Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computers & Operations Research 33 (2006) 2891–2917

computers & operations research

www.elsevier.com/locate/cor

On-line resources allocation for ATM networks with rerouting

Samir Loudni[a,*], Patrice Boizumault[a], Philippe David[b]

[a]*GREYC, CNRS UMR 6072, Université de Caen, Campus 2, 14032 Caen Cedex, France*
[b]*Ecole des Mines de Nantes - BP 20722, 44307 Nantes Cedex 3, France*

Available online 16 February 2005

Abstract

This paper presents an application we developed for France Telecom R&D to solve a difficult real-life network problem. The problem takes place in an Asynchronous Transfer Mode (ATM) network administration context and consists in planning demands of connection over a period of 1 year. A new demand is accepted if both bandwidth and Quality of Service (QoS) requirements are satisfied. Demands are not known prior to the assignment and must be performed *on-line* according to their arrival. Moreover, the acceptance or the reject of a demand must be decided within a given time of 1 *min*.

First, we look for a route satisfying the new demand. In case of failure, we try *to reroute* some already accepted connections in order to satisfy this new demand. Rerouting has been modelled as a Weighted Constraint Satisfaction Problem (WCSP) and solved by VNS/LDS + CP, a hybrid method well suited for solving WCSPs in *on-line* contexts. Experiments show that our rerouting enables to accept an average of 67% of demands that would be rejected otherwise.

Keywords: Network routing; ATM; Quality of service; Dijkstra shortest path algorithm; Anytime context; Constraint satisfaction problem; WCSP; Hybrid search algorithm; VNS; LDS

1. Introduction

The up-coming Gbps (Giga bits per second) high speed networks are expected to support a wide range of real-time multimedia applications. The requirement for time delivery of digitized audio-visual

* Corresponding author.
E-mail addresses: loudni@iutc3.unicaen.fr (S. Loudni), patrice.boizumault@info.unicaen.fr (P. Boizumault), philippe.david@emn.fr (P. David).

0305-0548/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2005.01.016

information raises new challenges for the next generation integrated-service broadband networks. One of the key issues for these newly emerged architectures is Quality of Service (QoS) routing, i.e., computing paths that satisfy a set of end-to-end QoS requirements. In this case, the objective of QoS routing is two-fold: (1) to find a suitable route which fulfills the demand's requirements, (2) to optimize the network resource allocation.

In this paper, we present an application we developed for France Telecom R&D. This application takes place in an Asynchronous Transfer Mode (ATM) network administration context, and is very close to a QoS routing problem. The problem consists in establishing connections, i.e., assigning routes to demands respecting QoS requirements and thus reserving necessary resources. Demands are planned over a *period of* 1 *year*. But demands are not known prior to the assignment and must be performed on-arrival in an on-line context. Moreover, when no route (respecting the QoS requirements) can be found for a demand, we allow *rerouting* of some connections that have already been established. If rerouting fails, the connection is blocked and the demand will be rejected.

Since routes are computed on-line as demands arrive, there is an additional delay in connection setup. This delay may be significant because computing a route satisfying multi-QoS criteria is often an NP-Complete problem (see Section 7). For these reasons, FT R&D has limited the amount of time spent searching for a route to *one minute*. Thus, accepting/rejecting a demand must be performed in at most 1 min.

This paper describes a rerouting-based approach for solving this problem. Rerouting is achieved in two main steps: computing a set of *demand conflicts* (see Section 4) and repairing these conflicts (see Section 5).

The aim of the first step is to find a route for a demand (without rerouting) or, in case of failure, to compute a set of demand conflicts. These demand conflicts are generated from a set of *conflicting shortest paths*,[1] provided by an extended Dijkstra's algorithm (EDSP).

In the second step, for each *demand conflict*, a Weighted Constraint Satisfaction Problem (WCSP), restricted to the area of the network where connections have to be rerouted, is built and then solved using a hybrid search method VNS/LDS + CP [1,2]. WCSPs corresponding to various *demand conflicts* are successively treated. If a solution to one WCSP is found (within less than 1 min), the demand will be accepted. If not, the demand will be rejected.

VNS/LDS + CP is a hybrid method [2] for solving constraint optimization problems formulated as WCSP (Weighted CSP). It combines a *Variable Neighborhood Search* [3] and a *Limited Discrepancy Search* [4] with *Constraint Propagation* to efficiently guide the search.

Experimental results for real life ATM network topologies and sets of demands provided by France Telecom R&D show that rerouting with VNS/LDS + CP enables to accept an average of 67% of demands that would be rejected by a greedy algorithm without rerouting [5].

The paper is organized as follows: first, we describe resource allocation in ATM networks, characterize the rerouting problem and review different approaches for solving it; then, we present the two main steps of our approach: computing the set of *demand conflicts* and rerouting using VNS/LDS + CP; next, we give an example, review some related works and present experimental results. Finally, we conclude with future works.

[1] A path p is in conflict with a demand d if d requires too much bandwidth on at least one link l of p, more precisely on at least one time interval on l.

2. Resource allocation in ATM networks

The EOLE project is a RNRT (Réseau National de Recherche en Télécommunications) project supported by the French Research Office. The aim of the project[2] is to solve on-line telecommunications combinatorial problems using Constraint Programming [6]. The application, we describe in this paper, is the second pilot application of the EOLE project.

Section 2 is organized as follows: first, we describe ATM technology; then, we introduce quality of service and metrics and consider them more specifically for ATM networks; finally, we present connections reservation for ATM networks.

2.1. ATM technology

In asynchronous transfer mode (ATM) networks, multiplexing and switching are performed on 53-byte cells. Two levels of cell forwarding are defined: virtual path (VP) and virtual channel (VC), which use the virtual path identifier (VPI) and virtual channel identifier (VCI) fields in the cell header, respectively [7]. According to such hierarchical structure, a collection of VP links (or physical links) form a virtual path connection (VPC) whereas a set of VC links identify a virtual channel connection (VCC) on which cells are transported across the network.

ATM is *connection oriented*. Once the route has been chosen, a virtual connection is set (connection set-up) before information is transferred from the terminal to the network. This establishment includes the allocation of a VCI and/or VPI, as well as the reservation of required bandwidth resources for links forming the route, in order to ensure the QoS.

The mechanism that is responsible for accepting or rejecting a connection request is called *Connection Admission Control* (CAC). It uses rules that depict bandwidth constraints to satisfy in order to route a demand on links. According to these rules, the CAC will determine whether a connection can be set up satisfying the requested QoS while maintaining the QoS of all the other connections (see Section 2.3).

2.2. Quality of Service and metrics

As defined in [8], Quality-of-Service (QoS) is a measurable level of service delivered to network users, which can be characterized by parameters such as packet loss ratio, available bandwidth, end-to-end delay, route length, etc. Such QoS can be provided by network service providers in terms of some agreement (Service Level Agreement, or SLA) between network users and service providers.

Service requirements have to be expressed in some measurable QoS metrics. Well-known metrics include bandwidth, delay, jitter, cost, loss ratio, etc. There are three kinds of *metrics* [9]: additive ones, multiplicative ones, and concave ones. They are defined as follows:

Let $P = (l_1, l_2, \dots, l_k)$ be a path of length k, where $l_1, \dots, l_k$ represent network links.

- *m is additive*, if $m(P) = m(l_1) + m(l_2) + \cdots + m(l_k)$,
  delay, jitter, cost and route length are additive metrics.
- *m is multiplicative*, if $m(P) = m(l_1) * m(l_2) * \cdots * m(l_k)$,
  loss ratio is a multiplicative metric.

[2] See http://www.lcr.thomson-csf.com/projets/www_eole.

- *m is concave*, if $m(P) = \min\{m(l_1), m(l_2), \ldots, m(l_k)\}$,

  an example is bandwidth: the available bandwidth of a path is the minimal available bandwidth of its links.

2.3. QoS in ATM network

ATM networks support multiple service classes (ATCs: ATM Transfert Capacity), with different QoS. For this problem, we consider only two service classes [10], CBR and VBR:

- *CBR* (*Constant bit rate*): for applications where bandwidth requirement is constant over the duration of a connection; the fixed value is specified in the connection contract and corresponds to the highest peak (PCR: Peak Cell Rate).
- *VBR* (*Variable bit rate*): for applications where bandwidth requirements can vary during the lifetime of a connection; for ATM networks, VBR is described by three parameters: the peak cell rate (PCR), the sustainable cell rate (SCR), which is an upper bound on the realized mean cell rate of the connection, and the maximal burst size (MBS) [11].

According to the service he needs, the user selects one of the two ATCs. The associated CAC (Connection Admission Control) rule is applied to control demands admission and resources allocation. Inequality (1) is the CAC implemented by France Telecom [12]:

$$\text{for each link of capacity } C: \quad \sum PCR_{ATC=CBR} + \sum SCR_{ATC=VBR} < C. \tag{1}$$

Thus, only two traffic parameters are actually used for connections reservation. Moreover, with such a rule, either the first or the second parameter is taken into account, according to the ATC chosen by the user.

2.4. Connections reservation for ATM networks

In this section, we present the specificities of the ATM framework provided by France Telecom R&D.

2.4.1. Connections reservation service

In ATM networks, three tasks have to be performed by the network manager:

- equipments follow-up, which concerns the network infrastructures.
- connections set-up on equipments, which is performed by the service manager attached to the equipments.
- treatment of demands, i.e., computing resources availability in order to decide whether a new demand can be accepted, possibly at the cost of rerouting of some already accepted connections.

In this application, to enable reservations, we are only interested in the global computing of resources availability.

For ATM networks, real-time routing is done by *signalling* with PNNI[3] [13], a hierarchical and dynamic link-state routing protocol. This is based on routing table, and on a local and partial knowledge.

[3] Private network-to-network interface.

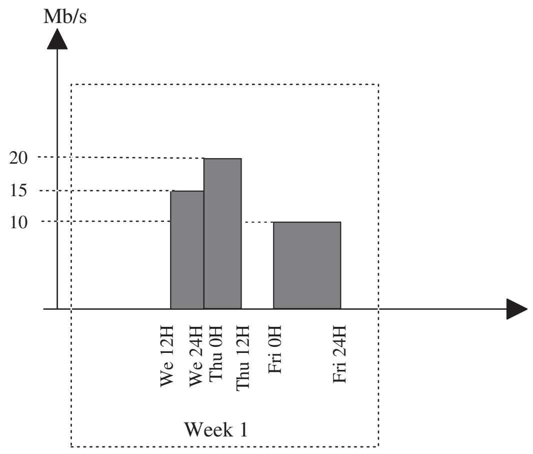

Fig. 1. Calendar for C_2.

In our case, a centralized system enables us to order equipments to establish a connection, under the control of an operator, and signalling is only limited to provide a guaranteed QoS on the selected path for a connection.

2.4.2. *Communication network*

ATM networks are constituted of nodes and communication links; links are characterized by their *bandwidth capacity*. ATM networks are *multi-graphs* $\langle \mathcal{N}, E \rangle$: for security reasons, several links may exist between two nodes. Nodes ($\mathcal{N}$) represent routers and hosts. Edges (E) represent communication links. As links are *asymmetric*, each link is represented by two directed edges in opposite directions.

Each demand d is defined by:

- a source node (s) and a destination node (t),
- an ATM Transfer Capacity parameter, depending on the required QoS,
- a traffic calendar.

Depending on the service he needs, the user chooses one ATC. For each ATC, rules depict bandwidth constraints to satisfy in order to accept the demand on links (see inequality (1)). The traffic calendar describes the traffic characteristics, such as a starting date, debit of the traffic and periodicity. Debit of the traffic is at least 1 h and is divided into quarters. As reservations are planned for a year, 35,000 slots of time have to be considered for the whole planning.

The user asks for a non-permanent connection, during at most one year. The required traffic must be constant on every time interval, and may be periodic. For example, one can ask for those calendars:

- *periodic* (C_1): 45 Mb/s every day, from 6.15 a.m. to 8.15 p.m.;
- *non-periodic* (C_2): week 1, 15 Mb/s on Wednesday, from noon to midnight, 20 Mb/s on Thursday, from midnight to noon, and 10 Mb/s on Friday, all day long (see Fig. 1);
- *continuous* (C_3): 50 Mb/s.

Several connections may be routed through a same link. But, because of technological limitations, it is impossible to share a connection among multiple routes (*mono-routing*). Retained QoS parameters are capacity and route length. Capacity is a concave metric per slot of time, whereas route length is an additive one. The length of a route is defined as the number of links constituting it.

2.4.3. Rerouting in ATM networks

The aim of our application is to find a route for each new demand; such a route must fulfil QoS requirements and must be found within the allowed computing time. Rerouting is achieved in two main steps: computing a set of *demand conflicts* and repairing these conflicts.

Step 1 tries to find a route for the demand (without rerouting). In case of failure, step-1 computes a set of demand conflicts from *conflicting shortest paths*. Each *demand conflict* represents a sub area of the network where rerouting can be performed. If a rerouting attempt succeeds within the allowed computing time, the demand will be accepted. If not, the demand will be rejected.

In ATM networks, rerouting cannot be carried out during a time slot where a connection is active, i.e., its debit is not null.

Our main optimization criterion is to maximize the number of accepted demands on the network. We will also take into account secondary criteria when we look for solutions:

- minimizing the number of rerouted connections (we are looking for stable solutions),
- minimizing the global length of routes (we are looking for shortest paths),
- balancing the network availabilities, in order to avoid over-loaded links.

3. Retained approach

In this section, we motivate the main choices of our approach. First, we propose an efficient time representation. Then, we characterize the rerouting problem and review different approaches for solving it. Finally, we present the two main steps of our approach:

1. We look for a *non-conflicting shortest path*. If there exists such a path, the demand is accepted. If not, for each *conflicting shortest path*, a *demand conflict* is computed.
2. We try to successively reroute *demand conflicts* formulated as WCSP. If a solution to one WCSP is found within the time assigned to handle a demand (1 min), the demand will be accepted. If not, the demand will be rejected.

3.1. Efficient time representation

One important difficulty comes from the duration of the planning. Indeed, to reserve a connection over a time period divided into n time slots, would require to solve n routing problems. As the number of time slots to consider is prohibitive (a year corresponds to 35,000 quarters of an hour), it is absolutely vital to have an effective and compact time representation.

In [12], a hierarchical structure close to Bryant's Binary Decision Diagrams [14], and called RNT (Restricted N_ary Tree), has been proposed to represent the time dimension as a compact tree. Superfluous data due to periodic demands are suppressed by creating a hierarchical level for each type of periodicity.

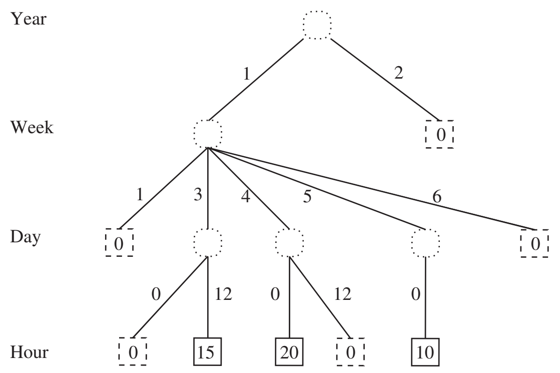

Fig. 2. Representation by RNT of calendar $C2$.

Superfluous data due to long constant slots are suppressed by gathering together consecutive and similar subtrees. Fig. 2 presents such a modelling for calendar $C2$.

We have used this structure to represent the traffic calendar of a demand, as well as the availability of a link. Several new operators on RNT have been defined in order to determine availability of links and to update their bandwidth capacities.

3.2. Characteristics of the problem

- This problem is *dynamic*: availability of network resources evolves with time. Due to rerouting, we have to reconsider routes of some already accepted connections. We look for stable solutions (by carrying out a few modifications for routes of rerouted connections).
- This is an *anytime* problem: a solution has to be computed within a given time (less than 1 min per new demand). Afterwards, the demand will be rejected. Section 3.3 describes more precisely *anytime* algorithms.
- This is a *hard* problem, due to the complexity of routing several demands. Moreover, allocating a single demand under multiple additive or multiplicative metrics has been proved to be NP-complete by itself [9].
- This is a *large-scale* problem, especially on its time dimension. Currently, reservations are planned over a period of 1 year, divided into quarters (35,000 time slots). The size of the network is about 100 nodes, 800 links, and 700 connections to be routed for a year.

3.3. Anytime algorithms

The term *anytime* algorithm was coined by Dean and Boddy in the mid-1980s in the context of their work on time-dependent planning [15]. Contrary to a standard algorithm, where no result is available until its ending, an *anytime* algorithm can be stopped, at “any time”, to provide a solution. Moreover, the quality of the computed solutions must increase over time. There are two kinds of *anytime* algorithms, namely, *interruptible* and *contract* ones [16]. A contract algorithm requires that the computing time must be known prior to its activation.

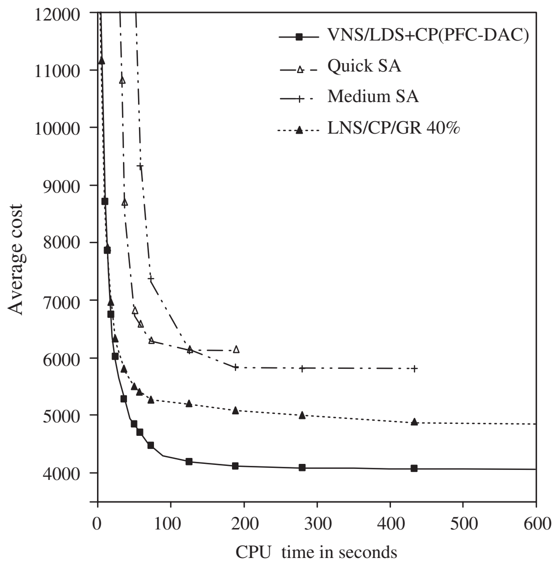

Fig. 3. Comparing performance profiles on the instance 6 of the *Radio Link Frequency Assignment Problem* (RLFAP—minimization).

Performance profiles describe the evolution of the solution quality as a function of the computing time. If anytime algorithms always improve the quality of the computed solution upon time, the behavior of such algorithms, at the first time instants, is crucial. A performance profile is particularly interesting if quality of solutions augments very quickly at the beginning of the execution. Then, the improvement can decrease progressively.

To illustrate this important property of anytime algorithms, Fig. 3 compares the performance profiles of four methods, obtained on the instance 6 of the *Radio Link Frequency Assignment Problem* (RLFAP). It consists of assigning a limited number of frequencies to radio links while minimizing electro-magnetic interference constraints due to the re-use of frequencies [17].

These methods are VNS/LDS + CP, LNS/CP/GR [18], another hybrid method which also relies on solving WCSPs, and two standard versions of *Simulated-Annealing* (SA): *Quick* and *Medium*. We can note the good performance profile of our method (VNS/LDS + CP) compared with those of SA and LNS/CP/GR.

3.4. Hybrid methods

Four main categories of methods could be used to solve this problem:

1. *Systematic constructive methods*, based upon backtracking or branch and bound techniques. These methods are complete and could provide an optimal solution, but they may be very time consuming. Constraint programming can be classified in this category.
2. *Local search methods* start from an initial solution (often randomly generated). From this initial solution, exchanges between components are achieved and results are evaluated. The exchange leading

to the best solution is retained and the procedure continues until a stopping test. The exchange process depends on the method (e.g., simulated annealing [19,20], tabu search [21,22], repair-based methods [23,39]) and the neighborhood system used. The choice of a neighborhood generating mechanism should be driven by the structure of the problem.
3. *Greedy methods* construct a solution from scratch with no backtracking mechanism. These methods are obviously very fast, but they can be used only on problems with very specific properties.
4. *Hybrid methods*, can be obtained by combining constructive methods (either systematic or not) and local search methods [24–27].

Systematic constructive methods could not be used because of the on-line context and of the size of the problem. Hybrid methods [2,44] usually provide appropriate compromises between complete and local search. More precisely, they can be very efficient by combining the advantages of both *constraint propagation* and *opportunistic exploration* of the search space (local search).

VNS/LDS + CP is a hybrid method where problems are formulated as WCSP (Weighted CSP). It combines a *Variable Neighborhood Search* [3] and a *Limited Discrepancy Search* [4] with *Constraint Propagation* to efficiently guide the search. As any hybrid method, VNS/LDS + CP will always improve the quality of the computed solution upon time.

Moreover, as VNS/LDS + CP provides valuable performance profiles (see the resolution of CELAR problems [2]), it appeared to be well suited for solving our rerouting problem.

3.5. *Overview of the resolution method*

Rerouting is achieved in two main steps: computing a set of *demand conflicts* and repairing these conflicts. When a new demand d arrives, step-1 tries to get a route for d (without any rerouting). If such a route r can be found, r will be the route associated to d. If not, step-1 returns a set of demand conflicts. These demand conflicts are generated from a set of *conflicting shortest paths*, provided by an extended Dijkstra's algorithm (EDSP).

In the second step, for each *demand conflict*, a weighted CSP (WCSP), restricted to the area of the network where connections have to be rerouted, is built and then solved using VNS/LDS + CP [1,2]. WCSPS corresponding to various *demand conflicts* are successively treated. If a solution to one WCSP is found (within the time assigned to handle a demand, i.e., 1 min), the demand will be accepted. If not, the demand will be rejected.

Section 4 will be devoted to the computation of *conflicting shortest paths* (using the EDSP algorithm) and *demand conflicts*. In Section 5, we recall the definition of the valued CSP framework (VCSP) [28]. Then, we show how to formulate each *demand conflict* as a WCSP restricted to the area of the network where connections have to be rerouted. Finally, we detail how rerouting is performed using our hybrid search method.

4. Computing demand conflicts

In this section, after introducing the necessary definitions, we describe how the problem of determining a subset of the demands to be rerouted (i.e., *demand conflicts*) is formulated and solved using an extension of Dijkstra's shortest path algorithm (EDSP).

4.1. Conflicting shortest path and demand conflict

Definition 1 (*Violated link of a path for a demand*). A link l of a path p is said to be violated for a demand d iff its available (remaining) amount of bandwidth is insufficient to accept d (i.e., the capacity constraint for link l would not be verified).

Definition 2 (*Conflicting path of order k*). Conflicting paths of order k for a demand d, are paths with exactly k violated links.

w_1 and w_2 are two additive weight functions. We note w_1-weight (resp. w_2-weight) the $w_1(p)$ (resp. the $w_2(p)$) of a path p. w_1 denotes the capacity violation on a link l, whereas w_2 represents the length of a path p.

$$w_1(l) = \begin{cases} 1 & \text{if } l \text{ is a violated link,} \\ 0 & \text{otherwise.} \end{cases}$$

For a path $p = (l_1, \ldots, l_q)$, $w_1(p) = \sum_{i=1}^{q} w_1(l_i)$. Let $\mathscr{P}ath(s, t, k)$ be the set of all conflicting paths from s to t of order k, we thus have:

$$\mathscr{P}ath(s, t, k) = \{p | p \text{ path from } s \text{ to } t, \text{ and } w_1(p) = k\}.$$

Definition 3 (*Conflicting shortest path of order k*). Conflicting shortest paths of order k are paths of smallest w_2-weight, among conflicting paths of order k, $\delta(s, t, k)$ denotes the shortest length of these paths between s and t:

$$\delta(s, t, k) = \min_{p \in \mathscr{P}ath(s,t,k)} \{w_2(p)\}.$$

Definition 4 (*Conflicting shortest path problem: CSPP*). Given $G\langle \mathscr{N}, E\rangle$ a directed graph, a source node s, a destination node t, two weight functions $w_1: E \to \mathbb{N}$ and $w_2: E \to \mathbb{N}$, a constant *maxLinks* $\in \mathbb{N}$; the problem denoted *CPP*: $\langle G, s, t, w_1, w_2, \mathit{maxLinks}\rangle$ is to find a set of paths p from s to t such that $w_1(p) \leqslant \mathit{maxLinks}$ and $w_2(p)$ minimal.

maxLinks[4] is the maximal number of admitted violated links per path. For a demand d, `NotOKrouteSet` is the set of all *conflicting shortest paths* of order k (for each $k \leqslant \mathit{maxLinks}$). For each *conflicting shortest path* `NotOKroute` $\in$ `NotOKrouteSet`, a *demand conflict*, corresponding to the sub-area of the network where connections have to be rerouted, is computed.

Definition 5 (*Demand conflict*). A demand conflict for a *conflicting shortest path* *`NotOKroute`* is a quadruple $\langle d, \mathtt{NotOKroute}, \mathscr{C}_l, \mathscr{C}_c\rangle$, where:

- d is the demand,
- *`NotOKroute`* is a possible route for d,
- $\mathscr{C}_l$ is the set of violated links of *`NotOKroute`*,
- $\mathscr{C}_c$ is a set of conflicting connections we need to reroute in order to accept d.

[4] If at least one metric takes bounded integer values, the CPP problem can still be solved in a polynomial time.

4.2. Extending Dijkstra's algorithm

We have extended Dijkstra's shortest path algorithm (EDSP, Algorithm 1) in order to solve the CSPP problem defined in Section 4.1.

Algorithm 1: Extended Dijkstra's algorithm

```
EDSP($G$: Network, $d$ : Demand, $maxLinks$: Integer )
begin
    $s \leftarrow$ source ($d$)
    $t \leftarrow$ destination ($d$)
    initialize ($G$,$s$,$maxLinks$)
    $\mathcal{S} \leftarrow \emptyset$
    $\mathcal{H} \leftarrow \{\langle s,k\rangle \mid k=0\}$
1   while $\mathcal{H} \neq \emptyset$ and $\neg$finish do
2       find $\langle u,k\rangle \in \mathcal{H}$ such that $k = \min_{\langle u',k'\rangle\in\mathcal{H}}\{k'\}$ and
            $d^{(k)}[u] = \min_{\langle u',k'\rangle\in\mathcal{H}}\{d^{(k')}[u']\}$
3       $\mathcal{H} \leftarrow \mathcal{H} - \{\langle u,k\rangle\}$
4       $\mathcal{S} \leftarrow \mathcal{S} + \{\langle u,k\rangle\}$
5       if $(u = t)$ then
6           finish $\leftarrow$ ($k = 0$ or $k = maxLinks$)
        ;; the for loop iterates on different adjacent vertices v
7       foreach outgoing edge of u, $(u,v) \in E$ do
8           scan ($u$,$k$,$v$)
    if $\mathcal{P}ath(s,t,0) \neq \emptyset$ then
        update-network ($G$,$\mathcal{P}ath(s,t,0)$)
        return nil
    else
        foreach $k \in [1 \ldots maxLinks]$, $\mathcal{P}ath(s,t,k) \neq \emptyset$ do
            NotOKrouteSet    $\leftarrow$    NotOKrouteSet
            $\cup\, \mathcal{P}ath(s,t,k)$
        return NotOKrouteSet
end
```

Consider a directed graph $G = (\mathcal{N}, E)$ with number of vertices N and number of arcs M. Let w_1 and w_2 be the two weight functions defined in 4.1 ($w_1(p)$ is the number of violated links of path p and $w_2(p)$ is the length of path p. *maxLinks* is the maximal number of admitted violated links per path: for each path p, $w_1(p) \leqslant maxLinks$).

For each node $u \in \mathcal{N}$, and each integer $k \in [0 \ldots maxLinks]$, the variable $d^{(k)}[u]$ is an estimation of the smallest w_2-weight, from source node s to current node u, of conflicting paths of order k (their w_1-weight equals to k). Let us note $\delta(s, u, k)$ the shortest distance for those paths, we then have:

$$\delta(s,u,k) = \min_{p\in\mathscr{P}ath(s,u,k)} \{w_2(p)\}. \tag{2}$$

Initially, for every node u and for each $k \in [0 \ldots maxLinks]$, $d^{(k)}[u]$ is initialized to $+\infty$. For each pair $\langle u, k\rangle$, the path from s to u is stored in the variable π. $\pi^{(k)}[u]$ keeps the immediate predecessors of u on the path. The algorithm starts by setting $d^{(0)}[s]$ to 0, and $\pi^{(0)}[s]$ to -1 (see function `initialize`, Algorithm 2).

Algorithm 2: Functions initialize and scan

```
initialize($G$: Network, $s$: Integer, $x$: Integer )
begin
    foreach vertex $v \in (\mathcal{N} \setminus \{s\})$, each $k \in [0 \ldots x]$ do
        $d^{(k)}[v] \leftarrow +\infty$
        $\pi^{(k)}[v] \leftarrow$ nil
    $d^{(0)}[s] \leftarrow 0,\ \ \pi^{(0)}[s] \leftarrow -1$
end
scan($u$: Integer, $k$: Integer, $v$:Integer )
begin
    $k' \leftarrow k + w_1(u,v)$
    if $k' \leq maxLinks$ and $\langle v, k'\rangle \notin S$ then
        if $d^{(k')}[v] = d^{(k)}[u] + w_2(u,v)$ then
            $\pi^{(k')}[v] \leftarrow \pi^{(k')}[v] \cup \{u\}$
        else if $d^{(k')}[v] > d^{(k)}[u] + w_2(u,v)$ then
            $d^{(k')}[v] \leftarrow d^{(k)}[u] + w_2(u,v)$
            $\pi^{(k')}[v] \leftarrow \{u\}$
            if $\langle v, k'\rangle \notin \mathcal{H}$ then
                insert $\langle v, k'\rangle$ in $\mathcal{H}$
            else
                update $d^{(k')}[v]$ in $\mathcal{H}$
end
```

During its execution, EDSP updates estimate distances $d^{(k)}[u]$. When the algorithm ends, we have $d^{(k)}[u] = \delta(s, u, k)$, for each $u \in \mathscr{N}$, and each $k \in [0 \ldots maxLinks]$. We obtain a set of *conflicting shortest paths* p^* of order k, $k \leqslant maxLinks$. Hence, each path p^* can be retrieved by tracing π from destination t, till reaching source s.

The algorithm also maintains two additional sets, $\mathscr{S}$ et $\mathscr{H}$. $\mathscr{S}$ contains all pairs $\langle u, k\rangle$, such that $d^{(k)}[u]$ equals to $\delta(s, u, k)$ (i.e., the shortest distance of conflicting paths from s to u of order k). $\mathscr{H}$ contains all remaining pairs, such that $d^{(k)}[u]$ is greater than $\delta(s, u, k)$. Initially, $\mathscr{S}$ is set to $\emptyset$ and $\mathscr{H}$ to $\{\langle s, k\rangle \mid k = 0\}$. In the *while* loop (lines 1–8 of Algorithm 1), each iteration moves a pair $\langle u, k\rangle$ from $\mathscr{H}$ to $\mathscr{S}$ and updates estimate distances of vertices that are adjacent to u by calling `scan(u,k,v)` (see Algorithm 2). For every $(u, v) \in E$, such that $d^{(k)}[u] + w_2(u, v) \leqslant d^{(k)}[v]$, we update $d^{(k)}[v]$ and $\pi^{(k)}[v]$.

The algorithm ends when ($\mathscr{H} = \emptyset$) or ending condition of lines (5–6, Algorithm 1) is reached, and EDSP returns a set of *conflicting shortest paths* of order k, for $1 \leqslant k \leqslant maxLinks$.

4.3. Computing and ordering demand conflicts

Given *maxLinks*, computing the set of *demand conflicts* for a demand d is performed in three steps:

1. EDSP computes `NotOKrouteSet`, the set of all *conflicting shortest paths* of order k, for $k \leqslant maxLinks$;
2. Redundant *conflicting shortest paths* are removed from `NotOKrouteSet`;
3. For each remaining *conflicting shortest path* p, its associated *demand conflict* ($\langle d, \mathtt{p}, \mathscr{C}_l, \mathscr{C}_c\rangle$) is then computed:
    - $\mathscr{C}_l$ is the set of the violated links of $\mathtt{p}$ (given by EDSP).
    - $\mathscr{C}_c$ is a set of conflicting connections we accept to reroute in order to admit d. To compute $\mathscr{C}_c$, we have to determine, for each link $l \in \mathscr{C}_l$, a set of (candidate) connections that share at least one common slot of time with the calendar of d.

As a consequence, we often obtain too many connections: although some of them can be unnecessarily rerouted, we order them according to their decreasing number of violated links and their decreasing *capacity*. Connections to be rerouted are then selected according to this ordering, till all links of $\mathscr{C}_l$ becomes non-conflicting. In such a way, we prevent unnecessary rerouting of some connections, even if they share one slot of time with d.

Demand conflicts are ordered according to the two following criteria:

- their number of connections to be rerouted (we are looking for stable solutions),
- their capacity, defined as the sum of capacities of all connections to be rerouted.

Such an ordering try to manage first the easiest or the most appropriate *demand conflict*. Moreover, this ordering enables to take into account some secondary optimisation criteria like minimizing the number of rerouted connections and globally balancing the network load.

5. Rerouting using VNS/LDS + CP

Let `conflictSet` be the set of demand conflicts computed during the first step. For each demand conflict in `conflictSet`, a weighted CSP (WCSP) (restricted to the area of the network where connections may be rerouted) is built. WCSPs, corresponding to various *demand conflicts*, are successively solved by VNS/LDS + CP. If a solution for one WCSP is found within the allowed computing time, the demand is accepted. If not, the demand will be rejected.

In this section, VNS/LDS + CP [1,2] will be presented through its application to the rerouting problem. More details concerning VNS/LDS + CP and comparisons with other competing methods (e.g., local searches) on the CELAR benchmarks can be found in [2].

First, we recall the definition of the valued CSP framework (VCSP) [28]. Then, we show how to associate, to each *demand conflict*, a weighted CSP restricted to the area of the network where connections have to be rerouted. Finally, we detail how rerouting is performed using VNS/LDS + CP.

5.1. Valued constraint satisfaction problems

The Valued CSP (VCSP) [29,28] framework is an extension of the CSP (*Constraint Satisfaction Problem*) framework, which allows to deal with *over-constrained* problems or preferences between solutions.

A CSP is defined as a triplet $(\mathscr{V}, \mathscr{D}, \mathscr{C})$, where $\mathscr{V}$ is a set of *variables*, $\mathscr{D}$ a set of finite *domains* associated to the variables and $\mathscr{C}$ a set of *constraints* between the variables. A VCSP is a CSP extended with a valuation structure $\mathscr{S}$ and a valuation function φ. The valuation structure $\mathscr{S}$ is a triplet $(\mathscr{W}, \succ, \otimes)$, where $\mathscr{W}$ is a valuation set, $\succ$ a total order on $\mathscr{W}$, and $\otimes$ a binary operation closed on $\mathscr{W}$. The valuation function φ, defined from $\mathscr{C}$ to $\mathscr{W}$, associates a valuation to each constraint; the valuation of a constraint denotes its importance.

Let $\mathscr{A}$ be an assignment of all the variables and $\mathscr{C}_{\text{unsat}}(\mathscr{A})$ be the set of the constraints unsatisfied by $\mathscr{A}$. The valuation of $\mathscr{A}$ is the aggregation of the valuation of all the constraints in $\mathscr{C}_{\text{unsat}}(\mathscr{A})$: $\varphi(\mathscr{A}) = \otimes\varphi(c)$ for $c \in \mathscr{C}_{\text{unsat}}(\mathscr{A})$.

The objective is to find a complete assignment with minimum valuation. In this paper, we retained the weighted CSP sub-framework α (the aggregation operator $\otimes$ is the sum). From an algorithmic point of view, WCSPs are generally the most difficult to solve [28].

Algorithm 3: Top-level procedure

```
ATM($G$: Network, $d$: Demand, $maxLinks$: Integer, $discrep$: Integer, $size$:Integer )
begin
    ;; look for a non conflicting shortest path
    NotOKrouteSet ← EDSP ($G, d, maxLink$)
    ;; no solution exists without rerouting
    if NotOKrouteSet $\neq \emptyset$ then
        conflictSet ← build-demandConflicts ($G$,$NotOKrouteSet$,$d$)
        success ← false
        ;; for each demand conflict, build and solve the corresponding WCSP
        while conflictSet $\neq \emptyset$ and not success do
            demandConflict ← choose-one-demandConflict (conflictSet)
            ;; "try and catch" is structure of CLAIRE to handle exceptions
            try (Test (time-contract)
                pi ← build-WCSP ($G$,$demandConflict$,$d$)
                s ← VNS ($pi$,$size$,$discrep$,$Str$)
                success ← true)
            catch contradiction nil
        if success then
            ;; a solution for a WCSP has been found
            return acceptance-notification ($d$)
        else return rejection-notification ($d$)
    else
        ;; a solution without rerouting exists
        return acceptance-notification ($d$)
end
```

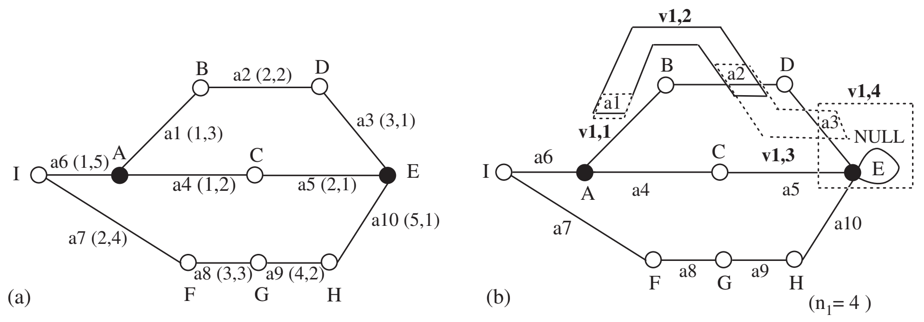

Fig. 4. An example of domains building for a demand between *A* and *E*. (a) Double labeling, (b) domains.

5.2. Building weighted CSPs

Each *demand conflict* contains a set of connections to be rerouted. Now the problem is: *given a set of connections $c_i \in \mathscr{C}_c$ to be rerouted, find a new* (*non-conflicting*) *route for each c_i*. This problem has been formulated as a WCSP [28].

5.2.1. Variables, domains and constraints

Each variable of set $\mathscr{V}$ describes which link is allocated for a rerouted connection at a certain position on its route. That is why, for each connection c_i, we introduce n_i variables, $v_{i,1}, v_{i,2}, \ldots, v_{i,n_i}$, where n_i is the maximal size of the route admitted for c_i. The maximal size n_i takes into account the size of the shortest path for c_i, and a "*freedom degree*" Δ, which expresses how much n_i can be greater than the shortest size.

The domain $\mathscr{D}_{i,j}$ of each variable $v_{i,j}$ is a set of links. First, for each rerouted connection c_i, a double labelling is computed for every link of the network, that indicates how far a link stands from source and from destination. Then, thanks to this double labelling, we can limit initial domains of variables by keeping only those links that are on correct routes, as defined below:

$$\begin{aligned}\mathscr{D}_{i,j} = \{l \in E \mid\, & j - \Delta \leqslant \mathsf{distFromSrc}(l) \leqslant j \text{ and} \\ & n_i + 1 - \Delta \leqslant \mathsf{distFromSrc}(l) + \mathsf{distFromDest}(l) \leqslant n_i + 1\}.\end{aligned}$$

A NULL value is introduced in some domains to represent routes shorter than n_i. Fig. 4 shows an example of domains building for a demand between *A* and *E*, with $\Delta = 1$.

There are two kinds of constraints in $\mathscr{C}$: *connectivity constraints*, between two consecutive variables $v_{i,j}$ and $v_{i,j+1}$ of a same connection c_i, and *capacity constraints*, over variables created for distinct connections which share at least one common link. Connectivity constraints respect the topology of the network, while capacity constraints express the satisfiability of *bandwidth requirements*.

5.2.2. Modelling as a WCSP

This CSP cannot be solved using systematic constructive methods because of the on-line context and of the size of problem (3.4). So, we have re-formulated this satisfaction problem as an optimization one in order to apply our method.

Capacity constraints will be considered as *hard* constraints. Connectivity constraints will be managed as *soft* constraints all having the same valuation. Propagation of soft constraints will allow us to prune the search tree and to determine an appropriate ordering heuristic on values (see Sections 5.3.6 and 5.3.7).

The objective is to minimize the sum (Σ) of valuations of the violated constraints; we look for solutions such that all constraints are satisfied.

5.3. *Rerouting with* VNS/LDS + CP

VNS/LDS + CP is basically a local search method (see Algorithm 4) which dynamically adjusts the neighborhood sizes, when the current solution is a *local optimum*. This choice will partially remedy to the weaknesses of pure local search methods (Simulated Annealing, Tabu Search). Indeed, the more the neighborhood is potentially large, the more there are chances that it contains good solutions and thus improves quickly the current solution. However, as neighborhoods grow larger, finding the best neighbor may require a too expensive computational effort. That is why we have selected the LDS partial search, combined with Constraint Propagation, to efficiently explore these neighborhoods.

Algorithm 4: The general VNS/LDS+CP algorithm

```
VNS ($pi$: Problem, $Size$, $discrep$: Integer, $Str$: Strategy )
begin
    $size \leftarrow Size$
    $s^* \leftarrow$ InitialAssignment ($pi$)
    $s \leftarrow s^*$
    $ub \leftarrow$ Cost ($s^*$)
    for $i \leftarrow 1$ to MAXMOVES do
        $rel \leftarrow$ Relax ($s$,$size$,$Str$)        ;; relax solution
        $s \leftarrow$ Rebuild ($s$,$rel$,$discrep$)    ;; rebuild solution
        if Cost ($s$) < Cost ($s^*$) then
            $s^* \leftarrow s$
        $size \leftarrow$ ControlNeighborhoodSize ($size$)   ;;
        adjust size
    return $s^*$
end
```

The algorithm starts with an initial complete assignment s^*; then, at each move, it *relaxes* (or unassign variables) a large part of the current solution s and then *rebuilds* it (or reassign variables) by selecting the best neighbor that strictly improves the cost of the current solution. The algorithm ends when the maximal number of local moves (MAXMOVES) has been reached.

LDS (*Limited Discrepancy Search*, see Section 5.3.5) explores the neighborhood defined by the relaxed part of the solution. It benefits from Constraint Propagation based on *lower bounds* computation, and on *static* and *dynamic* heuristics for variable and value ordering, respectively. Moreover, only judicious neighborhoods, related to *unsatisfied constraints*, are considered. Such a choice prevents LDS from modifying the value of variables appearing in satisfied constraints.

5.3.1. *Building the initial assignment*

The initial assignment used by VNS/LDS + CP, is built as follows: variables introduced for each connection are first instantiated to their previous values (i.e., links of the previous route), if they appear in their domains. Otherwise, a `NULL` value is preferred, but if it does not exist we choose randomly among values of their domains. Such a choice enables to build an initial good solution in terms of stability (by instantiating first to the previous value) and path length (by instantiating first to the `NULL` value).

5.3.2. *Relaxing variables*

Algorithm 5 describes how to select variables to relax, according to strategy *Str*. The function of line (1) computes the set ν of all currently violated constraints. To constitute *rel* (set of variables to be relaxed), constraints are randomly chosen in ν and their variables are added to *rel* until the current neighborhood size is reached ($card(rel) = size$). This choice enables a balance between choosing variables according to a specific strategy or completely at random. Experiments have shown that introducing some randomness enables the search to escape more quickly from *local minima*.

Algorithm 5: Relaxing and rebuilding a solution

```
    Relax ($s$: Solution, $size$ : Integer, $Str$ : Strategy )
    begin
1       $\nu \leftarrow$ GenerateNeighborhood ($s$,$Str$)
        $rel \leftarrow \phi$
        while $| rel | \neq size$ and $\nu \neq \emptyset$ do
2           $ct \leftarrow$ RandomElement ($\nu$)
            $rel \leftarrow rel \cup \{vars(ct)\}$
            $\nu \leftarrow \nu \setminus \{ct\}$
        return $rel$
    end
    Rebuild ($s$ : Solution, $rel$ : VarSet, $discrep$ : Integer )
    begin
        if $| rel | = 0$ then
            if Cost ($s$) $\prec ub$ then
                $ub \leftarrow$ Cost ($s$)
                BestSolution $\leftarrow s$
        else
            $x_i \leftarrow$ ChooseVariables ($rel$)
            $D_{x_i} \leftarrow$ SortDomain ($x_i$)
            $k \leftarrow 0$
            for $j \leftarrow 1$ à $DomainSize(x_i)$ and $k \leq discrep$ do
                $s[i] \leftarrow D_{x_i}[j]$
                $lb \leftarrow$ LB($s$,$x_i$)
                if $lb \prec ub$ then
                    Rebuild ($s, rel \setminus \{x_i\}, discrep - k$)
                $k \leftarrow k + 1$
        return BestSolution
    end
```

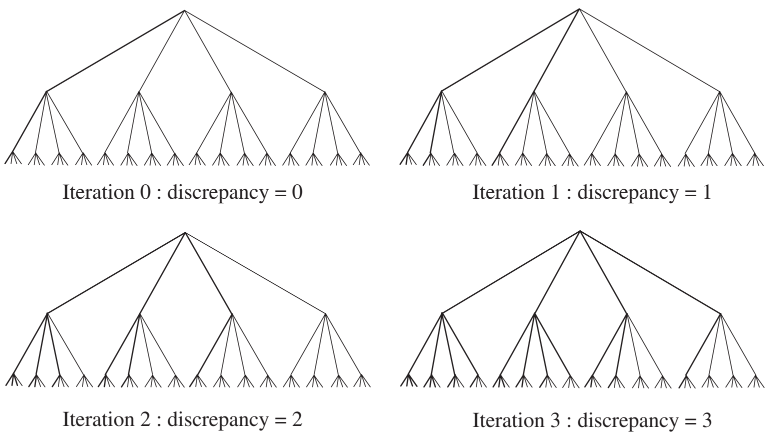
Iteration 0 : discrepancy = 0
Iteration 1 : discrepancy = 1
Iteration 2 : discrepancy = 2
Iteration 3 : discrepancy = 3

Fig. 5. Principle of LDS on n-ary tree.

5.3.3. *Control of the neighborhood size*

Initially, the neighborhood size (*size*) takes a *minimum value*. To control the neighborhood size, different strategies have been implemented. The best strategy we have found, increases systematically *size* by one, each time the method does not improve the cost of the current solution.

5.3.4. *Rebuilding a solution*

Algorithm 5 defines the function `Rebuild`. The global variables *ub* and *lb* record the upper and lower bounds of the problem optimum. `ChooseVariable` and `SortDomain` define respectively *static* ordering for variables and *dynamic* ordering for values. `LB`(s, x_i) computes a lower bound of the subproblem, limited to the variables after i. *discrep* sets the maximal number of choices that we can diverge from the heuristic (*discrepancies*).

5.3.5. *Limited discrepancy search*

The principle of LDS [4] is to explore heuristically good solutions that might be at a limited distance from a *greedy solution*. LDS ensures a more balanced exploration of the search tree, and speeds up the reconstruction step. LDS starts from the solution computed by the value heuristic, and successively explores solutions by increasing the number of discrepancies, until the fixed maximal number of discrepancies is reached.

We have used a generalized version of LDS on n-ary trees. Fig. 5 shows the paths (in bold) explored by our version of LDS. The discrepancy is measured according to the rank of the value chosen for a variable with respect to the order given by the heuristic on values. We count a single discrepancy for the *second cheapest* value of one variable, two discrepancies for either the *third cheapest* value of one variable, or the second cheapest values of two variables, and so on. We only perform *one iteration* of LDS, with a fixed number of discrepancies. This prevents to re-visit leaf nodes.

5.3.6. *Constraint propagation*

One of the main strengths of our approach lies in the use of Constraint Propagation to prune useless sub-trees, but also to guide the choice of values during the reconstruction, while keeping this step fast enough. At the beginning, we make the connectivity constraints *arc-consistent*. During the resolution, at each node of the search tree, *lower bounds* are computed in order to locally exclude all partial solutions which cannot lead to complete assignments of better valuation than the current best solution. These lower bounds are computed with *Partial Forward Checking + Directed Arc Consistency* (PFC–DAC) algorithm [30]. Only soft constraints are taken into account for computing lower bounds.

Because of the time dimension (35,000 slots of time), verifying capacity constraints is much more expensive than lower bounds computation on connectivity constraints. So, we have retained a weaker consistency for capacity constraints. As a connection cannot use a link twice, the result of capacity test on a variable $v_{i,j}$ does not depend on instantiated variables $v_{i,k}$, $k < j$, but only on instantiated variables $v_{l,h}, l \neq i$, of other connections. So, if the capacity constraint fails when $v_{i,j}$ is instantiated to a, it is removed from all the domains of uninstantiated variables $v_{i,k}$ $(k > j)$. To prevent redundant search a is also removed from the domains of instantiated variables $v_{i,l}$ $(l < j)$. Deleted values are only restored when backtracking is performed to a variable of an older connection.

5.3.7. Heuristics

Our variable ordering first takes variables created for the *connection of the largest capacity*, breaking ties by the largest connection in number of variables. To yield a profit from the connectivity constraints structure, we select variables of each connection from destination to source.

Constraint propagation, based on PFC–DAC algorithm, allows us to use a *dynamic minimum inconsistency count* value ordering. During search, for each value, so-called *Inconsistency Counts* (*ic*) and *Directed Arc Inconsistency counts* (*dac*) which memorize the look-ahead effects of an assignment over the uninstantiated variables, are computed. Variables are instantiated first to the `NULL` value (we prefer shortest paths). Then values are selected by their increasing *ic* + *dac*, breaking ties by the *least loaded link*.

6. Study of an example

6.1. Description

To illustrate our method, we present the resolution of a small subproblem. Time dimension is restricted to three slots of time, and we note (b_1, b_2, b_3) a calendar, where b_1, b_2 and b_3 are the bandwidth capacities required for each slot. For each demand, we only consider *conflicting shortest paths* with at most one violated link ($maxLinks = 1$). Moreover, when rerouting occurs, we only build routes whose size is equal to the shortest size, or routes having one additional link ($\Delta = 1$). For simplicity reasons, we use an undirected graph. Three connections have already been allocated in the network, as described in Fig. 6a.

- c_1, between *A* and *E*, has for calendar (0, 5, 5), and for route (a_4, a_5).
- c_2, between *I* and *E*, has for calendar (0, 4, 0), and for route (a_6, a_4, a_5).
- c_3, between *A* and *D*, has for calendar (5, 0, 0), and for route (a_1, a_2).

Each link has for label its name (a_i), its capacity, and its current reservations (a matrix with a line per connection and a column per slot of time).

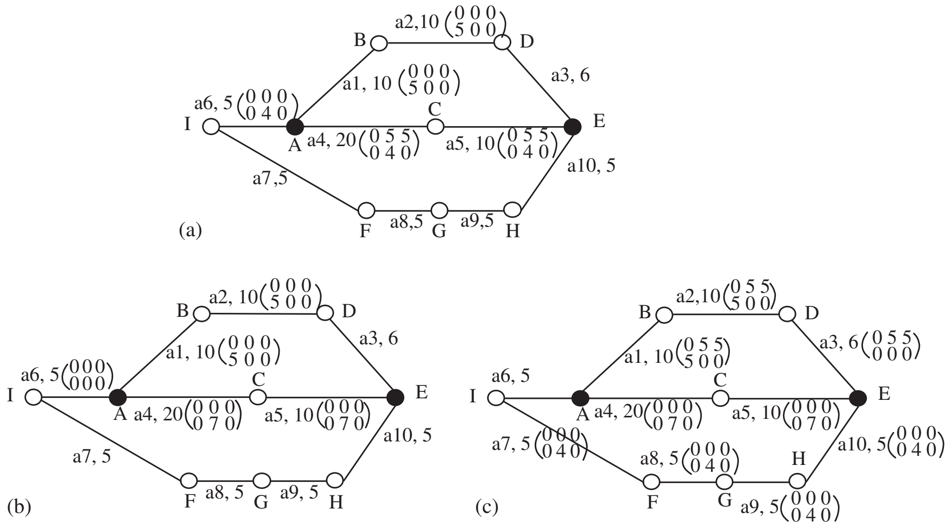

Fig. 6. States of the network. (a) Initial state of the network, (b) for a conflict Cf1, (c) final network.

6.2. *Arrival of a new demand*

Let us suppose that a new demand requires a connection between A and E, with a calendar $(0, 7, 0)$. This demand cannot be accepted without rerouting...

6.3. *Selecting demand conflicts*

The only *conflicting shortest path*, computed by EDSP, is $r_1 = (a_4, a_5)$. No other conflicting shortest path exists for this demand, since links a_3, a_6 cannot support the required capacity of the second slot (i.e., capacity b_2 of the calendar). The only violated link on route r_1 is a_5: it supports connections c_1 and c_2. Calendar of both connections are in conflict with the demand, so, we have one *demand conflict*: cf_1, composed of the route r_1, violated link a_5, and conflicting connections c_1 and c_2.

6.4. *Rerouting with* VNS/LDS + CP

To solve the *demand conflict* cf_1, we first remove connections c_1 and c_2, and consider that the demand is accepted on the route r_1. The associated network is depicted in Fig. 6b.

By taking into account the availability of links, we find a shortest route of three links for c_1. As *freedom degree* is one, the maximal size of the route admitted for c_1 is four. So, we introduce four variables: $v_{1,1}$, $v_{1,2}$, $v_{1,3}$ and $v_{1,4}$ for c_1. c_2 has got a shortest route of four links. As *freedom degree* is one, the maximal size of the route admitted for c_2 is five. So, we introduce five variables: $v_{2,1}$, $v_{2,2}$, $v_{2,3}$, $v_{2,4}$ and $v_{2,5}$ for c_2. The cost variable PENALTY records the sum of valuations of the violated constraints.

Double labelling is computed for each link, and according to it, the link may belong to a domain or not. We have detailed domains building for the connection c_1 in Fig. 4. Finally, variables and their initial

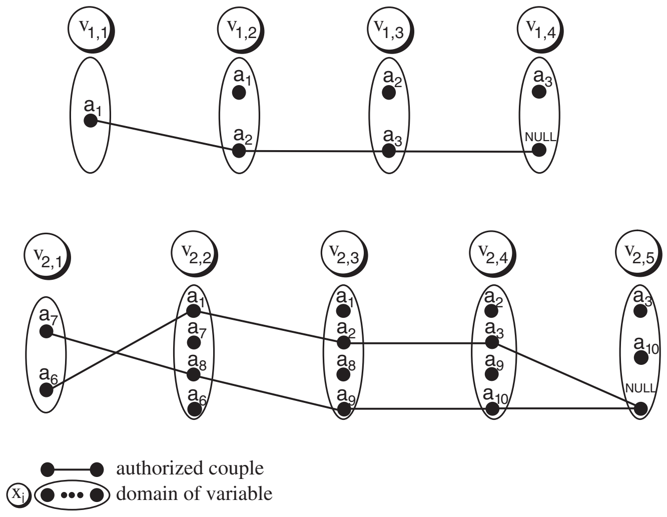

Fig. 7. Connectivity constraints for the conflict.

Table 1
Solution of the WCSP

$v_{1,4}$: NULL,	$v_{2,5}$: NULL,	PENALTY : 0
$v_{1,3}$: a_3,	$v_{2,4}$: a_{10},	
$v_{1,2}$: a_2,	$v_{2,3}$: a_9,	
$v_{1,1}$: a_1,	$v_{2,2}$: a_8,	
	$v_{2,1}$: a_7,	

domains are

$$v_{1,1} : \{a_1\};\ v_{1,2} : \{a_1, a_2\};\ v_{1,3} : \{a_2, a_3\};\ v_{1,4} : \{\mathtt{NULL}, a_3\};\ v_{2,1} : \{a_6, a_7\};\ v_{2,2} : \{a_1, a_6, a_7, a_8\};$$
$$v_{2,3} : \{a_1, a_2, a_8, a_9\};\ v_{2,4} : \{a_2, a_3, a_9, a_{10}\};\ v_{2,5} : \{\mathtt{NULL}, a_3, a_{10}\};\ \mathtt{PENALTY} : [0, 1].$$

There are two types of constraints: connectivity constraints, that are binary constraints, and capacity constraints, with one global constraint per link. A preliminary arc-consistency step on connectivity constraints (see Fig. 7) removes values that have no support in a domain. It removes, a_1 from the domain of $v_{1,2}$, a_2 from the domain of $v_{1,3}$, and a_3 from the domain of $v_{1,4}\ldots$

Finally, ordered variables and their new domains are

$$v_{1,4} : \{\mathtt{NULL}\};\ v_{1,3} : \{a_3\};\ v_{1,2} : \{a_2\};\ v_{1,1} : \{a_1\};\ v_{2,5} : \{\mathtt{NULL}\};\ v_{2,4} : \{a_3, a_{10}\};\ v_{2,3} : \{a_2, a_9\};$$
$$v_{2,2} : \{a_1, a_8\};\ v_{2,1} : \{a_6, a_7\};\ \mathtt{PENALTY} : [0, 1].$$

The resolution of the WCSP by VNS/LDS + CP allows us to obtain the solution depicted in Table 1. The final network is illustrated in Fig. 6c.

7. Related works

Several QoS routing algorithms have been published recently [31–35]. But, none of them performs rerouting.

Routing problems with multiple QoS constraints are extensively reviewed by Chen and Nahrstedt in [36]. The authors give a classification of these problems, discuss their time complexities and compare some algorithms (for an overview of these algorithms see [36]). However the time complexity of QoS routing algorithms strongly depends on the types of involved metrics, and are unfortunately NP-complete [9] in the case of two independent additive metrics.

Christien Frei [37] has formulated the problem of Resource Allocation in Networks (RAIN) as a CSP, where variables are demands; the domain of each variable is the set of all routes between the end-points of the demand, and constraints on each link ensure that the resource capacity is not exceeded by the demands routed through it. To compute and represent domains of variables, he introduced the Blocking Island abstraction (hierarchy of bandwidth availabilities), in order to create a compact representation of all possible routes satisfying a demand. All demands are known prior to the resolution, and are allocated *off-line*. The RAIN problem is close to a routing problem, but Frei's approach is not applicable to our problem for two reasons: first, our planning must be performed on-line; second, due to the time dimension, it would be unrealistic to use the Blocking Island paradigm over 35,000 slots of time.

Muriel Lauvergne [12], has proposed a first approach for resource allocation in ATM networks. Her approach mixes shortest path algorithms, Constraint Propagation and repairing principles [38] and can be summarized as follows: a preliminary step is achieved with a shortest path algorithm in order to compute a route for the connection demand. If such a route does not exist, reasons of failure are identified and constitute a set of *demand conflicts*. Successively, for each *demand conflict*, a RCSP (restricted to the area of the network that can be modified) is built and then solved as a Constraint Satisfaction Problem by a backtracking algorithm. Each resolution of a RCSP is limited by an *inner timeout*: the whole computing time (1 mn) is arbitrarily divided into k time slots of $60/k$ s, corresponding to the maximal duration allocated for the resolution of an RCSP. RCSPs are solved using chronological backtracking with Constraint Propagation. If an RCSP can be solved in the allowed computing time, the demand will be accepted. If not, the demand will be rejected.

This approach has two major drawbacks:

- Only shortest conflicting paths of order ($k = 1$) are considered to build demand conflicts. This choice strongly limits the possibilities to perform a successful rerouting. This noticing has motivated the development of our EDSP algorithm which computes the set of shortest conflicting paths of order k, ($1 \leqslant k \leqslant maxLinks$) (see Section 4). This greatly enlarges the sub-area of the network that can be repaired by considering more relevant *demand conflicts*.
- Second, solving an RCSP by a backtracking algorithm within an arbitrary inner timeout, is less adapted than a method conceived for solving *anytime problems*. In [12], 20% of the resolution of *demand conflicts* are interrupted by inner timeouts. So, using a constructive method like backtracking in an *anytime* context seems to be inappropriate (see Section 3.4): the resolution of a *demand conflict* may be interrupted even though it would lead quickly to a complete solution (at least, in less than the amount of remaining computing time). Moreover, work performed to obtain partial solutions is systematically lost upon backtracking.

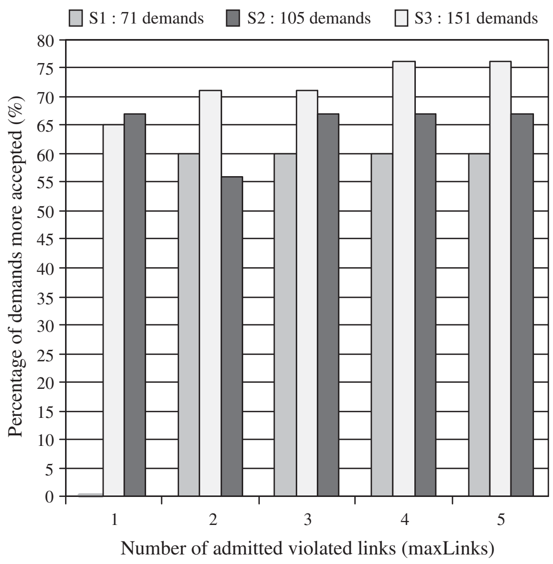

Fig. 8. Influence of *maxLinks* on the percentage of demands more accepted.

8. Experimental results

France Telecom R&D provided us with data for real-life ATM networks of 31 nodes, 134 links and a set of 60 demands. Few series of benchmarks have been built. In this paper, we study three series S_1, S_2 and S_3 of respective sizes 71, 105 and 151. For each series, we give the average results over 10 runs for VNS/LDS + CP. For each connection demand, we allow 1 min of computing time.

VNS/LDS + CP parameters are: the maximal number of local moves (MAXMOVES), the initial neighborhood size (*size*), and the number of discrepancies (*discrep*). For all experiments, MAXMOVES has been set to 50, *size* to 3, and *discrep* to 4 (these are the best values found for *size* and *discrep*). We carried out experiments with different values of *maxLinks*: {1, 2, 3, 4, 5}, and *freedom degree* (Δ): {0, 1, 2}.

8.1. Influence of maxLinks

maxLinks is the maximal number of violated links per conflicting path computed by the EDSP algorithm. For this experiment, we fixed the *freedom degree* to one. Fig. 8 indicates, as expected, that for low values of *maxLinks* results are worse: the sub-area of the network where rerouting can be performed is too small. (*maxLinks* = 4) and (maxLinks = 5) give the best results.

These results are similar because the average length of routes does not exceed 5. So, for the rest of our experiments, *maxLinks* will be set to 5.

8.2. Influence of the freedom degree

The degree of freedom enables to accept connections on routes that are longer than shortest paths. As depicted in Fig. 9, $\Delta = 1$ gives the best results. In fact, for higher values of Δ, demands are allocated on

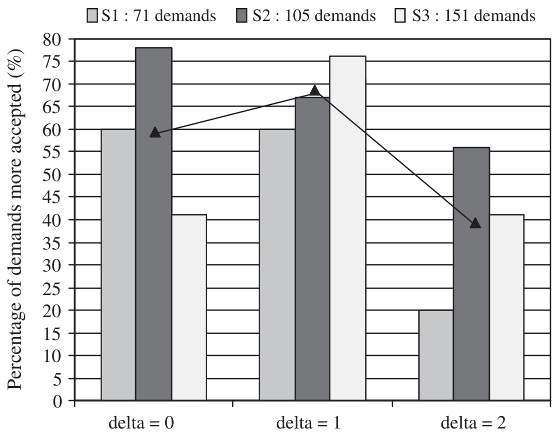

Fig. 9. Influence of freedom degree on the percentage of demands more accepted.

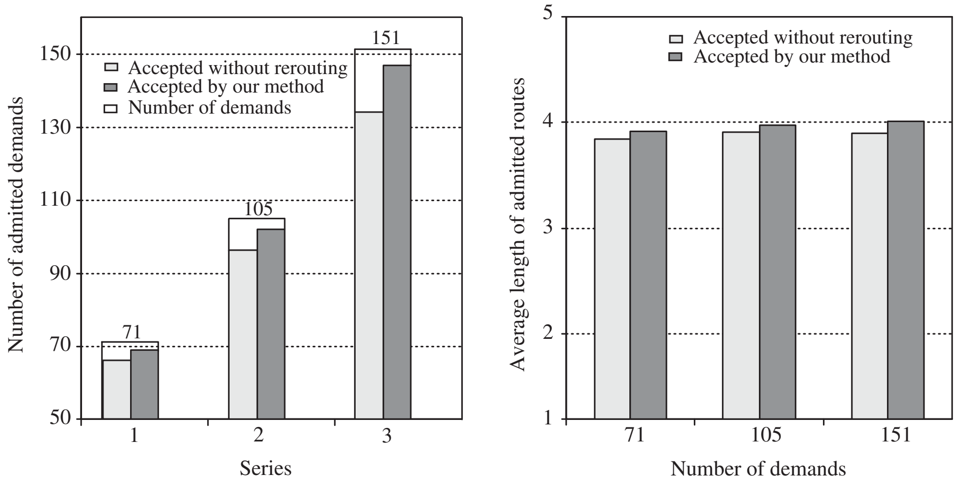

Fig. 10. Comparing the number of accepted demands and the average length of routes.

too long routes and the network will then reach stalemate very quickly since links will be over-loaded. Good results are also obtained for $\Delta = 0$. In that case, rerouting provides shortest routes.

8.3. Comparisons and discussion

Fig. 10 (left) compares the number of accepted demands with and without rerouting. The method without rerouting is a classical shortest path (obtained by setting *maxLinks* to 0 in the EDSP algorithm). Without rerouting, the number of rejected demands rapidly augments as the number of demands to be planned increases. Fig. 11 shows that rerouting significantly improves the amount of accepted demands.

Fig. 10 (right) compares both methods in terms of network resource utilization, i.e., lengths of routes for accepted demands. The overhead of rerouting is negligible from this point of view. This important

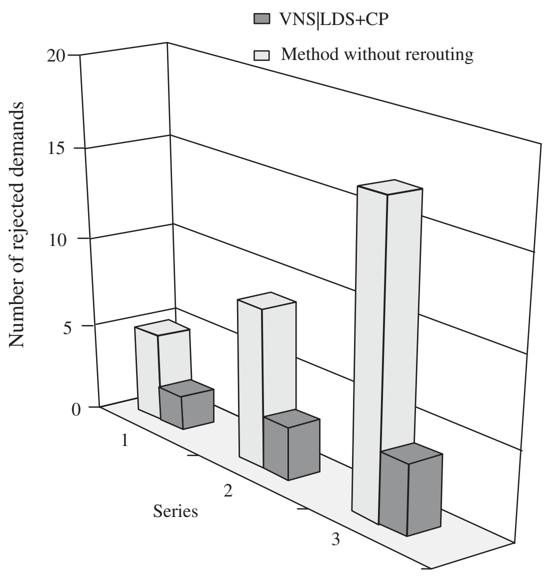

Fig. 11. Comparing the number of rejected demands.

property is due to the fact that, when we have to reroute some already established connection, we only accept routes whose length is very close to that of shortest paths ($\Delta = 1$).

This real-life application demonstrates the interest of VNS/LDS + CP for on-line rerouting. For this application, rerouting with VNS/LDS + CP enables to accept an average of 67% of demands that would be rejected by a greedy algorithm (see Fig. 11).

9. Conclusions and further works

We have used a Constraint based approach to plan connection demands in ATM networks. We have proposed an extended version of Dijkstra's algorithm to find shortest routes that minimize capacity violations on links. We have therefore modelled rerouting as a constraint optimization problem formulated as a WCSP solved using VNS/LDS + CP. First experiments have shown the efficiency of our approach since the number of accepted demands increases significantly compared to methods only based on shortest paths algorithms.

In the future work, we intend to integrate and study benefits of *recursive rerouting*. Suppose that, in order to accept a new demand d, we need to reroute a connection c_1. Moreover, suppose that c_1 can be rerouted if and only if another connection c_2 has to be itself rerouted. As rerouting is actually performed, it would fail in such a situation. We expect that *recursive rerouting* would lead to accept more demands. But the depth of recursive calls to rerouting will have to be managed very carefully taking into account our specific context of *on-line* planning.

Acknowledgements

We thank Remi Douence for all improvements he made to this paper.

References

[1] Loudni S, Boizumault P. A new hybrid method for solving constraint optimization problems in anytime contexts. In: Proceedings of the thirteenth IEEE international conference on tools with artificial intelligence (ICTAI'2001), vol. 1417. Dallas, USA, November 2001. IEEE Computer Society. p. 325–32.

[2] Loudni S, Boizumault P. Solving constraint optimization problems in anytime contexts. In: Gottlob G, Walsh T. editors. IJCAI-03: Proceedings of the 18th international joint conference on artificial intelligence, Acapulco, Mexico, August 2003. Los Altas, CA: Morgan Kaufmann. p. 251–6.

[3] Hansen P, Mladenovic N. Variable neighborhood search. Computers And Operations Research 1997;24:1097–100.

[4] William D Harvey, Matthew L Ginsberg. Limited discrepancy search. In: Mellish C, editor. IJCAI'95: Proceedings of the international joint conference, Montreal, August 1995.

[5] Loudni S. Conception et mise en œuvre d'algorithmes anytime: une approche à base de contraintes. PhD Thesis, École des Mines de Nantes, France, December 2002.

[6] de Givry S, Hamadi Y, Mattioli J, Lemaitre M, Verfaillie G, Aggun G, Gouachi I, Benoist T, Bourreau E, Laburthe F, Loudni S, David P, Bourgault S. Towards an on-line optimisation framework. In: OLCP 2001, international workshop on on-line combinatorial problem solving and constraint programming (at CP-2001), Paphos, Cyprus, 2001.

[7] Friesen VJ, Harms JJ, Wong JW. Resource management with virtual paths in ATM networks. IEEE Network 1996; 10–20.

[8] Ma Q. Quality-of-service routing in integrated services networks. PhD Thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, USA, January 1998.

[9] Wong Z, Crowcroft J. Quality of service routing for supporting multimedia applications. IEEE Journal on Selected Area in Communications 1996;17(7):1228–34.

[10] ITU-T: International Telecommunication Union. Traffic control and congestion controle in b-isdn. 2000.

[11] Brichet F, Roberts J, Simonian A, Gravey A, Sevilla K. Le contrôle d'admission des connexions pour le service SBR. Technical Report, Ft.bd/cnet/dac/ntr/n-5031, FT.BD, 1997.

[12] Lauvergne M, David P, Boizumault P. Connections reservation with rerouting for ATM networks: a hybrid approach with constraints. In: Principles and practice of constraint programming (CP-02), vol. 1713, Lectures notes in computer science. Ithaca, NY, Berlin: Springer. September 2002.

[13] ATM forum. Private Network Network Interface (PNNI). v1.0. 1996.

[14] Bryant RE. Graph-based algorithms for boolean function manipulation. IEEE, Transaction on Computers 1986;C-35(8): 677–91.

[15] Boddy M, Dean TL. Deliberation scheduling for problem solving in time-constrained environments. Artificial Intelligence 1994;67:245–95.

[16] Zilberstein S. Using anytime algorithms in intelligent systems. AI Magazine 1996;17(3):73–83.

[17] Cabon B, de Givry S, Lobjois L, Schiex T, Warners JP. W.K. Radio link frequency assignment. Constraints: An International Journal 1999;4(1):79–89.

[18] Lobjois L, Lemaître M, Verfaillie G. Large neighbourhood search using constraint propagation and greedy reconstruction for valued csp resolution. In: Proceedings of the ECAI2000 workshop on modelling and solving problems with constraints. 2000.

[19] Kirkpatrick S, Gelatt CD, Vecchi PM. Optimization by simulated annealing. Science 1983;220:671–80.

[20] Kirkpatrick S. Optimization by simulated annealing: quantitative studies. Journal of Statistical Physics 1984;34(5).

[21] Glover F, Laguna M. Modern heuristic techniques for combinatorial problems, Chapter Tabu Search. Oxford: Blackwell Scientific Publications; 1993. p. 70–141.

[22] Glover F, Laguna M. Tabu Search. Dordrecht: Kluwer Academic Publishers; 1997.

[23] Minton S, Johnston M, Philips A, Laird P. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. Artificial Intelligence 1992;58:161–205.

[24] Pesant G, Gendreau M. A view of local search in constraint programming. In: Proceeding of the second international conference on principles and practice of constraint programming (CP-96). Cambridge, USA; 1996. Berlin: Springer. p. 353–66.

[25] Apt A, Schaerf A. Search and imperative programming. In: Mellish C. editor. POLP-97: Proceedings of 24th annual SIGPLAN-SIGACT symposium on principles of programming languages. 1997. p. 67–79.

[26] Prestwich S. A hybrid search architecture applied to hard random 3-sat and low-autocorrelation binary sequences. In: Principles and practice of constraint programming (CP 2000), vol. 1894. Lecture notes in computer science, Singapore, September 2000. Springer. p. 337–52.
[27] Laburthe F, Caseau Y. SALSA, a language for search algorithms. In: CP-98, Lecture notes in computer science, vol. 1520. Berlin: Springer; 1998. p. 310–24.
[28] Schiex T, Fargier H, Verfaillie G. Valued constraint satisfaction problems: hard and easy problems. In: Mellish C. editor. IJCAI'95: Proceedings of the international joint conference on artificial intelligence, Montreal, August 1995.
[29] Bistarelli S, Montanari U, Rossi F, Schiex T, Verfaillie G, Fargier H. Semiring-based csps and valued csps: frameworks, properties, and comparison. Constraints: An International Journal 1999;4(3):199–240.
[30] Larrosa J, Meseguer P, Schiex T. Maintaining reversible dac for solving max-csp. Artificial Intelligence 1999;107(1): 149–63.
[31] de Neve H, van Mieghem P. A multiple quality of service routing algorithm for PNNI. In: ATM workshop 1998. p. 324–8.
[32] Hwang R-H, Chen M-X, Hsu C-M. Routing in ATM networks with multiple classes of QoS. In: Global telecommunications conference, GLOBECOM 2000, vol. 3. 2000. p. 1756–60.
[33] Liu J, Niu Z, Zheng J. A QoS routing algorithm for hierarchical ATM networks. In: APCC/OECC'99, vol. 1. 1999. p. 188–91.
[34] Vasilakos A, Saltouros MP, Atlassis AF, Pedrycs W. Optimizing QoS routing in hierarchical ATM networks using computational intelligent techniques. IEEE Transactions On Systems, Man and Cybernetics, Part C 2003;33(2): 297–312.
[35] Lee W, Hluchyi M, Humblet P. Routing subject to quality of service constraint in integrated communication networks, IEEE Network, July 1995, pp. 46–55.
[36] Chen S, Nahrstedt K. An overview of quality of service routing for the next generation high-speed networks: problems and solutions. IEEE Network Magazine 1998; 67–74.
[37] Frei C, Faltings B. Resource allocation in networks using abstraction and constraint satisfaction techniques. In: Principles and practice of constraint programming (CP-99), vol. 1713, Lecture notes in computer science. Berlin; Springer. 1999. p. 204–18.
[38] Lauvergne M. Réservation de Connexions Avec Reroutage Pour les Réseaux ATM, Une Approche Hybride par Programmation par Contraintes. PhD Thesis, École des Mines de Nantes, France, March 2002.
[39] Boizumault Patrice, David Philippe, Djellab Housni Resource allocation in a mobile telephone network: a constructive repair algorithm. RAIRO Operations Research 2001;35(2):189–209.
[44] Laburthe F. Choco: implementing a cp kernel. In: CP00 post conference workshop on techniques for implementing constraint programming systems (TRICS), Singapore, September 2000.

Further Reading

[40] Caseau Y, Josset F-X, Laburthe F. Claire: combining sets, search and rules to better express algorithms. In: De Schreye D, editor. Proceeding of the 15th international conference on logic programming, ICLP'99. Cambridge, MA: MIT Press; 1999. p. 245–590.
[41] Pesant G, Gendreau M. A constraint programming framework for local search methods. Journal of Heuristics 1999. p. 1–25.
[42] Shaw P. Using constraint programming and local search methods to solve vehicle routing problems. In: Proceedings of the international conference on principles and practice of constraint programming (CP-98), Pisa, Italy, 1998. p. 417–31.
[43] Focacci F, Laburthe F, Lodi A. Local search and constraint programming. In: Glover F, Kochenberger G, editors. Handbook on metaheuristics. Dordrecht: Kluwer Academic Publishers, 2004. p. 1–31.