Evolutionary Computation Techniques for Traffic Supervision Based On A Model Of Telephone Networks Built From Qualitative Knowledge*

Isabelle Servet, Louise Travé-Massuyes
Laboratoire d'Analyse et d'Architecture des Systemes
Centre National de la Recherche Scientifique
7, Avenue du Colonel Roche,
31077 Toulouse
FRANCE
e-mail:servet@laas.fr

Daniel Stern
CNET
France Telecom Research Center
38-40, rue du Général Leclerc
92131 Issy-les-Moulineaux
FRANCE

Abstract: Evolutionary computation techniques have been paid great attention as regards their potential as optimization techniques for complex functions. In this paper, we consider three of them: multiple restart hill-climbing, population-based incremental learning and genetic algorithms. Their binary version and a real-coded variant of each of these techniques are experimented on a real problem: traffic supervision in telephone networks. Indeed, this task needs determine streams responsible for call losses in a network by comparing their traffic values to nominal values. However, stream traffic values are not directly available from an on-line data acquisition system and, hence, they have to be computed by inverting a simple computational model of stream propagation in circuit-switched networks only based on Erlang's formula equation plus qualitative knowledge about the network.

Keywords : Evolutionary Computation, Genetic Algorithms, Qualitative Modeling, Telephone Networks.

Isabelle Servet received the Computer Science Engineering diploma and an Artificial Intelligence M.Sc. degree from the ENSEEINT School of Engineers, Toulouse, France, in 1993. Currently she prepares a Ph.D in Computer Science Engineering. Her research interests are in the fields of artificial intelligence and telecommunication networks.

Louise Travé-Massuyes is a CNRS researcher who works at LAAS where she is Head of the Intelligent Control Systems Research Group. The main interests of this group are qualitative reasoning and its applications to simulation and dynamic systems diagnosis, fuzzy control and supervised control.

Daniel Stern, received the Ph.D degree in Mathematical Statistics in 1984. In 1985. he joined the CNET (France Telecom Research Center) where he first worked on data analysis and statistical problems. Since 1988, he has been working at the Teletraffic Department where he is currently concerned with the network traffic management studies: performance evaluation, network management, network simulation, artificial intelligence use in these problems.

1. Introduction

Evolutionary computation is an umbrella term used to describe computer-based problem solving systems which use computational models of evolutionary processes as key elements in their design and implementation [Hei94].

They all share a common conceptual base of simulating the evolution of *individual* structures via processes of *selection*, *reproduction* or *mutation*. The processes depend on the perceived performance, computed thanks to a *fitness measure*, of the individual structures as defined by their environment. This fitness measure is a function that returns a value which is supposed to be proportional to the "utility" or "ability" of the corresponding individual [Beas93a]. Hence, before an evolutionary algorithm can be run, a suitable representation for each individual structure must be determined. It is assumed [Beas93a] that a potential solution to a problem may be represented as a set of parameters (for instance the number of streams that go through a network). These parameters, known as *genes* are put together to form a string of values, often called a *chromosome*.

Besides, genetic algorithms and other evolutionary algorithms (such as multiple restart hill-climbing or population-based incremental learning [Bal94]) are commonly used for static function optimization [Bal95]. However, the effectiveness of one or the other of these techniques strongly depends on the function to optimize. Thus, in complex systems with epistasis (i.e. high connectivity between genes), such as telephone network traffic supervision, only empirical comparisons can show whether a technique gives better results than another one or not. Moreover, whereas binary-coded evolutionary algorithms are traditionally considered, a growing number of researchers in the evolutionary computation techniques community has supported the idea of real-coded genes. [Esh92] show that real-coded variants of these algorithms, especially for genetic algorithms, often have an advantage over binary-coded versions (for instance, it exploits local continuities in function optimization) although

* This work has been supported by CNET as part of contract n°93 1B 142, project n° 513

some failures can occur with real-coded evolutionary algorithms.

Hence, in the second part of this paper, we present both binary and real-coded variants of three evolutionary algorithms: multiple restart hill-climbing, population-based incremental learning and a standard genetic algorithm. Then, in Part 3, we describe our stream propagation model which is dedicated to compute all traffic losses and offered traffic for each organ, for each stream and the traffic losses and offered traffic for each stream at each network element, all this given the network topology (its size and structure) and the offered traffic to each stream. The six previous evolutionary algorithms are, then applied to this model in Part 4 of this article.

2. Evolutionary Computation Techniques

Three evolutionary computation techniques are presented here:

- *multiple restart hill-climbing*: an heuristic and, then, very simple method;
- *population-based incremental learning* which can be seen as an abstraction of genetic algorithms;
- *a standard genetic algorithm*, a well-known method to solve complex optimization problems.

They have been chosen for the pretty good results they give in many situations and for their increasing complexity.

As there already exists a real-coded variant for genetic algorithms, we have conceived, thanks to real-coded genetic algorithms principles, the real variant of the two other methods which initially were binary ones.

2.1 Multiple Restart Hill-Climbing (MRHC)

2.1.1 Binary Version

This iterative method, very simple as it is, can lead, for many problems, to better results than genetic algorithms do [Bal94]. The simplest algorithm describing it is:

```
V ← randomly generated solution vector
BEST ← V
Loop NB_ITERATIONS
  MUT_POSITION ← random bit position
  N ← mutation of V in MUT_POSITION
  If N is better than BEST
        BEST ← N
        V ← N
```

What follows considers a variant of this algorithm which consists in maintaining a list of mutation position bits that have been attempted without any improvement. These positions are not attempted again until a better solution (according to the fitness measure) is found.

When a better solution is found, the mutation positions list is emptied. If the list becomes as large as the solution vector size, no mutation of V can improve a better solution than V itself. Then, the algorithm is restarted at a new random location with a new empty mutation positions list. Finally, once the total number of iterations completed, the best solution ever found is returned.

2.1.2 Real-coded Variant

First, the solution vector V, which is randomly generated , like in the binary version of the method, is a real vector. Hence, when optimizing a function, each component of this vector represents the value of the corresponding variable.

Then, *MUT_POSITION* (see 2.1.1) corresponds to the number of the variable which has to undergo a mutation. Therefore, a mutation of the i^{th} component of solution vector V (i.e. *V[i]*) consists, like in genetic algorithms real-coded variants [Beas93b], in:

- either adding to *V[i]* a small random real (for instance, a real between *-V[i]/4 and V[i]/4)*
- or multiplying *V[i]* by a random real close to 1 (for instance, a real between *0.75* and *1.25*).

As these 2 possibilities are equivalent, the numerical results presented in Part 4 of this paper have been computed with the former one only.

2.2 Population-based Incremental Learning (PBIL)

2.2.1 Binary Version

Contrary to the previous method, the algorithm does not study only a single solution vector but a set of *nb* (*nb* being chosen by the user) solution vectors called a *generation*.

Population-based incremental learning is a combination of evolutionary computation and hill-climbing [Bal94]. The object of this method is to create a real valued *probability vector P* whose size is equal to the solution vector size and which is dedicated to reveal high quality solution vector with high probabilities. Hence, P is

updated at each generation and the solution vectors of each generation are created according to its values: each component *V[i]* of each solution vector *V* is determined as follows:

Given $random_i$ a random value between 0 and 1,
if $random_i > P[i]$ then $V[i] \leftarrow 0$ else $V[i] \leftarrow 1$

Therefore, the i^{th} component of *P* represents the probability of having '1' at the i^{th} position of the best solution vector.

P evolution, from a generation to another one, is computed thanks to competitive learning mechanisms.

Then, this method needs some user defined constants [Bal94] :

- **GENERATIONS**, the number of iterations to allow learning.
- **NB**, the number of samples to produce per generation.
- **LENGTH**, the size of encoded solution.
- **MUT_PROBABILITY**, the probability that a mutation occurs at each position.
- **MUT_SHIFT**, the amount for mutation to affect the **probability vector.**
- **LR** ,the learning rate and **NEGATIVE_LR**, the negative learning rate.

and its algorithm can be described as follows :

```
For i in 1.. LENGTH   P[i] ← 0.5
Loop GENERATIONS
   Generate the NB solution vectors according to P values
   BEST ← best of the NB solution vectors (according to the fitness measure)
   WORST ← worst of the NB solution vectors
   /* Update probability vector P according to BEST and WORST. */
   For i in 1.. LENGTH   P[i] ←P[i]*(1-LR) + BEST[i]* LR
   For i in 1.. LENGTH
     If BEST[i] ≠ WORST[i]
        P[i] ← P[i]*(1-NEGATIVE_LR) + BEST[i]* NEGATIVE_LR
   /* Mutation of probability vector P */
   For i in 1.. LENGTH
    random_i ← random value in [0,1]
   If random_i < MUT_PROBABILITY,  P[i] ← P[i] * (1 - MUT_SHIFT)
```

As the effectiveness of this algorithm strongly depends on the choice of the user defined constants, they can also be computed by a meta-level evolutionary algorithm [Gref86].

2.2.2 Real-coded Variant

First, we assume that each gene (i.e. each variable in the case of function optimization) has a continuous and bounded definition set. Hence, we consider that each component *V[i]* of a solution vector *V* belongs to the interval $[Low_i , Up_i]$.

Moreover, in this variant of the PBIL method, LENGTH represents the number of system unknown parameters.

Each component *P[i]* of *P* represents the probability that the i^{th} variable value is greater than $\frac{Low_i + Up_i}{2}$.

The probability vector *P* updating is similar to *P* updating in the binary version of the algorithm but a stage for $[Low_i , Up_i]$ intervals updating is added.

Then, for a given generation, the i^{th} component of the probability vector *P* is such that:

- ***P[i]* ≥ 0.9,**

Therefore, *V[i]* is certainly greater than $\frac{Low_i + Up_i}{2}$. Low_i is updated to $\frac{Low_i + Up_i}{2}$ and *P[i]* is reinitialized at 0.5.

- ***P[i]* ≤ 0.1,**

Therefore, *V[i]* is surely lower than $\frac{Low_i + Up_i}{2}$. Up_i is updated to $\frac{Low_i + Up_i}{2}$ and *P[i]* is reinitialized at 0.5.

The reinitialization of the probability vector *P* is due to the fact that the change of a bound of the $[Low_i , Up_i]$ interval makes the algorithm start again with new definition sets for the systems parameters.

2.3 A Standard Genetic Algorithm (GA)

2.3.1 Binary Version

In this method, an offspring can result from a gene mutation of an existing chromosome, as in

the two previous algorithms, but also from 2 chromosomes crossover. We have chosen to apply *1-point crossover* which is the most basic form of the operator [Beas93a]. This crossover operator takes two individuals and "cuts" their chromosome strings at some randomly chosen position to produce two "head" and two "tail" segments. Then, the tail segments are swapped over to produce two full length chromosomes.

As the parents selection is randomized, there is no guarantee that the best solution in the current population will be selected for reproduction. Then, if this solution vector is lost from the population, there is no guarantee that it will be found again. Methods such as elitist strategies have been proposed to address this problem : they ensure that the best solution is transferred to the current generation [Bal94]. We have chosen a common implementation of this solution which consists in replacing the worst chromosome of the new generation by the best chromosome of the old generation.

Hence, a standard genetic algorithm description [Beas93a,Bal94,Hei94] is:

Generate POPULATION_SIZE random vectors
Compute fitness of each individual of the population
Loop GENERATIONS
  Loop POPULATION_SIZE / 2
    • Select probabilistically (according to fitness evaluation) two vectors *ONE* and *TWO* from the old generation
    • Recombine (thanks to crossover) *ONE* and *TWO* to give two offsprings *CHILD1* and *CHILD2*
    • Perform mutation, according to MUTATION_RATE, of *CHILD1* and *CHILD2*
    • Compute fitness of *CHILD1* and *CHILD2*
    • Insert *CHILD1* and *CHILD2* in the new generation
  WORST ← worst vector of the new generation
  BEST ← best vector of the old generation
  Replace *WORST* by *BEST* in the new generation

where the user defined constants are:

- **GENERATION**, the number of generations the algorithm is allowed to continue.
- **POPULATION_SIZE**, the number of samples generated by generation.
- **MUTATION_RATE**, the probability that a mutation occurs.

2.3.2 Real-coded Variant

As in the real variant of MRHC, the only changes in real-coded GAs concern the way the chromosomes operators (mutation and crossover) are done.

The main mutation operators have been described in 2.1.2.

Many real-coded crossover operators can be envisaged (the arithmetic average, the geometric mean,...). However, to keep the trace of some random choice in the crossover operators (as in 2.3.1), we apply a crossover operator based on *blend-crossover* (BLX-α) [Esh92] and which is able to create, like usual binary crossover operators, two offsprings.

Let us consider two parents p_1 and p_2 and their two offsprings c_1 and c_2. Then, each component $c_i[j]$ (with i = 1,2) of c_i is a random value that belongs, if we assume that $p_1[j] < p_2[j]$, to the interval

$$[p_1[j] - \alpha|p_1[j] - p_2[j]|\ ,\ p_2[j] - \alpha|p_1[j] - p_2[j]|\ ]$$

where α is a user specified genetic algorithm parameter.

3. Stream Propagation Model

Due to the increasing size of telephone networks (in particular the French long distance network) and to their high connectivity (several organs are shared by several streams), the task of network managers becomes more and more complex. To determine which streams are responsible for an overloaded situation is difficult to do, especially as the disturbances may propagate within the network in a very short time frame.

If real time networks supervision does not require accurate values (the approached values are sufficient enough to incriminate the streams responsible for overloads and to determine the best actions for minimizing the effects of disturbances) but instead it required that the values are readily computed .

Thus, the stream propagation computation cannot be done by means of a simulation program because of the extremely long computer running time required for simulation. This is the reason why analytical methods are interesting to use [LeG84]. The simplest approach to network analysis is the one-moment method where offered traffic is characterized by only one parameter: its mean.

The model presented here uses an one-moment method and has been developed under the

classical assumptions of single moment modeling methods [Pas84,LeG84].

This model is based on the concept of blocking networks elements and the search for such elements in the network.

Definition 1. A network element (NE): exchange or circuit group is said to be *blocking* when, according to the network structure, this NE is likely to experiment traffic loss.

It can be divided into 3 main stages:

- *Blocking organ search.* It is indeed in these organs that stream propagation disturbances (i.e. traffic loss) occur.
- *Blocking NEs ordering.* This is especially interesting when overflow from one circuit group to another is allowed, with the offered traffic of the latter depending on the amount of lost traffic of the former.
- *Traffic loss computation* for each blocking NE.

Some numerical results and comparisons with other one-moment models can be found in [Ser96].

3.1 Blocking Organs Search

This first stage consists in removing from the set of possible blocking NEs the ones on which loss is very unlikely. It is performed by taking into account only the network topology and according to the following two rules:

- ***Overdimensioned capacities rule*** :

A NE whose capacity is greater than the maximal number of the upstream resources (i.e. in case of a circuit group, the sum of the capacities of circuit groups arriving to the origin of the studied NE and being used by at least one of the streams routed through this circuit group) is very unlikely to sustain traffic loss.

This first qualitative rule can be processed independently of the network streams offered traffic values.

- ***Loss rates rule:***

If an organ capacity is not overdimensioned, its resource availability must be estimated. This is performed by using **Erlang's formula** (cf. Definition 2) as a qualitative indicator: if the result given by this formula is not null, the studied NE is assumed as blocking and its loss rate is equal to this result.

Definition 2. Consider an organ whose capacity is N and which routes a traffic stream of intensity A[1]. Then, the probability that the N circuits are busy is:

$$E[N,A] = \frac{A^N/N!}{1+...+A^N/N!}$$

3.2. Blocking NEs Ordering

When a call is offered to a circuit group whose circuits are all busy, it may (depending on its routing table) overflow to another group. In this way, a stream can divide itself into several substreams that have the same destination switch.

Let us consider, for instance, the following network, the stream $stream_1$ going from switch 4 to switch 5 and the stream $stream_2$ going from switch 4 to switch 3 of which routes appear in the following Figure:

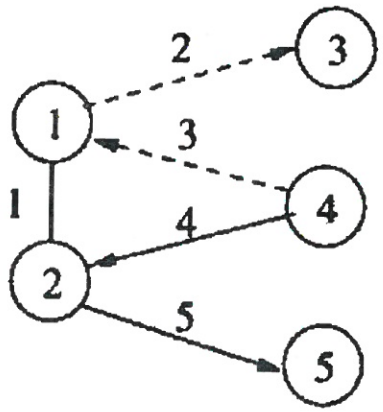

Figure 1. Network Configuration

Let us suppose now that the circuit group 4 is overloaded. Then $stream_1$ may overflow to circuit group 3. A fictitious substream, going from switch 4 to switch 5 through circuit groups 3,1 and 5 is created. The route of this substream appears in Figure 2.

In this situation, it is obvious that the offered traffic value to circuit group 3 depends on the traffic loss on circuit group 4. Then, traffic loss on circuit group 4 must be studied before the computation of traffic loss on circuit group 3. This blocking NEs ordering is as important as large the number of streams routing through each organ .

[1] A is equal to the sum of offered traffic values to each stream which goes through the NE considered.

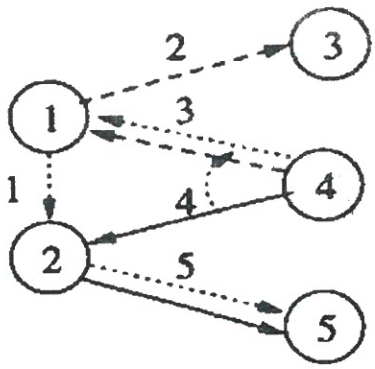

Figure 2. Fictitious Substream Instance

3.3. Traffic Loss Computation

Once the blocking NEs are determined and the blocking NEs ordering is studied, an algorithm whose basic principle is the **resource availability backpropagation** is applied. In fact, as observed from tests made with the network simulator SuperMac developed at CNET [Ste93], *a local propagation method* has to be banned. This local propagation method would consist in considering each NE individually, in the route order of the studied stream, and in applying successively Erlang's formula, by updating the values of the traffic offered to this NE in accordance with the number of call losses that have occurred upstream. This method is improper because of the phenomenon we call *resource availability backpropagation,* an example of which is described below.

Let us take, for instance, the following network and the stream whose path is hereafter represented in dotted line (see Figure 3).

Let us consider a value of 70 Erlangs for the stream intensity, a capacity equal to 50 for switch 3 and a capacity of 30 for circuit group 2. The other organs capacities are equal to 100. Then, a local propagation method indicates that calls are lost in both switch 3 and circuit group 2.

However, if looking at the simulated results, we can see that no call is lost on switch 3. They are all lost in circuit group 2. As a matter of fact, the resource availability backpropagation phenomenon occurs because the telephone traffic sets up end to end. Thus, every call occupies a certain resource at each NE it uses (e.g. a circuit in the case of circuit groups) during the whole communication duration.

Let us provide a simple explanation, in the case of the previous example. The NE whose loss rate, computed based on Erlang's formula, is the

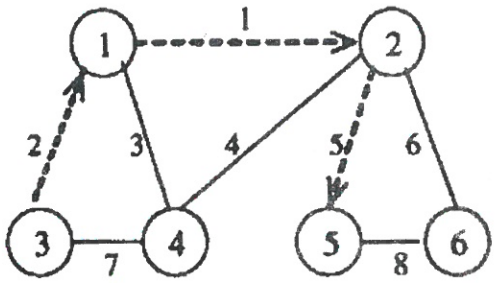

Figure 3. Route of Flux 3->5

highest[2] is situated downstream of the first organ on which losses are possible (i.e. switch 3). The loss occurring in circuit group 2 impacts on switch 3 because the lost calls do not last as long as initially foreseen. They occupy, only during the short period of call set up, the involved resources which may, then, be available for new calls that would have failed otherwise.

In consequence, we apply a **resource availability backpropagation** in the following sense. The value of lost traffic on the *most blocking NE* is backpropagated on the upstream NE. Finding the most blocking organ is not always obvious: several NEs may have equal loss rates. For instance, let us consider a network carrying only one stream. If, in this case, 2 NEs in the stream path have exactly the same capacity N and, hence, the same loss rate, the blocking organs search stage tells us that there cannot be call losses on the downstream organ. All the calls are lost on the most upstream NE.

Let us consider now that the upstream NE capacity is equal to $N+1$ while the other organ capacity is still equal to N. Then, according to the *resource availability backpropagation*, most of the calls are lost on the downstream organ. Two different behaviours have thus been determined and they seem to lead to a discontinuity in the stream propagation model. As stream propagation is obviously a continuous phenomenon, our model computes in a specific way the case of organs which have *almost equal* loss rates.

Definition 3. Consider two loss rates R_1 and R_2. They are said to be *almost equal* if :

$$R_1 \geq 0.125 \text{ and } \frac{|R_1 - R_2|}{R^1 - R^2} < 0.075 \quad \text{or}$$

$$R^1 < 0.125 \text{ and } \frac{|R_1 - R_2|}{R^1 - R^2} < 0.6$$

If several organs have almost equal loss rates, the model tries to determine whether certain organs can be considered as more blocking than the

[2] The organ whose capacity is the lowest when the organs are not shared among many streams (for example, when only one stream goes through the network)

others and, then, to study them by "sharing" the losses corresponding to the same streams between these organs as explained in the following section.

3.4 Equal Loss Rates Case

Let us consider n organs O_1, O_2,....O_n ($n \leq 2$) whose loss rates are *almost equal* and which can be considered as the most blocking NEs.

Then, the stream propagation model tries to reduce the number of blocking organs considering the following two rules:

- ***Upstream priority rule:***
  Let us suppose that 2 NEs O_1 and O_2 are shared by the same stream. Then, there were, a priori, as much chance that calls are lost on O_1 as on O_2. The rule says that, according to an intuitive logic, they are lost in priority on the first NE thrown, e.g. on the most upstream organ.
- ***Larger stream number rule:***
  The larger the number of streams that go through an organ is, the more the organ is considered as blocking.

Once these 2 rules successively applied to O_1, O_2 ... O_n, the following 3 possibilities can turn up :

- *Only one organ has been chosen* :
  The traffic loss computation is based on Erlang's formula.
- *Some organs can be considered as more blocking than the others* :
  The traffic loss computation is performed by the application of the following two rules that have been deduced from the simulated results :
  - For each stream which goes through a set O_{i1}....O_{im} ($m < n$) of chosen organs, the sum of the traffic loss on O_{i1}....O_{im} is equal to the traffic loss, computed with Erlang's formula, on any organ O_{i1}....O_{im}.
  - Each organ O_{ij} ($j = 1...m$) acts as a high-pass filter for traffic loss towards its upstream NEs of O_{i1}....O_{im}.
- *The n organs are as much blocking :*
  Let us consider an organ O_i of O_1, O_2 ... O_n of which capacity is equal to C_i . Then, if we call:
  - Up_i the capacity such that $Up_i > C_i$ and (Up_i +1) is the lower bound such that O_i loss rate is not *almost equal* to the other loss rates of O_1, O_2 ... O_n
  - $Down_i$ the capacity such that $Down_i < C_i$ and ($Down_i$ -1) is the upper bound such that O_i loss rate is not *almost equal* to the other loss rates of O_1, O_2 ... O_n

  we notice, thanks to simulations, that, for each organ O_i, traffic loss is a linear function between $Down_i$ and Up_i.

4. Model Inversion

To supervise the French long distance network, we propose to determine streams responsible for call losses by comparing their traffic values to nominal values. However, stream traffic values are not directly available from the on-line data acquisition system and, hence, need to be computed in real time. This is done by inverting our stream propagation model based on the evolutionary computation techniques presented in Part 2.

It consists in iteratively calling the stream propagation model with stream traffic values, updated by one of the previous methods, and using a *fitness measure* to compute the distance between observed values and computed values. First, the observed (measured) quantity can be either the number of calls offered to each organ or the stream carried traffic value for each stream, the only on-line available measures in the French long distance telephone network.

Thus, the distance between observed values and computed values can be :

- Euclidean Distance (ED), defined by $\sqrt{\sum (\text{observed value} - \text{computed value})^2}$
- χ^2 distance (χ^2), defined by $\sqrt{\sum \frac{(\text{observed value} - \text{computed value})^2}{\text{observed value}}}$
- Infinite Norm (IN), defined by *max* |observed value - computed value|

These distances have been chosen for their simplicity and have been tested through both the number of calls offered to each organ and the stream carried traffic value for each stream.

The four optimization methods described in Part 2 have been tried successively with the six previous fitness measures, the three distances being applied to the number of offered calls (OC) and the stream carried traffic (CT).

In a perspective of real-time traffic supervision, each of these methods has been computed with 500 iterations only (whereas, in many

evolutionary computation techniques, the number of iterations is about 20 times larger).

The PBIL user defined constants have, as proposed in [Bal95], the following values :

MUT_PROBABILITY = 0.02,

MUT_SHIFT = 0.05,

LR = 0.1

NEGATIVE_LR = 0.075.

For each stream i, the initial value of Low_i (cf. 3.1.2 Real Variant) is null and the initial value of Up_i is equal to ten times the nominal offered traffic value of stream i. Hence, we consider that no overflow greater than 900% of the nominal traffic value (this bound has been given by telecommunication experts) can occur.

The GA user defined constants are: POPULATION_SIZE = 50, MUTATION _RATE = 0.01 and α, the GA real-coded variant parameter, is taken equal to 0.5. Moreover, as our stream propagation model is a qualitative one and, so it yields imperfect results, we have chosen to select the parents of any crossover thanks to *tournament selection*. In fact, it is a useful and robust selection mechanism often used in conjunction with noisy fitness function [Mil95].

Some numerical results are given in Appendix A.

5. Conclusion

First of all, although the inverted model gives only approached values and despite its complexity, non-linearity and non differentiability, the results computed based on evolutionary computation techniques are rather good.

So, an important conclusion of our study is the influence of distance choice on the evolutionary computation techniques performance. Unfortunately, actually, there is no way, except for empirical methods of course, to say whether a distance is able to give better results than another one or not.

Besides, our application clearly shows that, sometimes, a real-coded variant, that is closer to human reasoning, can outperform a binary evolutionary technique. It also points out the importance, especially in the real-coded case, of mutation operator in the first stages of evolutionary algorithms. Hence, as the results are computed with a quite small number of iterations, the best results are provided by the real-coded variant of multiple restart hill-climbing.

Finally, the quite disappointing results of a standard genetic algorithm for our model inversion can be explained by the small size of its population: if the GA population size increases the GA will be able to maintain more dissimilar points in its population and, then to use crossover more efficiently [Bal95]. However, in a perspective of real-time traffic supervision, a larger population GA is not envisaged because of its long running time.

REFERENCES

[Bal94] BALUJA, S., **Population-based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning**, Technical Report CMU-CS-94-163, 1994. Available by anonymous ftp at reports.adm.cs.edu.

[Bal95] BALUJA, S., **An Empirical Comparison of Seven Iterative and Evolutionary Function Optimization Heuristics,** Technical Report CMU-CS-95-193,1995. Available by anonymous ftp at reports.adm.cs.edu.

[Beas93a] BEASLEY, D., BULL, D.R. and MARTIN, D.R., **An Overview of Genetic Algorithms : Part 1, Fundamentals**, UNIVERSITY COMPUTING, 15(2), 1993, pp. 58-69.

[Beas93b] BEASLEY, D. , BULL, D.R. and MARTIN, D.R., **An Overview of Genetic Algorithms : Part 2, Research Topics**, UNIVERSITY COMPUTING, 15(4), 1993, pp. 170-181.

[DJo92] DE JONG , K., **Genetic Algorithms Are NOT Function Optimizers,** Foundations of Genetic Algorithms 2, MORGAN KAUFMANN PUBLISHERS , 1992, San Mateo, CA, USA.

[Esh92] ESHELMAN, L.J. and SCHAFFER, J.D., **Real-coded Genetic Algorithms and Interval-Schemata**, Foundations of Genetic Algorithms 2, MORGAN KAUFMANN PUBLISHERS, 1992, San Mateo, CA, USA.

[Gref86] GREFNSTETTE, J.J,. **Optimization of Control Parameters for Genetic Algorithms**, IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS, Vol. 16, No 1, 1986, pp. 122-128.

[Hei94] HEITKOTTER, J. and BEASLEY, D., **The Hitch-Hiker's Guide To Evolutionary Computation: A List of Frequently Asked Questions(FAQ)**,1994, Usenet :

comp.ai.genetic. Available by anonymous ftp from rtfm.mit.edu: /pub/usenet/news.answers/ai-faq/genetic/

[LeG84] LE GALL, F., BERNUSSOU, J. and GARCIA, J.M.,. **A One-moment Method for Telephone Networks with Dependence On Link Blocking Probabilities,** PERFORMANCE, 1984), pp. 449-458.

[Mil95] MILLER, B.L. and GOLDBERG, D.E., **Genetic Algorithms, Tournament Selection and the Effects of Noise,** IlliGAL Report No. 95006, 1995. Available by anonymous ftp at gal4.ge.uiuc.edu: /pub/papers/IlliGALs/

[Pas84] PASSERON, A., **Notions élémentaires sur le trafic téléphonique**, Technical Report DE/ATR/57.84, CNET, Issy-les-Moulineaux, France, 1984.

[Ser96] SERVET, I., TRAVÉ-MASSUYES, L. and STERN, D., **Traffic Supervision Based On A One-moment Model of Telephone Networks Built from Qualitative Knowledge**, IMACS/IEEE CESA'96 International Conference, Villeneuve d'Ascq, France, July 1996.

[Ste93] STERN,D., **SuperMac V2.3 Un logiciel de simulation de réseaux pour la gestion en temps réél du trafic téléphonique**, Technical Report DEPAA/ ATR/111.91/DS.CNET, Issy-les-Moulineaux, France, 1993 .

APPENDIX

Let us consider the following network :

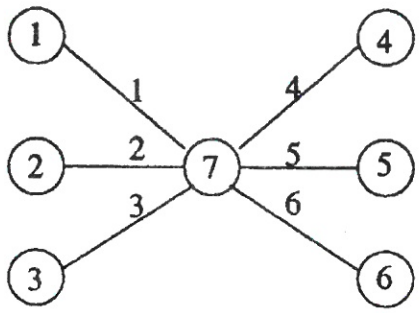

Figure 4. One Transit Switch

where the switch capacities are infinite, the circuit group capacities are:

Table 1. Circuit Group Capacities

Circuit group	1	2	3	4	5	6
Capacity	**2**	**4**	**6**	**3**	**3**	**5**

and the offered traffic values of (I, J) streams (i.e. going from I switch to J switch) are:

Table 2. Offered Traffic Values (in Erlangs)

(I,J) Stream	(1,4)	(1,5)	(1,6)	(2,4)	(2,5)	(2,6)	(3,4)	(3,5)	(3,6)
Traffic value	**0.5**	**0.5**	**0.8**	**0.6**	**0.8**	**1.2**	**1.3**	**1.4**	**2.7**

Then, the different algorithms presented in this article provide the following results:

Table 3. Computed Offered Traffic Values (in Erlang)

		(1,4)	(1,5)	(1,6)	(2,4)	(2,5)	(2,6)	(3,4)	(3,5)	(3,6)
Binary MRHC	IN (CT)	**0.29**	**0.29**	**0.38**	**0.38**	**0.38**	**0.29**	**0.14**	**0.27**	**0.27**
	IN (OC)	**0.30**	**0.28**	**0.28**	**0.38**	**0.38**	**0.28**	**0.14**	**0.28**	**0.28**
	ED (CT)	**0.29**	**0.27**	**0.29**	**0.38**	**0.38**	**0.29**	**0.14**	**0.29**	**0.29**
	ED (OC)	**0.30**	**0.28**	**0.28**	**0.38**	**0.38**	**0.28**	**0.14**	**0.28**	**0.28**
	χ^2 (CT)	**0.30**	**0.28**	**0.29**	**0.38**	**0.38**	**0.29**	**0.14**	**0.28**	**0.28**
	χ^2 (OC)	**0.30**	**0.28**	**0.28**	**0.38**	**0.38**	**0.28**	**0.14**	**0.28**	**0.28**
Real-coded MRHC	IN (CT)	**0.18**	**0.25**	**0.18**	**0.07**	**1.09**	**0.17**	**0.34**	**0.20**	**0.07**
	IN (OC)	**0.29**	**0.33**	**0.28**	**0.15**	**0.55**	**1.27**	**0.74**	**0.48**	**0.03**
	ED (CT)	**0.50**	**0.47**	**0.89**	**0.32**	**1.13**	**1.36**	**1.40**	**0.91**	**2.07**
	ED (OC)	**0.29**	**0.33**	**0.28**	**0.15**	**0.55**	**1.36**	**1.36**	**0.48**	**0.03**
	χ^2 (OC)	**0.54**	**0.57**	**0.93**	**0.32**	**1.18**	**1.26**	**1.37**	**1.12**	**0.07**
	χ^2 (CT)	**0.29**	**0.33**	**0.28**	**0.15**	**0.55**	**1.36**	**1.36**	**0.48**	**0.03**
Binary PBIL	IN (CT)	**0.32**	**0.61**	**0.51**	**0.46**	**1.49**	**3.31**	**2.64**	**0.73**	**1.87**
	IN (OC)	**0.28**	**0.36**	**1.66**	**0.40**	**0.43**	**3.37**	**2.22**	**1.36**	**0.47**
	ED (CT)	**0.76**	**0.92**	**1.45**	**0.74**	**1.64**	**4.00**	**2.73**	**2.36**	**6.61**
	ED (OC)	**0.43**	**0.02**	**0.37**	**0.59**	**2.00**	**0.04**	**0.51**	**0.02**	**3.13**
	χ^2 (CT)	**0.64**	**0.91**	**1.98**	**0.76**	**2.47**	**3.95**	**2.37**	**2.36**	**6.62**
	χ^2 (OC)	**0.07**	**0.20**	**1.54**	**1.30**	**0.55**	**2.56**	**0.99**	**2.02**	**2.05**
Real-coded PBIL	IN (CT)	**1.37**	**2.32**	**6.80**	**1.44**	**2.28**	**6.84**	**6.04**	**4.13**	**16.74**
	IN (OC)	**0.75**	**0.37**	**2.84**	**1.68**	**0.36**	**3.24**	**1.49**	**5.67**	**10.93**
	ED (CT)	**2.20**	**2.80**	**5.28**	**0.45**	**2.08**	**3.96**	**11.83**	**7.84**	**19.17**
	ED (OC)	**1.82**	**0.85**	**0.76**	**0.60**	**3.28**	**10.96**	**4.29**	**2.94**	**1.62**
	χ^2 (CT)	**2.87**	**2.85**	**6.20**	**2.82**	**3.84**	**7.62**	**9.36**	**6.44**	**17.95**
	χ^2 (OC)	**0.47**	**0.50**	**3.28**	**0.57**	**2.64**	**2.52**	**0.78**	**7.91**	**4.59**
Binary GA	IN (CT)	**0.12**	**0.32**	**0.30**	**0.13**	**0.33**	**0.41**	**0.40**	**0.34**	**1.41**
	IN (OC)	**0.29**	**0.28**	**0.29**	**0.40**	**0.37**	**0.28**	**0.14**	**0.29**	**0.27**
	ED (CT)	**0.40**	**0.29**	**0.71**	**0.29**	**0.41**	**0.84**	**1.73**	**0.41**	**2.50**
	ED (OC)	**0.36**	**0.30**	**0.28**	**0.14**	**0.23**	**0.29**	**1.74**	**0.28**	**2.50**
	χ^2 (CT)	**0.40**	**0.27**	**0.40**	**0.34**	**0.50**	**0.93**	**1.92**	**0.41**	**2.50**
	χ^2 (OC)	**0.40**	**0.40**	**0.94**	**0.28**	**0.29**	**0.40**	**1.69**	**0.23**	**3.05**
Real-coded GA	IN (CT)	**2.50**	**6.17**	**8.89**	**3.96**	**26.10**	**24.18**	**15.84**	**17.62**	**17.62**
	IN (OC)	**1.20**	**5.53**	**2.99**	**2.20**	**2.19**	**7.21**	**5.61**	**12.48**	**9.56**
	ED (CT)	**2.95**	**11.20**	**11.83**	**7.13**	**25.08**	**25.96**	**13.95**	**14.57**	**29.97**
	ED (OC)	**1.53**	**1.24**	**8.06**	**1.51**	**2.77**	**4.10**	**4.25**	**5.01**	**1.32**
	χ^2 (CT)	**2.95**	**7.60**	**12.01**	**5.03**	**26.46**	**26.36**	**11.09**	**14.44**	**24.99**
	χ^2 (OC)	**1.06**	**3.94**	**0.49**	**3.77**	**3.04**	**2.53**	**2.39**	**0.40**	**4.08**