $N^-(d) = \{b, c\}$, $N^+(d) = \{a, b, c\}$, and so $N^-(v) \cap N^+(v) = \{b, c\}$, nodes b and c have loops in D^*. We have $\mathrm{pn}(\mathrm{D}^*) = 2$ and from a process strategy for D^*, we can deduce a process strategy for $(D, \{d\})$, as will be proved in Proposition 4.

Proposition 4. *For every digraph $D = (V, A)$ and set $Q := \{v\}$ for some vertex $v \in V$ which has no loop, we have* $\mathrm{pn}^*(\mathrm{D}, \mathrm{Q}) = \mathrm{pn}(\mathrm{D}^*(\mathrm{v}))$.

Proof. Let P^* be a processing strategy for (D, Q). We apply it to $D^*(v)$, ignoring the step that processes the vertex v. This processes the whole graph $D^*(v)$: there can only be a problem when we process a vertex $u \in N_D^-(v)$. But when any such step is reached, note that (since P^* is a valid strategy) the vertex v has already been processed earlier. Consequently, all the vertices of $N_D^+(v)$ are already processed or covered by an agent, and the vertex u can safely be processed. Thus, $\mathrm{pn}(\mathrm{D}^*(\mathrm{v})) \leq \mathrm{pn}^*(\mathrm{D}, \mathrm{Q})$.

Conversely, let P be a processing strategy for $D^*(v)$. We obtain P^* from P by adding the step "process v" just before the first vertex of $N^-(v)$ is processed. By the definition of $D^*(v)$, all the vertices of $N^+(v)$ must be already processed or covered by an agent. Consequently, we obtain a valid processing strategy for (D, Q), and thus $\mathrm{pn}^*(\mathrm{D}, \mathrm{Q}) \leq \mathrm{pn}(\mathrm{D}^*(\mathrm{v}))$. This concludes the proof. □

More generally, by recursively applying Proposition 4, given a dependency digraph $D = (V, A)$ and a set $Q \subseteq V$ of `black` nodes, we can construct digraph D^* without `black` nodes and deduce the process strategy of (D, Q) from the process strategy for D^*.

Corollary 5. *Let $D = (V, A)$ be a digraph and let $Q := \{v_1, v_2, \ldots, v_f\} \subseteq V$. We set $D_1 := D^*(v_1)$ and, for each $i \in \{2, 3, \ldots, f\}$, we set $D(v_{i+1}) := D_i^*(v_{i+1})$ if v_{i+1} has no loop in V_i. Then, $p^*(D, Q)$ is finite if and only if D_f is defined, and then* $\mathrm{pn}^*(\mathrm{D}, \mathrm{Q}) = \mathrm{pn}(\mathrm{D_f})$.

Notice that if v_{i+1} has a loop in V_i, then D contains a direct cycle of `black` nodes and so the digraph cannot be processed.

To conclude this section, we prove an upper bound on the amount of extra searchers needed to process a dependency digraph $D = (V, A)$ containing `black` nodes compared to the process number of D without `black` nodes. Let $Q \subseteq V$. In the following, $N^+(Q)$ denotes the set of vertices that are outneighbors of some vertex in Q.

Proposition 6. *For every digraph $D = (V, A)$ and set $Q \subseteq V$ of* `black` *nodes,* $\mathrm{pn}^*(\mathrm{D}, \mathrm{Q}) \leq \mathrm{pn}(\mathrm{D}) + |\mathrm{N}^+(\mathrm{Q})|$. *Moreover, this bound is asymptotically tight.*

Proof. Let P be a processing strategy for D using $\mathrm{pn}(\mathrm{D})$ agents. Let us apply P to (D, Q) with the following adaptations. At every step of P that consists of placing an agent at some vertex $v \in Q$, let us replace this operation by the sequence of operations that consists of placing an agent at every node in $N^+(v)$. If a vertex in $N^+(v)$ is a `black` node, we also place an agent at every vertex of its outneighborhood and so on recursively. Then, all the `black` nodes considered at this step are processed. Now, if at some step of the process strategy, P decides to place an agent at an already occupied vertex (this may happen if this vertex is the outneighbor of a `black` node), then we simply skip this step. Finally, if P decides to process a vertex that is not occupied (in P) but that is occupied by an agent following the modified strategy, then this agent is removed. It is easy to see that the strategy defined above is a process strategy for (D, Q) because when a vertex v of D is occupied by an agent in P, either it is also the case in our strategy, or v is a `black` node and has been processed. Moreover, the modified strategy trivially used at most $\mathrm{pn}(\mathrm{D}) + |\mathrm{N}^+(\mathrm{Q})|$ agents.

To prove that this bound is tight, let us fixed $k \geq 1$. Consider the digraph D that consists of k symmetric cliques of size k: $C_1, C_2, \cdots, C_k$ and of the $k+1$ vertices $v_0, \cdots, v_k$. Now, add an edge from v_0 to v_i for any $i \leq k$. Then, for any $i \leq k$, let us add an edge from v_i to any vertex of C_i and from any vertex of C_i to any vertex of C_{i+1}. Finally, add an edge from any vertex of $U_{i \leq k} C_i$ to v_0. We first prove $\mathrm{pn}(\mathrm{D}) = \mathrm{k} + 1$. Indeed, the strategy consists in placing an agent at v_0, processing $v_1, \cdots, v_k$, and finally placing successively an agent at every vertex of C_i, for i from 1 to k. To conclude, let $Q = \{v_0, \cdots, v_k\}$. We prove that $\mathrm{pn}^*(\mathrm{D}, \mathrm{Q}) = \sum_{\mathrm{i} \leq \mathrm{k}} |\mathrm{C_i}|$. By applying the transformation described above at the vertices $v_1, \cdots, v_k$, we obtain a digraph in which the `black` node v_0 is a in-neighbor of all vertices in $\bigcup_{i \leq k} C_i$. The result follows Proposition 4. □

4 Heuristic algorithms

In this section we present an heuristic for the reconfiguration problem. This heuristic is based on an improvement of the heuristic proposed in [4]. We first recall it and then we further study its behaviour and possible improvements. Then, we explain how this heuristic can be used when the network contains some `black` nodes.

4.1 Principles of the heuristic in [4]

Before going into the detail of the algorithm proposed in [4], let us notice that the process number of a digraph is simply the maximum over the process numbers of its strongly connected components. Therefore, at each step of the execution of the heuristic in [4], a similar process will be executed to any strongly connected components of the digraph.

The heuristic of [4] proceeds as follows. At each step, an unprocessed and unoccupied vertex is chosen. Then, an agent is placed at this vertex and the protocol processes all

possible vertices. Moreover, the agents occupying a processed vertex are removed. The main part of the heuristic clearly consists in choosing the vertex where an agent will be placed. For this purpose, [4] proposes to use the *flow circulation* method. Intuitively, every vertex of the network wants to be processed as soon as possible without being interrupted. For this purpose, the better choice for a vertex v would be to place an agent at one of its outneighbors. The flow circulation method somehow models a voting process at the end of which the vertex that maximize the happiness of all the vertices is chosen.

Let us describe the flow circulation method more formally. Let us consider a strongly connected component C of the digraph induced by the not yet processed nor occupied vertices. A weight is assigned to any vertex of C, then a flow circulates in the following way. At each round, any vertex u in C sends a uniform fraction of its weight to its out-neighbors. Simultaneously, u receives a fraction of the weight of its in-neighbors and updates its new weight as the sum of the flows it has received. The same process is executed during $|V|$ rounds and the vertex with maximum final weight is chosen. By modeling this process by a discrete Markov chain, Coudert and Mazauric proved that the vector of weights always admits a unique stationary vector and that convergence is achieved after $|V|$ rounds [4]. In Fig. 2(d), we can show the digraph of Fig. 2(c) from a Markov point of view to apply *flow circulation* method. See [4] for more details.

Lemma 7. *The worst case time complexity of the heuristic proposed in [4] is* $O(n^2(n+m))$.

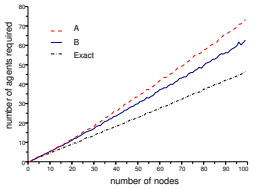
4.2 Improvements

In this section, we further study the behaviour of the heuristic described above and propose a modification allowing to improve its performances.

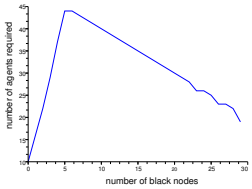
To illustrate the behaviour of the heuristic in [4], let us consider the graph defined by a path of cliques. More precisely, the (symmetric) graph G consists of $r > 0$ cliques $C_1, \cdots, C_r$ each of size 6 such that, for any $0 < i < r$, C_i intersects C_{i-1} and C_{i+1} in 2 vertices and no other cliques intersect C_i. The optimal process strategy for G consists in placing agents at the vertices in $C_1 \cap C_2$. Then, more agents are placed at the vertices of C_1 but in one vertex in $C_1 \setminus C_2$, that is then processed. Then, all agents at $C_1 \setminus C_2$ are removed and placed at $C_2 \cap C_3$. Again, other agents are placed at the yet unoccupied vertices of $C_2 \setminus C_3$ but one, that is then processed. And so on. Such a strategy uses exactly 5 agents.

In this particular class of graph, we can observe a faulty behaviour of the heuristic. Indeed, while the first steps executed by the algorithm consists in placing agents at the vertices of C_1, once the vertices in $C_1 \setminus C_2$ have been processed, the algorithm places agents at the vertices in $C_3 \setminus C_4$. Finally, some agents are placed in the vertices in $C_1 \setminus C_2$ and the vertices in C_2 are occupied and processed. Therefore, at least 7 agents are used. It is worth to notice that during the processing of the vertices in C_2, the agents occupying $C_3 \setminus C_4$ are useless and should not be taken into account. The new heuristic that we propose takes advantage of this remark.

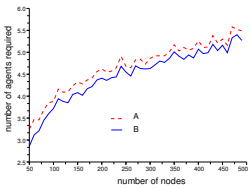
The above remark leads to the following improvement of the heuristic of [4]. At every step of the execution of the previous algorithm, after having chosen a vertex at which an agent



(a) Circular arc graphs



(b) Circulant digraphs with 50 nodes and arcs from node i to nodes $\{i+1, i+2, \ldots, i+10\}$.



(c) 2-digraphs

Figure 3: Simulation results on circular arc graphs, circulant digraphs and 2-digraphs. A correspond to the heuristic algorithm proposed in [4] and B to the heuristic algorithm proposed in this paper. For each size of digraph, 100 digraphs are randomly generated for our experiments.

will be placed, the vertices on which rule R_2 applies are processed. In our algorithm, before processing those nodes (and only if at least one such node exists), we remove all agents that are occupying a node v not in-neighbors of which have been processed previously. Indeed, such a vertex v has not been used before in contrast with another kind of vertices occupied by agents that cannot be removed: those occupied nodes belonging to the in-neighborhood of some processed node u that have been used for the processing of u. The agents that have been removed at this stage are not taken into account in the count of the number of agents.

To conclude this section, we show that the heuristic can easily be adapted in the case of digraphs containing `black` nodes. Using this heuristic, it is indeed not necessary to implement the transformation described in Section 3. It is sufficient to proceed in the following way: when choosing a candidate vertex to place an agent, `black` nodes are not considered. So, the complexity of the algorithm is unchanged.

5 Simulations

To validate the heuristic algorithm proposed above, we have done experiments on different topologies. Recall that computing the process number is NP-complete in general. Therefore, and to make fair comparisons, we have restricted ourself to digraph classes for which efficient exact algorithms (or good approximations) computing the process number have been designed. For each type of topology, described below, and for a range of networks'size, we create 100 random instances of networks belonging to this type. Then we run on each of them the heuristic proposed in [4], the improved heuristic designed in Section 4, and an algorithm computing the pathwidth (and so the process number up to one). The results are represented in Fig. 3.

Circular arc graphs are particularly interesting because they corresponds to the case where the physical topology is a ring. A graph is a *circular arc graph* if (i) its vertices

can be represented by a subinterval over the circular interval $[1, n]$ and (*ii*) two vertices are adjacent if the corresponding interval intersect. When the network is a ring, each communication between processors represents an interval. Therefore, when proceeding to rerouting, two paths will interfer if the corresponding intervals intersect. This yield to a dependency digraph that is precisely a circular arc graph. To make comparisons, we have also implemented an algorithm to compute optimaly the pathwidth of circular arc graphs [18]. Fig. 3(a) shows that our heuristic algorithm (B) is better than (A) [4], and that the deviation with optimal value is small.

Digraphs with process number 2 have been characterized in [6], where a polynomial time algorithm to decide whether a digraph has process number 2 or not is also given. On such digraphs, the improvement of our heuristic algorithm is small compare to [4], as reported in Fig. 3(c). However, it shows that our algorithms are relatively fast, since it took us only one minute to perform all the simulations depict in Fig. 3(c) on a standard laptop.

We now consider *circulant* digraphs with n vertices $(v_1, \ldots, v_n)$ and arcs from v_i to $\{v_{i+1}, v_{i+2}, \ldots, v_{i+k}\}$, for $i = 1 \ldots n$ and for some fixed k (indexes are taken modulo n). In particular, the directed cycle $(v_1, \ldots, v_n)$ is a subgraph of D. Simulation has been done in order to illustrate the impact of the `black` nodes in terms of process number (see Proposition 6). It is easy to see that, in absence of `black` nodes, the optimal process strategy uses k agents and consists of the following: place agents at $(v_1, \ldots, v_k)$ and then, iteratively processes v_n, v_{n-1} to v_{k+1}. Without any `black` node, our heuristic computes this optimal process number for any pair n, k. In Fig. 3(b), we consider a circulant graph with 50 nodes and outdegree $k = 10$. We gradually increase the number $|Q|$ of `black` nodes. For each value of $|Q|$, we run 100 tests with random positions of the `black` nodes and plot the maximum value found. While $|Q| \leq 5$, with high probability, the `black` nodes create a configuration similar to the one of the graph defined in the proof of Proposition 6, increasing the number of agents up to $\sum_{v\ \texttt{black}node} |N^+(v)|$. When $|Q|$ increases too much, with high probability, strongly connected components consisting of `black` nodes are created leading to many impossible reconfigurations. Simultaneously, when $|Q|$ is large enough, the only valid configurations are very specific and forbid situations like in Proposition 6.

6 Conclusion

In this paper, we have investigated the reconfiguration problem: switching connections routed in a WDM backbone network from current routing to a pre-determined destination routing under the constraints that (1) connections are switched one after the other, and (2) some connections may refused any temporarily interruption. This last constraint, modeled using `black` nodes in the dependency digraph, may cause impossibility situation for the reconfiguration and so the number of `black` nodes must remain small. Then, we have shown how to handle `black` nodes in the process strategy by constructing a new dependency digraph without `black` nodes. Finally, we have proposed a new heuristic algorithm that performs well according to our experiments.

An interesting open question is now to evaluate the number of priority clients that an operator may accept in the network in order to ensure the faisability of the reconfiguration with high probability.

Acknowledgments

This work has been partially supported by région PACA, and European project IST FET AEOLUS.

References

[1] Hans L. Bodlaender, Ton Kloks, and Dieter Kratsch. Treewidth and pathwidth of permutation graphs. *SIAM J. Discrete Math.*, 8(4):606–616, 1995.

[2] Hans L. Bodlaender and Rolf H. Möhring. The pathwidth and treewidth of cographs. *SIAM J. Discrete Math.*, 6(2):181–188, 1993.

[3] H.L. Bodlaender, J.R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, March 1995.

[4] D. Coudert and D. Mazauric. Network reconfiguration using cops-and-robber games. Technical Report inria-00315568, INRIA, August 2008.

[5] D. Coudert, S. Perennes, Q.-C. Pham, and J.-S. Sereni. Rerouting requests in wdm networks. In *AlgoTel'05*, pages 17–20, Presqu'île de Giens, France, mai 2005.

[6] D. Coudert and J-S. Sereni. Characterization of graphs and digraphs with small process number. Research Report 6285, INRIA, September 2007.

[7] N. Deo, S. Krishnamoorthy, and M. A. Langston. Exact and approximate solutions for the gate matrix layout problem. *IEEE Transactions on Computer-Aided Design*, 6:79–84, 1987.

[8] R. Dutta, A. Kamal, and G. Rouskas, editors. *Traffic Grooming for Optical Networks: Foundations, Techniques and Frontiers.* Optical Networks. Springer, August 2008.

[9] F. Fomin and D. Thilikos. An annotated bibliography on guaranteed graph searching. *Theor. Comput. Sci.*, 399(3):236–245, 2008.

[10] N. Jose and A.K. Somani. Connection rerouting/network reconfiguration. *Design of Reliable Communication Networks*, pages 23–30, October 2003.

[11] M. Kirousis and C.H. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47(2):205–218, 1986.

[12] K. Lu, G. Xiao, and I. Chlamtac. Analysis of blocking probability for distributed lightpath establishment in WDM optical networks. *IEEE/ACM Trans. Netw.*, 13(1):187–197, February 2005.

[13] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou. The complexity of searching a graph. *J. Assoc. Comput. Mach.*, 35(1):18–44, 1988.

[14] B. Mukherjee. *Optical WDM Networks*. Springer, 2006.

[15] N. Robertson and P. D. Seymour. Graph minors. I. Excluding a forest. *J. Combin. Theory Ser. B*, 35(1):39–61, 1983.

[16] P. Saengudomlert, E. Modiano, and R. Gallager. On-line routing and wavelength assignment for dynamic traffic in WDM ring and torus networks. *IEEE/ACM Trans. Netw.*, 14(2):330–340, 2006.

[17] K. Skodinis. Construction of linear tree-layouts which are optimal with respect to vertex separation in linear time. *Journal of Algorithms*, 47(1):40–59, 2003.

[18] Karol Suchan and Ioan Todinca. Pathwidth of circular-arc graphs. In *Proceedings of the 33rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 258–269, 2007.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399