Ecole Polytechnique
Promotion X99
91128 Palaiseau cedex
FRANCE

ZIB
Konrad-Zuse-Zentrum für
Informationstechnik Berlin
Takusstrasse 7,
14195 Berlin-Dahlem
GERMANY

RAPPORT DE STAGE D'OPTION SCIENTIFIQUE

Combinatorical heuristics for the design of survivable network

NON CONFIDENTIEL

Nicolas Omont

Option :	Mathématiques appliquées
Champ de l'option :	Réseaux de télécommunication
Directeur de l'option :	François Baccelli
Directeur du stage :	Roland Wessäly
Dates du stage :	15/04/02 - 30/06/02

ABSTRACT

Failing components of a network may induce serious damages. These may have different causes as natural disasters or cables being torn during construction works, etc. A substantial amount of research has been carried out to allow the design of cost-effective and survivable networks.

The purpose of this paper is the following. Given are a set of communication nodes, a set of potentially installed links and communication demands. The task is to design topology that allows to satisfy all the demands and that also fulfill requirements concerning the survivability of the network at the lowest cost possible.

First, it is important to have a model simple enough to be able to implement efficient computation, complex enough to give usable results and flexible enough to accommodate many different situations. The model described is derived from the MSI model (Module-Slot-Interface). It allows the use of the MSI framework of DISCNET, a tool for dimensioning networks developed by Atesio in cooperation with the ZIB.

Then, solutions to this network design problem can, for example, be found using cutting-plane algorithms , based on a linear relaxation of the problem. This is DISCNET main approach and generally gives the best results The work described in this paper aims at developing combinatorial procedures to compute solutions.

They eventually enable a fast computing of rather good solutions (in comparison with another technique) and may constitute a basis to develop other combinatorial algorithms as regards this problem.

Key-words: survivable networks, MSI model, combinatorial heuristics

RESUME

Les défaillances de certaines composantes d'un réseau de télécommunications peuvent avoir de graves répercussions sur son fonctionnement. Celles-ci ont des origines diverses comme des catastrophes naturelles ou la coupure de câbles lors de travaux, par exemple. Face à de telles éventualités diverses recherches ont été portées sur la conception de réseaux dits fiables.

L'enoncé du problème particulier ici traité est le suivant. Etant donné un ensemble de noeuds, de liens et de demandes à satisfaire, il s'agit de déterminer la topologie du réseau de coût minimum permettant de satisfaire toutes les demandes mais également d'autres contraintes concernant les cas de défaillance de l'une de ses composantes au coût le plus bas.

Tout d'abord, il est important de disposer d'un modèle Ensuite, face à ce problème ont été employés des procédés basés sur des techniques issues de la programmation linéaire. C'est l'approche privilégiée de DISCNET et c'est celle qui donne de manière générale les meilleurs résultats.. Le travail réalisé a consisté à développer une autre approche pour sa résolution, reposant sur des heuristiques combinatoires.

Elles constituent ainsi une base intéressante pour obtenir rapidement une solution, et divers développements peuvent leurs encore leur être apportés.

Mots clés : réseaux de communications, modèle MSI, heuristiques combinatoires.

Contents

1 Introduction

In our economy, network operators want to satisfy communication needs at the lowest cost possible. On the one hand, hardware progress makes increasing capacity available at lowest cost, on the other hand, the increasing complexity of networks makes it too difficult to conceive an efficient network "with pencil and paper". Therefore, there is a need of computer assisted network dimensioning. Whatever the algorithms used, this approach requires an insight into the real problem in order to modelise it so that the solution found is really pertinent to build the real network. That is why this introduction is dedicated to the problem, the second part deals with modelisation aspects, then come an algorithm to solve the problem modelised. Finally, the given results close the loop as they are solutions usable for the initial problem.

1.1 Demand

A communication network is not only characterised by its transmission capacities, but also by the quality of service it provides. In a word, users do not only want to send data from one point to another, they also want a guarantee on the "speed" of the connection whatever the circumstances (failures, overload...).

With this observation, what are the interesting parameters for dimensioning such a network:

- Capacity: It is obvious that text, voice or video transmission require different flow rate and network capacity.
- Survivability: A survivable network is a network for which failure cases have been anticipated so that transmissions are still possible in such states.
- Transmission delay: For some traffic, like voice, the data transmission delay is an important matter. For instance, conversation is disturbed when this delay is above 50ms.

We define a demand as a transmission need between two points under capacity, survivability and transmission delay constraints. We have to modelise such a demand. For capacity, we just keep the flow rate needed. Then, many survivability models have been developed. We only keep the diversification model. It says that for each demand, no more than a given fraction of the flow can be routed through one component of the network. We call this fraction the diversification parameter of the demand. For transmission delay, we consider that most of the transmission time occurs in switches, so that we want to limit the number of switches a flow can cross. Therefore, we have three parameters for each demand: capacity required, diversification parameter, and length restriction parameter.

For instance, on figure 1, a demand between node 1 and 7 is represented by the dotted line. Its parameters are a required capacity of 35, a diversification parameter of 3/7, and a length restriction of 3. The continuous lines represent links between nodes, with their associated capacity.

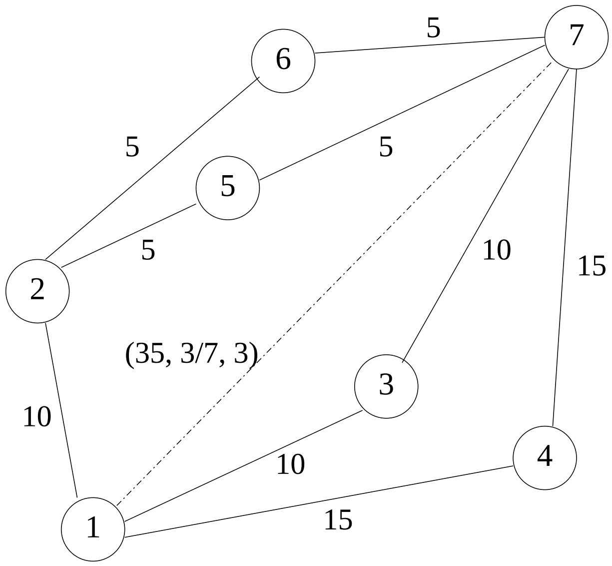

Figure 1: Simple network

1.2 Hardware

To satisfy these demands at the lowest possible cost, i.e. to optimise the network, operators uses hardware like switches and communication links. Would it be possible to build switches and links of any capacity, optimisation would be easy. However, capacities have a discrete structure, because the hardware produced is only available for a limited numbers of capacities.

Therefore, we have to look carefully at the hardware to modelise it in a way that our results clearly tell what hardware should be install where.

We will study Siemens hardware to build this model called Module-Switch-Interface (MSI), but it is far more general. The information are taken out from [1].

- First of all, the links. There are four different types of capacity configurations. In absence of other constraints, it is possible to install as many configurations as we want on one link.

Configurations	Capacity
STM1	1 STM1
STM4	4 STM1
STM16	16 STM1
ADM16	16 STM1

Table 1: Link capacities

- Then, the switch. For instance we consider Siemens SLD16:

Configurations	Modules	Cost	Total capacity		
			modules	switch	slot
SLD16	4xSTM1, 1xSTM4, 1xSTM16, 1xADM16	2,49	64 STM1	64 STM1	8

Table 2: Node type parameters

This switch has a total switching capacity, which is expressed in multiple of the smallest link capacity STM1. It means that, even if it is physically possible to plug a total link capacity above 64 STM1, it is impossible to route more than 64 STM1 through this switch.

Then, links are not directly plugged into the switch. Modules are found between them. They are cards that hold on one side plugs for one type of edge capacity, which we call interfaces, and on the other side hardware to be plugged in slots of the switch. Therefore, the switch is limited by its number of slots (8), and the total capacity of modules that may be plugged into it (64 STM1, here equal to the switching capacity).

- Finally, the table gives the characteristics of modules used, i.e. the interfaces they provide, their capacity (which is deduced from the link capacity table), and the number of slots they require.

Module	Interfaces			Module capacity	Slots	Cost
	STM1	STM4	STM16			
4xSTM1	4	0	0	4 STM1	1	120 TDM
1xSTM4	0	1	0	4 STM1	1	120 TDM
1xSTM16	0	0	1	16 STM1	1	160 TDM
1xADM16	0	0	1	16 STM1	0	140 TDM

Table 3: Modules parameters

To make it clearer, figure 2 is the representation of one SLD16 with modules and interfaces. One must keep in mind that it is possible to have many SLD16 connected together so that we can consider them as one switch.

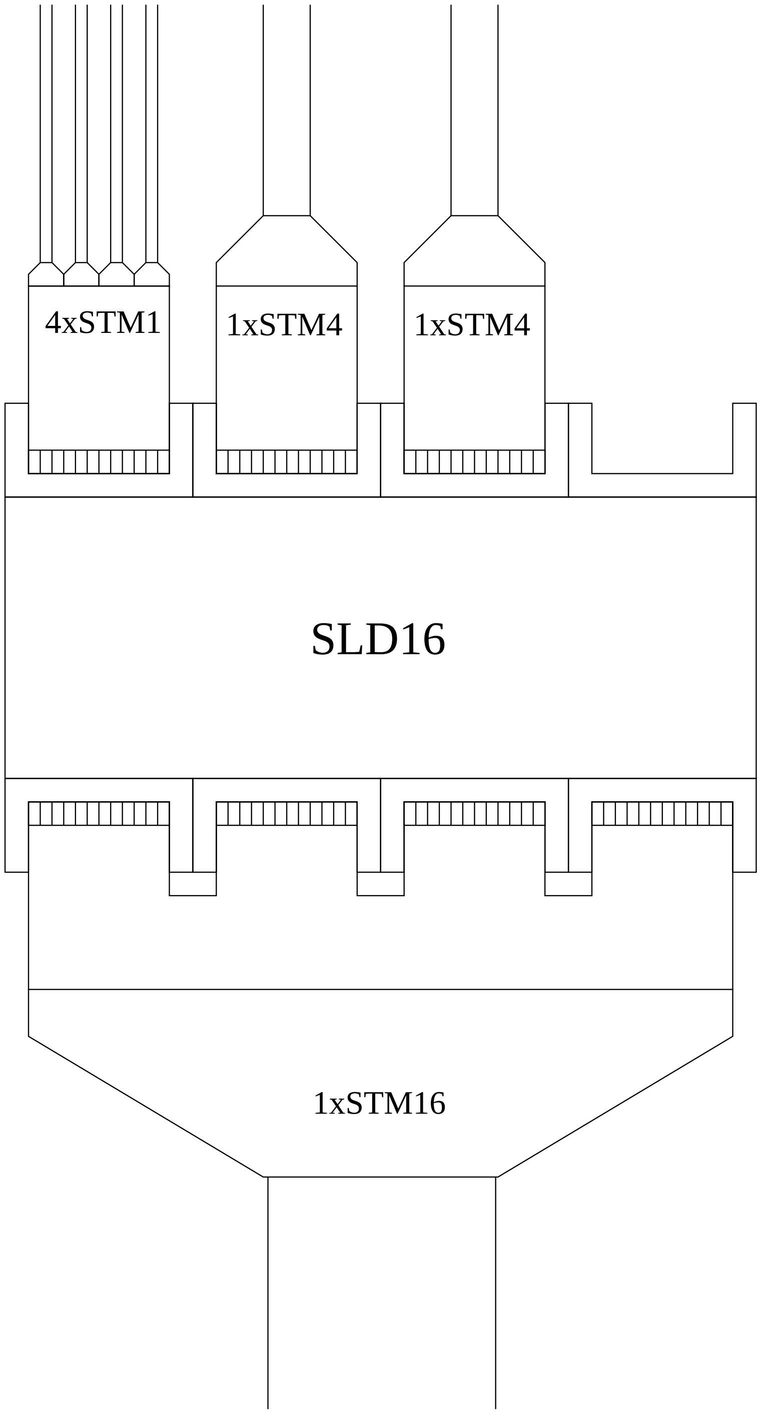

Figure 2: MSI model

2 MSI Model (Modules-Slots-Interfaces)

This model is derived from the models described in [2] and [3].

2.1 Data

2.1.1 Supply graph, demand graph

These graphs provide the topology of the network. The demand graph is supposed to be completely known, whereas variables are associated with the nodes and edges of the supply graph, describing which equipment should be installed.

- $G = (V, E)$: supply graph.
  - V: set of nodes. It corresponds to the physical location of equipments.
  - E: set of edges. It corresponds to the physical or logical links that may be used.
- $H = (V, D)$: demand graph.
  - V: set of nodes. It is the same as the set of nodes of the supply graph.
  - D: set of edges. For each edge $uv \in D$, is defined
    - $d_{uv} \in \mathbb{Z}_+$: communication demand between nodes u and v. In this mode, different service classes (voice traffic, signaling data, data traffic...) are aggregated in a single demand.

2.1.2 Capacities and costs

In this section, the data are defined by the hardware potentially used (switches, multiplexers, transmission links).

- **General parameters**

  These parameters are set once for all the network.

- $S(E)$: set of admissible edge configurations. Edge configurations corresponds to available capacities on physical links. These capacities have a discrete and finite structure due to the equipment used (multiplexer...). It is possible to take into account pre-installed equipment by making it available in all edge configurations.
- $S(V)$: set of admissible node configurations. A node configuration corresponds to one set of equipment used on nodes together with one way to interconnect it. It is possible to take into account pre-installed equipment by making it available in all node configurations.
- I: set of admissible interfaces. An interface is an abstraction for the hardware used to connect a link into a node.
- M: set of admissible modules. A module is a set of interfaces that are physically bounded. It can correspond to a communication card plugged into one slot of a switch.

- **Edge configuration parameters**

  For each edge $e \in E$:

  - a_s^i: number of interfaces $i \in I$ required (at both end-nodes) by edge configuration $s \in S(E)$.
  - C_s: switch capacity required (at both end-nodes) by link configuration $s \in S(E)$. In this model, it is assimilated to the capacity of the edge.
  - K_s^e: cost of edge configuration $s \in S(E)$ for edge $e \in E$. For example, it can be the evaluation of a length-dependent cost function.

- **Module parameters**

  For each module $m \in M$:

  - a_m^i: number of interfaces $i \in I$ provided by module $m \in M$.
  - K_m: cost of a module $m \in M$.
  - β_m: Number of slots used by a module $m \in M$.

- **Node configuration parameters**

  For each node $v \in V$:

  - C_s^v: maximum switchable capacity for node configuration $s \in S(V)$.
  - K_s^v: cost of configuration $s \in S(V)$ at node v. Usually K_s^v does not depend on the node v considered.
  - β_s^v: number of slots provided by node v in configuration s.
  - $\underline{z}_{v,s}^m, \overline{z}_{v,s}^m \in \mathbb{N}$: lower and upper bound for the number of modules $m \in M$ at configuration s at node v.

2.1.3 Routing and diversification

In this section, parameters are added to the model to improve the "quality" of the solution.

- P: a path P between node u and node v is list of connected edges going from u to v.

- $\ell_{uv} \in \mathbb{N}$: path-length restriction parameter. The maximum number of supply edges allowed in any path on which parts of the demand between the end-nodes u and v is routed.

- $\mathcal{P}_{uv}$: set of admissible paths between nodes u and v. These paths satisfy the path-length restriction constraint : $|P| \leq \ell_{uv}, uv \in D$.

- δ_{uv}, $0 < \delta_{uv} \leq 1$: diversification parameter. The maximum fraction of the demand d_{uv} allowed to flow through any edge or node. Diversification is a solution to make a network survivable to the failure of on of its element without rerouting. It is therefore attractive from a maintenance point of view.

2.2 Variables

Capacities variables describes the hardware installed, whereas routing variables describe how data is transported between two end-nodes.

2.2.1 Capacities

- $x_e^s \in \{0,1\}, e \in E$: 1 when the edge configuration s is chosen in the solution and 0 else.
- $x_v^s \in \{0,1\}, v \in V$: 1 when the node configuration s is chosen in the solution and 0 else.
- $z_v^m \in \mathbb{N}, v \in V$: number of modules m at node v.

2.2.2 Routing

- $f_{uv}(P), P \in \mathcal{P}_{uv}, uv \in D$: flow or path variable. Part of the communicating demand d_{uv} routed on path P.

2.3 Mathematical model

We define a feasible network as one set of values for variables satisfying the constraints (2) to (12). We define a solution of the problem as a feasible network minimising the cost objective function (1).

2.3.1 Cost

The cost objective function gives the cost of the whole network, i.e. edges and nodes. This is a global function, so its minimisation *cannot* be seen as an minimisation for each supply or demand edge.

- **Cost objective function**:

$$\min \left[\sum_{e \in E} \sum_{s \in S(E)} K_s^e x_e^s + \sum_{v \in V} \left(\sum_{s \in S(V)} K_s^v x_v^s + \sum_{m \in M} K_m z_v^m \right) \right] \tag{1}$$

2.3.2 Capacities

- **Configuration-bound constraint:**
  One and exactly one configuration must be chosen for each node, and that no more than one configuration can be chosen for each edge.

If no capacity is needed for a node, a configuration with no capacities is available at no cost.

$$\sum_{s \in S(V)} x_v^s = 1,\ v \in V \tag{2}$$

$$\sum_{s \in S(E)} x_e^s \leq 1,\ e \in E \tag{3}$$

- **Edge capacity constraint:**
  Each edge must have a capacity greater than the total demand routed through it.

$$\sum_{uv \in D} \sum_{P \in \mathcal{P}_{uv} : e \in P} f_{uv}(P) \leq \sum_{s \in S(E)} C_s x_e^s,\ e \in E \tag{4}$$

- **Switch capacity cut-inequality**
  Each node must have enough switching capacity to support the edges it is connected to.

$$\sum_{e \in \delta(v)} \sum_{s \in S(E)} C_s x_e^s \leq \sum_{s \in S(V)} C_s^v x_v^s,\ v \in V \tag{5}$$

2.3.3 Modules-Interfaces-Slots

- **Interface inequality:**
  There must be enough of each kind of interface to plug the edges into each node.

$$\sum_{e \in \delta(v)} \sum_{s \in S(E)} a_s^i x_e^s \leq \sum_{m \in M} a_m^i z_v^m,\ i \in I, v \in V \tag{6}$$

- **Slot inequality:**
  There must be enough slots to plug the modules into each node.

$$\sum_{m \in M} \beta_m z_v^m \leq \sum_{s \in S(V)} \beta_s^v x_v^s,\ v \in V \tag{7}$$

- **Module-bound inequalities:**
  There are a lower bound and an upper bound to the number of modules of each type plugged into each node. Contrary to inequality (7)

which sets a constraint on all modules together, these inequalities set a constraint for each module type m.

$$\sum_{s \in S(V)} \underline{z}_{v,s}^m x_v^s \leq z_v^m \leq \sum_{s \in S(V)} \overline{z}_{v,s}^m x_v^s,\ m \in M, v \in V \tag{8}$$

2.3.4 Diversification

The node-flow and the edge-flow constraints guarantee the diversification. It means that if a node or an edge fails (except the two end-nodes u and v), at least δ_{uv} of the demand will still be supplied by the network.

- **Node-flow constraint:**

$$\sum_{P \in \mathcal{P}_{uv}: w \in P} f_{uv}(P) \leq \delta_{uv} d_{uv},\ w \in V \backslash \{u, v\}, e \in E \tag{9}$$

- **Edge-flow constraint:**

$$f_{uv}(P) \leq \delta_{uv} d_{uv},\ uv \in D, P \in \mathcal{P}_{uv} : |P| = 1 \tag{10}$$

2.3.5 Routing

- **Demand constraint:**
  All the demand between two nodes must be satisfied by the network.

$$\sum_{P \in \mathcal{P}_{uv}} f_{uv}(P) = d_{uv},\ P \in \mathcal{P}_{uv}, uv \in D \tag{11}$$

- **Non-negativity constraint:**
  As demands represent data transmission needs, flow variables cannot be negative.

$$f_{uv}(P) \geq 0,\ P \in \mathcal{P}_{uv}, uv \in D \tag{12}$$

2.4 Summary

To sum up, in our MSI model, a feasible network is satisfying to constraints (13) to (24) and minimising the cost objective function as expressed in (13).

- **Cost**

$$\min \left[\sum_{e \in E} \sum_{s \in S(E)} K_s^e x_e^s + \sum_{v \in V} \left(\sum_{s \in S(V)} K_s^v x_v^s + \sum_{m \in M} K_m z_v^m \right) \right] \tag{13}$$

- **Capacities**

$$\sum_{s \in S(V)} x_v^s = 1, \qquad v \in V \tag{14}$$

$$\sum_{s \in S(E)} x_e^s \leq 1, \qquad e \in E \tag{15}$$

$$\sum_{uv \in D} \sum_{P \in \mathcal{P}_{uv}: e \in P} f_{uv}(P) \leq \sum_{s \in S(E)} C_s x_e^s, \ e \in E \tag{16}$$

$$\sum_{e \in \delta(v)} \sum_{s \in S(E)} C_s x_e^s \leq \sum_{s \in S(V)} C_s^v x_v^s, \, v \in V \tag{17}$$

- **Modules-Slots-Interfaces**

$$\sum_{e \in \delta(v)} \sum_{s \in S(E)} a_s^i x_e^s \leq \sum_{m \in M} a_m^i z_v^m, \quad i \in I, v \in V \tag{18}$$

$$\sum_{m \in M} \beta_m z_v^m \leq \sum_{s \in S(V)} \beta_s^v x_v^s, \quad v \in V \tag{19}$$

$$\sum_{s \in S(V)} \underline{z}_{v,s}^m x_v^s \leq z_v^m \leq \sum_{s \in S(V)} \overline{z}_{v,s}^m x_v^s, \, m \in M, v \in V \tag{20}$$

- **Diversification and routing**

$$\sum_{P \in \mathcal{P}_{uv}: w \in P} f_{uv}(P) \leq \delta_{uv} d_{uv}, \, w \in V \backslash \{u, v\}, e \in E \tag{21}$$

$$f_{uv}(P) \leq \delta_{uv} d_{uv}, \, uv \in D, P \in \mathcal{P}_{uv} : |P| = 1 \tag{22}$$

$$\sum_{P \in \mathcal{P}_{uv}} f_{uv}(P) = d_{uv}, \qquad P \in \mathcal{P}_{uv}, uv \in D \tag{23}$$

$$f_{uv}(P) \geq 0, \qquad P \in \mathcal{P}_{uv}, uv \in D \tag{24}$$

3 Heuristics

This heuristic framework is the extension of the ideas described in [6].

3.1 Definitions and notations

3.1.1 Auxiliary variables

All these variables are updated during the algorithm. If the algorithm is successful, then the feasible network is given by the last values of these variables.

- $s^{current} = s, s|x_e^s = 1$: current configuration of edge e.
- $f_{uv}(e) = \sum_{uv \in D} \sum_{P \in \mathcal{P}_{uv}|e \in P} f_{uv}(P)$: part of the demand d_{uv} routed through edge e at the current step of the algorithm.
- $f_{uv}(v) = \sum_{uv \in D} \sum_{P \in \mathcal{P}_{uv}|v \in P} f_{uv}(P)$: part of the demand d_{uv} routed through node v at the current step of the algorithm.
- $y_e = \sum_{uv \in D} f_{uv}(e)$: total demand routed through edge e at the current step of the algorithm.
- $y_w = \sum_{uv \in D} f_{uv}(v)$ total demand routed through node w at the current step of the algorithm.
- $d_{uv}^{remaining} = d_{uv} - \sum_{P \in \mathcal{P}_{uv}} f_{uv}(P)$: remaining demand. Demand still to satisfy.

3.1.2 Auxiliary parameters

We need no decisions to obtain these parameters.

- $k_{uv} = \lceil 1/\delta_{uv} \rceil$: the minimum number of node-disjoint paths to find in order to satisfy a demand edge uv.
- $f^{max}(P) = \max(d_{uv}|uv \in D)$: the value of the most important demand.
- $K = \max(k_{uv}|uv \in D)$: the maximum number of node-disjointed paths we will have to find.

3.1.3 New parameters

We need to take decisions in order to set these parameters.

- $D^{remaining}$: ordered list of remaining demand edges
- $\overline{k}, \overline{i}$: algorithm iteration bounds
- g, $0 < g \leq 1$: greediness function. The greater it is, the eager to use installed capacities the algorithm is.
- a_e: edge attraction parameter. Its value is an estimation of the flow that might be routed through it further on in the algorithm.
- a_v: node attraction parameter. Its value is an estimation of the flow that might be routed through it further on in the algorithm.
- C^e_{pseudo}: pseudo-capacity of edge e. Its value is used by min-cost flows algorithms. $C^E_{pseudo} = (C^e_{pseudo}, e \in E)$.
- K^e_{pseudo}: pseudo-cost of edge e. Its value is used by min-cost flows algorithms. $K^E_{pseudo} = (K^e_{pseudo}, e \in E)$.

3.2 General description of the algorithm

We are trying to build a low-cost network (i.e. topology and capacity) that satisfies the routing constraints. At any step of the algorithm, capacities and diversification constraints are satisfied as well as node balances. It computes a feasible flow. The MSI constraints are satisfied in a second step by an algorithm from DISCNET [3]. This algorithm checks whether the flow returned is feasible and then uses it as a lower bound to find the network configuration of minimum cost on which it is possible to route the feasible flow.

The algorithm is structured as one while loop called k-loop divided into two main parts. The first part is the $CAPACITY\_INSTALL$ functions. It install new capacities. The second part is the $DEMAND\_ROUTING$ function. It routes demand on installed capacities.

- The first part is the installation of capacities using global information. We define for each node and edge an attraction parameter that represent an approximation of the quantity of demand that may be routed through this component in an optimal solution. We use this parameter to install edge capacities.

- The $DEMAND\_ROUTING$ algorithm takes demands one by one and tries to satisfy them at low cost. This process is done through a while loop called the i-loop. If a demand is not entirely satisfied, the algorithm puts the remaining demand at the end of the demand list so as to satisfy it later on an other iteration of the k-loop.

At the end of the heuristic, the algorithm returns a feasible flow, which is used by DISCNET to compute the minimum cost solution satisfying the MSI constraints.

Algorithm 1 Initial heuristics general structure

$INPUT - OUTPUT:$
$-G = (V, E) : supply\ graph$
$-H = (V, D) : demand\ graph$

$VARIABLES\_INIT$
  $\rightarrow x_e^s, f_{uv}(e), a_e, \forall\ e \in E, uv \in D$
  $\rightarrow y_w, f_{uv}(w), a_w, \forall\ w \in V, uv \in D$
$D \rightarrow D^{remaining}$
$(0, K\_BOUND) \rightarrow (k, \overline{k})$
while $(k < \overline{k})$ and $(D^{remaining} \neq \emptyset)$ **do**
  $k + 1 \rightarrow k$
  $CAPACITY\_INSTALL \rightarrow x_e^s, \forall\ e \in E$
  $DEMAND\_ROUTING \rightarrow a_e, a_v, f_{uv}(e), f_{uv}(w), \forall e \in E, w \in V, uv \in D^{remaining})$
end while
if $D^{remaining} \neq \emptyset$ **then**
  $FAILURE$
else
  $RETURN(f_{uv}(e))$
end if

3.3 Capacity installation ($CAPACITY\_INSTALL$)

We want to install capacities using global information on the network. This information can either be given as a data to the program or computed from a flow computed with relaxed constraints. From this and from a correction computed by the $DEMAND\_ROUTING$ function, we compute an attraction parameter for each component and finally we install new capacities on high attraction edges ($CAPACITY\_INSTALL$ function).

3.3.1 Global information

The problem of dimensioning network is easier to solve with continuous variables, so that a program like DISCNET returns during its execution partial solutions for which configurations variables are not integers. However, these solutions are information that worths to be used. We can therefore take a feasible flow associated to a partial solution as a basis to compute the attraction level, i.e. as the guessed flow $f_{uv}^{guessed}(e)$ used in the equation (25).

If we cannot use this information, we can use the same routing algorithm as described in section 3.4 with different pseudo-capacity and pseudo-cost definitions. As a pseudo-capacity definition, we choose the maximum capacity among all possible configurations of one edge, so that the computed flow gives a good idea of attractive edges. As pseudo-cost, we may choose the marginal cost because it helps to use installed capacities completely.

Algorithm 2 Attraction definition and capacity install

{Attraction definition}
$ATTRACTION\_INIT((y_e, C_{s^{current}}, \epsilon_a(e), \forall e \in E), (y_w, \forall w \in V))$
  $\to (a_e, \forall e \in E), (a_w, \forall w \in V)$
for all $uv \in D$ **do**
  for all $c \in G | c \in \mathcal{P}_{uv}^{pseudo}$ **do**
    $a_c + d_{uv}^{remaining} / k_{uv} \to a_c$
  end for
end for

{Capacity installation}
$ATTRACTION\_FUNCTION(a_e, \forall e \in E) \to A(a_e), \forall e \in E$
for all $e = uv \in E$ **do**
  $NEW\_CONFIGURATION\_FIND(A(a_e), x_e^s) \to x_e^s$
  $CONSTRAINT\_CHECK(x_e^s, (\max(C_s^w, \forall s \in S(V)), w \in u, v)) \to x_e^s$
end for

3.3.2 Attraction definition

With a computed flow $f_{uv}^{guessed}(e)$, we affect to each edge the coefficient:

$$a_e = y_e + \epsilon_a(e) + f_{uv}^{guessed}(e), \ e \in E \tag{25}$$

As we will use the attraction parameter as an estimation of capacities to install, if we installed too many capacities at the previous iteration, i.e. $C_{s^{current}} - y_e > 0$, they must be taken off the estimation. The $\epsilon_a(e)$ parameter is a local correction of attraction parameter computed by the *FEASIBLE_FLOW* function. It is defined in equation (50). It can be used if this function "sees" that an edge is really attractive.

With a similar definition, we define the parameter a_v for each node v. We will use it as an objective for the switching capacities to install on each node.

$$a_v = y_v + f_{uv}^{guessed}(v) \tag{26}$$

3.3.3 Capacity installation

The attraction parameter is an estimation of the flow that will be routed through an edge during the iteration. Therefore, thanks to the attraction function $A(a_e)$, we install capacities in this way:

$$s^{current} = s|\min\left((1-g)C_s + gC_{s+1} - A(a_e) \geq 0\right) \tag{27}$$

$$A(a_e) = \frac{A_0}{2}\left(|a_e - \underline{f}f^{max}(P)| + a_e - \underline{f}f^{max}(P)\right) \tag{28}$$

$$f^{max}(P) = \max\left(d_{uv}\delta_{uv}\right), \ uv \in D \tag{29}$$

$$0 < A_0 \ \leq \ 1 \tag{30}$$

$$0 < \underline{f} \ \leq \ 1 \tag{31}$$

We also take care that removing capacity is forbidden, i.e. from one iteration to an other, the installed capacity can only increase.

The greediness g defines a threshold, which is a weighted average between two contiguous configurations: if the attraction is higher than the threshold, the upper capacity will be installed, else the lower capacity is.

The constant A_0 also sets the greediness of the capacity installation. The higher it is, the more capacities are installed. Indeed, the constant $\underline{f}f^{max}(P)$ defines a lower bound for capacity installation: Capacities may be installed only if $a_e \geq \underline{f} f^{max}(P)$. The higher the $\underline{f}$ constant is, the more concentrated on a few edges the installation of capacities is.

The parameters can be k-dependent. For instance, we want to install more capacities but more concentrated at the first iteration:

$$A_0 = \max\left(2 - k, \frac{3}{4}\right) \tag{32}$$

$$\underline{f} = \max\left(\frac{3}{2} - k, 0\right) \tag{33}$$

For the greediness, as it is defined, it must be low at the beginning so that only a few capacities are installed and should be increased at each iteration so that some new capacities are installed. This is why we choose this definition:

$$g = k/(k+1); \tag{34}$$

On this occasion, we could also check node switching capacity constraints ($CONSTRAINT\_CHECK$ function). It guarantees that there is a configuration of the node that is able to support the configuration of the edges connected to it: If the node switching capacity constraints for one of the end nodes of the edge are violated, then we have to redefine $s^{current}$:

$$\exists w \in \{u, v | uv = e\} | \sum_{e \in \delta(w)} C_s x_e^s > \max\left(C_s^w, s \in S(V)\right) \Rightarrow$$

$$s^{current} = s | \min\left(\max(C_s^w, s \in S(V)) - \sum_{e \in \delta(w)} C_s x_e^s > 0\right) \tag{35}$$

Note: It is easy to determine the configuration of the nodes here instead of checking the upper bound limit. In the same way, all the MSI constraints could be checked or determined here.

3.4 Demand routing ($DEMAND\_ROUTING$ function)

This function routes demand on installed capacities. It cannot install new capacities. However, it can add corrections to attraction parameters, so that it induces the installation of new capacities. This section is a detailed description of its structure.

Algorithm 3 DEMAND_ROUTING

$INPUT - OUTPUT:$
$-G = (V, E) : supply\ graph$
$-H = (V, D) : demand\ graph$

$DEMAND\_ORDER(D) \rightarrow D^{remaining}$
$(0, I\_BOUND) \rightarrow (i, \bar{i})$
while $(i < \bar{i})$ and $(D^{remaining} \neq \emptyset)$ **do**
  $i + 1 \rightarrow i$
  $NEXT\_EDGE(D^{remaining}) \rightarrow uv$
  $EDGE\_PSEUDO\_CAPACITY(i, j, uv, f_{uv}(E)) \rightarrow C^E_{pseudo}$
  $EDGE\_PSEUDO\_COST(i, j, C^E_{pseudo}, uv, a_V) \rightarrow K^E_{pseudo}$
  $CAPACITATED\_MIN\_COST\_FLOW(K^E_{pseudo}, C^E_{pseudo}, d^{remaining}_{uv}, uv)$
    $\rightarrow f^{computed}_{uv}(\mathcal{P}_{uv})$
  $FEASIBLE\_FLOW(f_{uv}(E), f_{uv}(V), d^{remaining}_{uv}, f^{computed}_{uv}(\mathcal{P}_{uv}))$
    $\rightarrow f^{feasible}_{uv}(\mathcal{P}_{uv}), lengthboundfailure$
  if $(k \geq k_0)$ and $(lengthboundfailure = 1)$ **then**
    $LENGTH\_RESTRICTED\_FLOW(C^E_{pseudo}, d^{remaining}_{uv}, uv)$
      $\rightarrow f^{feasible}_{uv}(\mathcal{P}_{uv})$
  end if
  $UPDATE(f_{uv}(E), f_{uv}(V), d^{remaining}_{uv}, f^{feasible}_{uv}(\mathcal{P}_{uv}))$
    $\rightarrow, f_{uv}(E) f_{uv}(V), d^{remaining}_{uv}$
  if $(d^{remaining}_{uv} > 0)$ **then**
    $ADD\_END(D^{remaining}, uv)$
  end if
end while

First of all, we want to use classic min-cost flow algorithms described in [4], especially Andreas Bley's implementations (cf [5]). Therefore, we have to define capacities and costs usable by these algorithms, i.e. to define a maximum flow routable on each edge, and a cost for each edge, usually seen as the cost of one unit of flow routed. We call this capacity a pseudo-capacity and this cost a pseudo-cost, as they are defined using the original capacities or costs among other parameters. The $EDGE\_PSEUDO\_CAPACITY$ and the $EDGE\_PSEUDO\_COST$ functions implement these definitions.

Then, we use a min-cost flow algorithm with the pseudo-capacities and pseudo-costs previously defined to find a flow satisfying edge diversification and optionally capacity constraints ($CAPACITATED\_MIN\_COST\_FLOW$ function). However, this function does not take into account length restriction and node diversification constraints, so that the computed flow $f_{uv}^{computed}(\mathcal{P}_{uv})$ might violate these constraints. Therefore, we need a function to check these constraints and gives a feasible flow $f_{uv}^{feasible}(\mathcal{P}_{uv})$ from the computed flow. This flow is then added to the network by the $FLOW\_UPDATE$ function.

Finally, in case of length restriction constraint violation, instead of postponing the demand, we may use the $LENGTH\_RESTRICTED\_FLOW$ heuristic described in algorithm 4 to try to satisfy the demand under conditions discussed in section 3.4.5. This algorithm is an heuristic based on the $LENGTH\_RESTRICTED\_SHORTEST\_PATH$ algorithm. It does not take any cost into account.

At this step, the satisfaction of the remaining demand is postponed.

Algorithm 4 Length restricted flow heuristic ($LENGTH\_RESTRICTED\_FLOW$)

$DEFINE\_PSEUDO\_COST$
while $(j < \overline{j})$ and $(d_{uv}^{remaining} > 0)$ **do**
  $j = j + 1$
  $LENGTH\_RESTRICTED\_SHORTEST\_PATH(uv, \ell_{uv}, K_{pseudo}^{e})$
  $PATH\_FLOW\_DETERMINATION(P) \rightarrow f_{uv}^{computed}(P)$
  $d_{uv}^{remaining} = d_{uv}^{remaining} - f_{uv}^{computed}(P)$
  $UPDATE\_PSEUDO\_COST$
end while

3.4.1 Pseudo-capacity (PSEUDO_CAPACITY function)

We speak of the pseudo-capacity, because $CAPACITATED\_MIN\_COST\_FLOW$ and $PATH\_FLOW\_DETERMINATION$ functions take this edge pseudo-capacity as the upper bound of the part of the demand that can be supplied by this edge. This pseudo-capacity is function of the capacity and other parameters like the loop iterators k and i. In all strategies, one must keep in mind that, to take into account diversification, this pseudo-capacity is limited by $d_{uv}\delta_{uv} - f_{uv}(e)$, where:

$$f_{uv}(e) = \sum_{uv \in D} \sum_{P \in \mathcal{P}_{uv} | e \in P} f_{uv}(P) \tag{36}$$

Let the guessed flow $f_{uv}^{guessed}(e)$ be the flow that might be routed to the edge e during the iteration. Usually, $f_{uv}^{guessed}(e) = d_{uv}\delta_{uv} - f_{uv}(e)$

- Free capacity. The pseudo-capacity is the capacity available at no cost an the edge. It guarantees that the flow will be routing on installed capacities.

$$C^e_{pseudo} = C_s - y_e, s = s^{current} \tag{37}$$

- Maximum capacity: The pseudo-capacity is the maximum capacity of the edge. It is the more greedy strategy.

$$C^e_{pseudo} = C_s - y_e, s | \max(C_s) \tag{38}$$

- The pseudo-capacity is defined by this equation:

$$\begin{aligned} C^e_{pseudo} &= C_s - y_e, \\ s \geq s^{current} | \min(|C_s - & \; y_e \; - C^e_{guessed}|) \end{aligned} \tag{39}$$

  This means that a capacity is considered in the pseudo-capacity definition only if more than half of it would be used if the guessed flow $f_{uv}^{guessed}(e)$ is routed.

- A generalisation of the last two pseudo-capacities is to introduce the greediness coefficient. If configuration are ordered by capacity, we have the following definition:

$$\begin{aligned} C^e_{pseudo} &= C_s - y_e, \\ s \geq s^{current} | \min(|(1-g)C_s + gC_{s+1} - & \; y_e \; - C^e_{guessed}|) \end{aligned} \tag{40}$$

If $g = 1$, the pseudo-capacity is equivalent to the maximum capacity of the edge. If $g = 0$, the pseudo-capacity is equivalent to the third definition. In fact g is a lower bound for the fraction of a capacity increase that the guessed flow $f_{uv}^{guessed}(e)$ must use in order that this capacity increase is considered in the pseudo-capacity. If the used fraction is lower, this capacity will not be considered in the pseudo-capacity.

A strategy is to use the last definition, so that we have to define the greediness function. We want the algorithm to be conservative at the beginning of i-loops so that the flow is routed through existing capacities. On the contrary, at the end of i-loops, we want to know where capacities lack, so that greediness function g should increase with i. Moreover, g should decrease with k, because the odds that installed capacities remain unused augment towards the end of the algorithm. So, we define the greediness function:

$$g = \min(i/\bar{i}, 1 - k/\overline{k}) \tag{41}$$

As a strategy, in a first time, the pseudo-capacity of an edge will be defined as the maximum installable capacity of the edge to compute attraction levels and will be equal to the currently installed capacity of the edge to compute routings.

To use the greediness, we should use a pseudo-capacity larger than the installed capacity. The problem is that it is likely to be that all the flow returned by the $CAPACITATED\_MIN\_COST\_FLOW$ algorithm cannot really be routed, so that we have to be able to route demands partially, which was not implemented. However, the information that part of the flow could not be routed on an edge could be then used to correct the attraction coefficient of the edge on the next iteration, as defined in equation (50).

3.4.2 Pseudo-cost (PSEUDO_COST function)

We speak of pseudo-cost of a path, because the $CAPACITATED\_MIN\_COST\_FLOW$ function use this parameter as a cost function but it is different from the cost of using this path, as the quantity of demand that will be routed through it is not yet known. Let the guessed capacity $C_{guessed}^{e}$ be the capacity that might be added to the edge e.

- 1^{st} unit cost: the pseudo-cost is the cost of the first unit of demand transported. This strategy favors the use of existing free capacities.

$$K^e_{pseudo} = K^e_s - K^e_{s_{current}},$$
$$s|\min(C_s - y_e - 1 \geq 0) \tag{42}$$

- Marginal cost: relative cost of the first unit installed

$$K^e_{pseudo} = \frac{K^e_s - K^e_{s_{current}}}{C_s - C_{s_{current}}},$$
$$s|\min(C_s - y_e - 1 \geq 0) \tag{43}$$

- Relative cost: average cost of the capacities of the last configuration installed when $C^e_{guessed}$ units of demand are added on the edge. $s-1$ is the preceding configuration with configurations sorted by increasing capacity.

$$K^e_{pseudo} = \frac{K^e_s - K^e_{s-1}}{C_s - C_{s-1}},$$
$$s|C_s - y_e \geq C^e_{guessed} \tag{44}$$

- Free cost: average cost of capacities installed when $C^e_{guessed}$ units of demand are added on the edge. If no capacities are installed, it worths 0. It helps to use already installed configurations.

$$K^e_{pseudo} = (C_s - y_e)\frac{K^e_s - K^e_{s-1}}{C_s - C_{s-1}},$$
$$s|C_s - y_e \geq C^e_{guessed} \tag{45}$$

- Used cost: cost of used capacity on the current edges. This cost is low when a configuration is little used and high when it is almost fully used. It helps to use edges where configurations are little used. It is useful to avoid saturating edges.

$$K^e_{pseudo} = (y_e - C_{s_{current}})\frac{K^e_s - K^e_{s-1}}{C_s - C_{s-1}},$$
$$s|C_s - y_e \geq C^e_{guessed} \tag{46}$$

- Edge cost: From any cost definition K^e_{pseudo}, one defines the edge cost K^e_{edge} by:

$$K^e_{edge} = \frac{K^0_{edge}}{|E|} + \frac{K^e_{pseudo}}{\sum_{e_1 \in E} K^{e_1}_{pseudo}} \tag{47}$$

  This costs helps to find length-restricted paths when looking for the lowest cost path with no length restriction. The edge cost constant K^0_{edge} defines the importance of length in this composite cost. This parameter could be for example j-dependent so as to increase the importance of length when j increases.

- Node cost: The node cost K^e_{node} of edge $e = uv$ is:

$$K^e_{node} = \frac{1}{2}\left(K_{node}\left(\frac{y_u}{a_u}\right) + K_{node}\left(\frac{y_v}{a_v}\right)\right) \tag{48}$$

$$K_{node}(c) = K^0_{node} * c + \frac{K^0_{edge}}{|E|} \tag{49}$$

  This cost helps to prevent the saturation of a node, as the cost of using it increases when a node is near from saturation. The node cost constant K^0_{node} defines the importance of node saturation when using this cost in a composite cost. Finally, the edge cost is a particular node cost with $K^0_{node} = 0$, so that the edge cost is redundant if a node cost is already used.

- With these definitions, what are possible strategies?

  - A strategy is to use the marginal cost. It goes with a conservative general strategy.
  - A strategy is to use the pseudo-capacity as a value for $C^e_{guessed}$ to determine a pseudo-cost using any cost definition, for instance relative cost. It goes with a greedy strategy for capacity installation.
  - We will try relative, free, used, and marginal cost for the attraction computation, and relative and used cost for the routing. As used and free cost have opposite definition, the comparison of results will show the influence of cost definition. In all cases, we will use the current computed partial flow as guessed capacity.

3.4.3 Flow feasibility (FEASIBLE_FLOW function)

The algorithm compute flows that might not be feasible. Therefore, we must check constraints to decide if a flow is feasible or not. If it is not, when possible, we remove part of the flow so that the remaining part is feasible. That is why we check these constraints in this order:

- Length restriction constraints

  We check the length restriction constraint for every computed flow paths. If it is violated, the flow is not affected and $lengthboundfailure$ is set to 1. If the $LENGTH\_RESTRICTED\_FLOW$ algorithm computed the flow, then this verification is useless.

  To check this constraint, we have to find enough length restricted paths in the flow to carry all the demand, i.e. to decompose the flow in paths. For that purpose, we can use the $LENGTH\_RESTRICTED\_FLOW$ heuristic with the value of the flow as pseudo-capacities.

- Node diversification constraints

  We check the node diversification constraint for nodes $w \in f_{uv}^{computed}(\mathcal{P}_{uv})$. The affectation is impossible if $f_{uv}(w) + f_{uv}^{computed}(w) > \delta_{uv} d_{uv}$. With the $CAPACITATED\_MIN\_COST\_FLOW$, bounding the edge and node pseudo-capacity by $\delta_{uv} d_{uv} - f_{uv}(w)$ is enough to guarantee diversification. Inside the $LENGTH\_RESTRICTED\_FLOW$ heuristic, paths are found one by one. The flow affected to paths found takes into account the node diversification constraints. If a node is saturated, the algorithm makes it impossible to find new paths using it.

- Edge capacity constraints

  Finally, we check the edge capacity constraint. It is useless if the pseudo-capacity is defined as the free capacity. If a capacity constraint is violated, i.e. $f_{uv}(e) + f_{uv}^{computed}(\mathcal{P}_{uv}) > C_{s^{current}}$, then we set $C_{pseudo}^{e} = C_{s^{current}} - f_{uv}(e)$. Once we checked all the edge capacity constraints and updated the pseudo-capacity, we apply a max flow algorithm to the subset of the network for which $f_{uv}^{computed}(\mathcal{P}_{uv}) > 0$. The result is a feasible flow.

  Besides, an edge for which the capacity constraint is violated is an edge for which we did not install enough capacity, so that we should install more capacities at the next iteration of the algorithm. This is why we introduce here a correction $\epsilon_a(e)$ to attraction parameter:

$$\epsilon_a(e) = \max(0, f_{uv}(e) + f_{uv}^{computed}(\mathcal{P}_{uv}) - C_{s_{current}}) \qquad (50)$$

3.4.4 Flow update (UPDATE function)

This function add a flow $f_{uv}(P)$ to a network by updating its variables.

- Edge and node affected flow:

$$f_{uv}(e) \leftarrow f_{uv}(e) + f_{uv}^{feasible}(e) \qquad (51)$$
$$f_{uv}(v) \leftarrow f_{uv}(v) + f_{uv}^{feasible}(v) \qquad (52)$$

- Finally, we update in the same way the pseudo-capacity of edges on which flow is affected, and return the remaining demand.

3.4.5 General routing strategy

- How should demands be ordered?

  The order of demands is a key problem of this algorithm. In fact, a trade-off between cost and chance to find a solution must be taken.

  With a given demand order, the algorithm might fail to find any solution. For example, if an edge has a small maximum capacity, if the first demands have saturated the edge, if the current demand can only be satisfied through this edge, the algorithm will fail, whereas it may have succeeded if this demand had been satisfied before.

  The first idea is to order them by demand size, so as to provide resource for the large demands and then fill in the gaps with the small ones.

  An other idea is to determine a lowest cost path through the network, using for example the relative cost of the first unit to be installed on an edge as a pseudo-cost for this edge. Then, demands could be ordered by decreasing $(d_{uv} * pseudo\_cost\_of\_lowest\_cost\_path)$. With this idea, we satisfy demands requiring more resources first.

  However, these two ideas do not help to find a solution when the maximum capacity of an edge is lower than the flow that the algorithm would like to route through it. We would like to use an information related to the "difficulty" to satisfy a demand. For example a demand is probably difficult if its end-nodes are low-connected. We call connectivity the minimum degree of the end-nodes.

To see if these different values are useful to sort the demand, we will compare the result of the heuristic with demand sorted in increasing and decreasing order. The gap between the results should tell what is the best order.

- When the satisfaction of a demand should be postponed?

  When the demand is considered, the $CAPACITATED\_MIN\_COST\_FLOW$ algorithm is first used. Demand may remain for different reasons:

  - Length-bound restriction is violated. Then we have either to postpone the demand so that the installation of new capacities opens a new short path or to use the $LENGTH\_RESTRICTED\_FLOW$ path algorithm. To lessen the risk of such a violation, we use an edge cost with the $CAPACITATED\_MIN\_COST\_FLOW$ algorithm.
  - The maximum flow of the network is lower than the demand. Then, the algorithm will fail again unless pseudo-capacities are changed: there are not enough capacity installed, we have to postpone the demand. If this happens during the computation of attraction level, i.e. when pseudo-capacities are set to their maximum, we know that it will be impossible to find a solution with the demand sorted in the current order, so the algorithm puts the demand at the front of the remaining demand list.

  Therefore, the failures fall into two parts: One the one hand, capacity failures, for which new capacities have to be installed. We can postpone the remaining demand in this case. On the other hand, length-bound failures, for which postponing the demand might not be efficient, so that we may use the $LENGTH\_RESTRICTED\_FLOW$ algorithm.

  Finally, the strategy we will use is to postpone the satisfaction of all remaining demands except if the remaining demands comes from a length-bound failure and that we already tried k_0 times to satisfy this demand.

4 Results

The implementation of the ideas developed in the previous part was realised in C++, using the environments and libraries of the DISCNET software.

Tests have been realized over 15 problems covering 8 networks. This problems are the modelisation of real telecom operators problems. The "-nodiv" extension means that the diversification parameter is set to one for all demands and the "-div" extension means that the diversification is set to 0.5 for all demands. The "-greenfield" and "-preinstalled" extensions show whether resources are already installed on the given network.

Due to the short time remaining for the tests, the influence of two parameters has been studied: the demand order and the pseudo-cost definition.

Network	Nodes	Edges	Demands
s1_nodiv	11	34	24
s1_div	11	34	24
s2_nodiv	11	34	24
s2_div	11	34	24
s3_nodiv	18	33	101
s3_div	18	33	101
s4_div	22	34	143
m2_nodiv	19	34	33
m2_div	19	34	33
m3_greenfield	58	87	672
m3_preinstalled	58	87	672
m4_nodiv	50	88	682
m4_div	50	88	682
b2_greenfield	44	129	861
b2_preinstalled	44	129	861

Table 4: Networks

4.1 Influence of demand order

For the study of demand order, the pseudo-cost is always defined as the used cost, for both attraction computation and routing. Indeed, this definition leads to robust heuristics that give rather good results.

4.1.1 Demand value

Whereas it seems more logical to satisfy large demands first in order to build the core of the network, it happens that routing small demands first is more efficient. It even gave the best result for *s2_div*.

However, the intuition is that routing large demands at the end is risky. Indeed, they require a lot of capacities, that may not be available anymore at the end of the routing. This is why the heuristic did not find solutions for two of the networks with the increasing order, whereas it found results with the decreasing order.

Network	Decreasing	Increasing
s1_nodiv	13571832	13571832
s1_div	17606045	20108839
s2_nodiv	15303049	-
s2_div	21540482	19175978
s3_nodiv	18780697	21355534
s3_div	23355831	24100627
s4_div	14206256.500	13450529.500
m2_nodiv	24571496	24571496
m2_div	31524779	30105414
m3_greenfield	41134777.900	44554946.660
m3_preinstalled	49639334.120	48486588.080
m4_nodiv	66619.400	-
m4_div	-	-
b2_greenfield	11176678	8812034
b2_preinstalled	15363928.364	14247681.364

Table 5: Demand sorted by value

4.1.2 Length of shortest path between end nodes

The intuition concerning this order was that it was better to route first demands for which nodes are separated by only few edges. Indeed, it is likely to be that there are only a few possible paths between these nodes. Therefore, the idea is to build the core of the network with neighborhood demands and then to route long-distance demands on installed capacities.

The result is that the increasing order is almost always better, and even the best overall result for *s2_div*. At the same time it is more risky and more often fails.

Network	Decreasing	Increasing
s1_nodiv	-	13571832
s1_div	17606045	20108839
s2_nodiv	15303049	-
s2_div	21540482	19175978
s3_nodiv	20154371	17711420
s3_div	26279386	23039501
s4_div	13450529.500	14206256.500
m2_nodiv	24571496	24571496
m2_div	31115090	29622094
m3_greenfield	42466488.130	41898715.120
m3_preinstalled	48861737.300	49974257.160
m4_nodiv	68628.400	-
m4_div	-	-
b2_greenfield	10274950	10576445
b2_preinstalled	14749992.364	14891407.364

Table 6: Demand sorted by length of shortest path between end nodes

4.1.3 Connectivity

The idea of this order, where the connectivity is the minimum degree of the end nodes, is to get a solution in the case the other orders failed. Indeed, it is probable that a demand with a low connectivity will have only few possibility to be satisfied, so that it should be satisfied first.

The result is that this order is the only one that allowed the algorithm to find a solution for *m4_div*. However, a bug linked to an internal limitation of the implementation prevented the algorithm from finding a solution for the small problem instances.

Network	Decreasing	Increasing
s1_nodiv	15542882	-
s1_div	21520283	-
s2_nodiv	18242158	-
s2_div	21628225	-
s3_nodiv	19369066	19572988
s3_div	24447764	25142718
s4_div	14206256.500	13450529.500
m2_nodiv	24571496	24571496
m2_div	31206945	31058917
m3_greenfield	-	37682947.890
m3_preinstalled	-	49039293.140
m4_nodiv	-	67186.800
m4_div	-	77344.200
b2_greenfield	11970577	10081114
b2_preinstalled	15225948	14299478.364

Table 7: Demand sorted by connectivity

4.1.4 Shortest path length - demand value product

What seems a good idea to build the core of a network is to route first long-distance big demands, hence the idea of the product, which is an approximation of the amount of resources used to satisfy the demand.

The increasing order gave good results and even the best results for *m3_preinstalled* and *b2_preinstalled*, whereas the decreasing order is below average. The increasing order seems to be an interesting order for problems with preinstalled capacities. Indeed, this order is not interessant to build a network, because it does not route the demands that requires the more resources first, but it can be good to use already installed capacities.

Network	Decreasing	Increasing
s1_nodiv	-	13571832
s1_div	17606045	20108839
s2_nodiv	15303049	-
s2_div	21540482	19175978
s3_nodiv	19886982	19690776
s3_div	23727534	26549374
s4_div	13450529.500	13450529.500
m2_nodiv	24571496	24571496
m2_div	31260503	30105414
m3_greenfield	41523255.860	44672165.710
m3_preinstalled	49062245.420	47263159.200
m4_nodiv	-	-
m4_div	-	-
b2_greenfield	11720093	9095242
b2_preinstalled	15310147.364	14105030.364

Table 8: Demand sorted by length of shortest path - demand value product

4.2 Pseudo-cost definition

For the study of the influence of the pseudo-cost, the demands are ordered by decreasing value, because of the robustness of this order.

4.2.1 Relative cost

We used the relative cost for both attraction and routing. As the used cost does not help to use installed capacity completely, we expect the results to be better with the relative cost and the odds of success to be lower.

The result shows that the odds of success are lower (it failed on *m4_nodiv*) and, one the other hand, it can find good solutions (it found the best one for *s2_nodiv*).

Network	Relative Cost	Used Cost
s1_nodiv	13612213	13571832
s1_div	24035032	17606045
s2_nodiv	14604909	15303049
s2_div	22728512	21540482
s3_nodiv	25705942	18780697
s3_div	27403138	23355831
s4_div	13443115.500	14206256.500
m2_nodiv	27514800	24571496
m2_div	38735714	31524779
m3_greenfield	35127765.230	41134777.900
m3_preinstalled	49639334.120	49639334.120
m4_nodiv	-	66619.400
m4_div	-	-
b2_greenfield	11176678	11176678
b2_preinstalled	15363928.364	15363928.364

Table 9: Relative cost - used cost comparison

4.2.2 Free cost

The use of the free cost is consistent only for the attraction computation. So, we compare here these two heuristics: (free cost for attraction - used cost for routing) with (used cost for attraction - used cost for routing). The results shows that the free cost is seldom better than the used cost, so that this cost should not be used. It did not find any best solution.

Network	Free Cost	Used Cost
s1_nodiv	14230993	13571832
s1_div	23842705	17606045
s2_nodiv	19356147	15303049
s2_div	22306177	21540482
s3_nodiv	26752312	18780697
s3_div	30269545	23355831
s4_div	13443115.500	14206256.500
m2_nodiv	32879142	24571496
m2_div	42386869	31524779
m3_greenfield	41134777.900	41134777.900
m3_preinstalled	49639334.120	49639334.120
m4_nodiv	80451.400	66619.400
m4_div	-	-
b2_greenfield	11176678	11176678
b2_preinstalled	15363928.364	15363928.364

Table 10: Free cost - Used cost comparison

4.2.3 Marginal cost

Once again, we keep the used cost for the routing but we change the cost to marginal cost for the attraction computation. This pair marginal cost-used cost is quite good. It gives three overall best results for *s3_nodiv*, *s4_div*, and *m2_div*.

We can also remark that it has found a better solution on the *m2_div* problem than on the *m2_nodiv* problem, whereas a solution of the first problem is obviously a solution of the second one.

Network	Marginal Cost	Used Cost
s1_nodiv	18548164	13571832
s1_div	-	17606045
s2_nodiv	20639577	15303049
s2_div	29392657	21540482
s3_nodiv	17425514	18780697
s3_div	-	23355831
s4_div	12672788.860	14206256.500
m2_nodiv	30591425	24571496
m2_div	26560367	31524779
m3_greenfield	43515563.930	41134777.900
m3_preinstalled	49639334.120	49639334.120
m4_nodiv	75399	66619.400
m4_div	-	-
b2_greenfield	11176678	11176678
b2_preinstalled	15363928.364	15363928.364

Table 11: Marginal cost - Used cost comparison

4.3 Best results

In this section we compare the best results obtained with heuristics to DISCNET results. It never happened that the heuristic found a better result. We can see that the best heuristic does not always come from the same parameters, which is a problem to get quickly a rather good solution.

Nevertheless, it seems that sorting the demands by value has the most significant effect: There are good results in both direction, and the decreasing order leads to robust heuristics. It also seems that using the marginal cost for attraction often leads to good solutions, however used cost should be preferred to get a robust algorithm.

Network	DISCNET	Heuristic	Parameters
s1_nodiv	12319470	13571832	uu-vd, uu-vi, uu-li, uu-lvi
s1_div	14881216	17606045	uu-vd, uu-ld, uu-lvd
s2_nodiv	12081464	14604909	rr-vd
s2_div	18980976	19175978	uu-vi, uu-li
s3_nodiv	12747272	17425514	mu-vd
s3_div	16615649	22698315	mr-vi
s4_div	9697104.940	12672788.860	mu-vi, mu-vd
m2_nodiv	20196525	24571496	uu-vd, uu-ld, uu-li, uu-lvd, uu-lvi, uu-vd, uu-ci
m2_div	-	26560367	mu-vd
m3_greenfield	-	32618655.790	fuu-vi
m3_preinstalled	-	47263159.200	uu-lvi
m4_nodiv	-	66619.400	uu-vd
m4_div	-	77344.200	uu-ci
b2_greenfield	-	8812034	uu-vi, mr-vi, fr-vi, fu-vi
b2_preinstalled	-	14105030.364	uu-lvi

Heuristic naming convention: uv-xy
u,v: pseudo cost definition for attraction computation and for routing respectively, u - used, f - free, r - relative, fu - first unit, m - marginal
x: demand order, v - value, l - length, c - connectivity, lv - length-value product
y: i - increasing order, d - decreasing order

Table 12: Best results

Conclusion

Based on the MSI model, the heuristics lead to results ready to use by telecommunication operators. These first results show that it worth continuing to improve the heuristic. For example, the implementation of the routing of part of demands together with the implementation of attraction correction by the routing algorithm would lead to a faster and better heuristic.

The heuristic has been developed with an object-oriented design. It is fully integrated in the DISCNET environment, using the MSI model. Therefore, it is a good basis for further developments, and eventually integration to the main algorithm of the software. Indeed, the combined use of linear programming and combinatorial heuristic techniques will allows the researcher to extract the information needed to compute a quality solution to the problem.

References

[1] Andreas Bley, Alexander Kröller, Roland Wessäly, *A component-resource model for the design of communication networks*, Z.I.B. and Atesio GmbH, 2002.

[2] Andreas Bley, Alexander Kröller, Roland Wessäly, *MSI - Models based on Modules-Slots-Interfaces*, Z.I.B, 2002.

[3] Roland Wessäly, *DImensioning Survivable Capacitated NETworks*, Cuvillier Verlag Göttingen, 2000. ISBN 3-89712-864-0.

[4] Ravindra K. Ahuja, Thomas L. Magnant, James B. Orlin, *Network flows*, Prentice Hall, 1993. ISBN 0-13-617549-X.

[5] Andreas Bley, *Node-Disjoint Length-Restricted Paths*, Diplomarbeit, Technische Universität Berlin, 1997.

[6] Jose Neto, *Larges Heuristiques Combinatoires pour des réseaux de communication fiables*, Rapport de stage, Z.I.B. and I.S.I.M.A., 2001.

[7] Bruce Eckel, *C++ inside & out*, Osborne McGraw-Hill, 1993. ISBN 0-07-881809-5.