Telecommunication Network Design with Parallel Multi-objective Evolutionary Algorithms

Susana Duarte Flores, Benjamín Barán Cegla, and Diana Benítez Cáceres

Centro Nacional de Computación, Universidad Nacional de Asunción
San Lorenzo, Paraguay - P.O. Box: 1439
bbaran @ cba.com.py
http://www.cnc.una.py

Abstract

This paper proposes parallel asynchronous versions of promising multi-objective evolutionary algorithms, implemented over a network of personal computers, with the aim of designing an optimal telecommunication network in the presence of multiple conflicting objectives as cost and performance. The resulting tool provides a set of Pareto optimal solutions, facilitating the decision making process of designing a telecommunication network with a mix of different technologies.

1 Introduction

The design of communication networks has been solved using operational research approaches for several years [1]. At the beginning, it was solved as a single objective optimization problem, using the cost of the network as a typical objective to be minimized, subject to several constrains as reliability, maximum delay, etc.

However; it is now clear that the design of a communication network is better stated as a multi-objective optimization problem [2]. In this new multi-objective context, the aim of a designer is to simultaneously optimize a set of conflicting objectives as: reliability, cost, delays, throughput, capacity, etc, while maintaining restrictions over another set of requirements as: minimum reliability, maximum cost, maximum acceptable delay, minimum speed, etc. This problem is known to be NP-Hard [3].

Many approaches have been designed to address this problem, some of them based on various kinds of graph perturbation heuristics [4, 5], and others founded in techniques from artificial intelligence (taboo search [6], simulated annealing [7] and genetic algorithms [1, 2, 8, 9]). An interesting summary of these methods can be found in [9]. To shorten the discussion it is useful to say that:

a) none of them treats the problem as a multi-objective problem, but they would rather choose an objective to

LANC'03 October 3-5, 2003, La Paz, Bolivia

optimize, leaving the others as restrictions;

b) all of them can be applied only to networks of limited magnitude, and in very restricted situations. As the size of commercial systems grows there is a complete lack of tools to aid in the designing process; and the methodology of trial & error that has been applied is neither effective nor efficient.

The present work proposes the use of *Multi-Objective Evolutionary Algorithms (MOEA)* to solve the design problem and presents an implementation of two versions of the *Strength Pareto Evolutionary Algorithm (SPEA)* [10] and the *Non Dominated Sorting Genetic Algorithm (NSGA)* [11]. This paper also examines and compares the results obtained with both algorithms, with the aim of helping the designer of a network to choose the best tool for his/her work. The SPEA was chosen because it implements elitism through the maintenance of an external population of best solutions found during the whole generational loop; then, convergence is guaranteed [12]. NSGA was chosen because of its promising experimental results [13]. As this later algorithm does not implement elitism, we have decided to alter its original formulation slightly. In addition; it was added an external population with the only purpose of archiving good solutions already found. This external population of non-dominated solutions does not participate of the genetic operators.

The rest of this work is organized in the following way: section 2 introduces the problem to be solved with its restrictions and generalities. Section 3 discusses the test problem. Section 4 and 5 contain descriptions of our implementations. Section 6 includes performance metrics used for the testing procedure. Section 7 presents experimental results. Finally, section 8 presents some conclusions and directions for further work.

2 Statement of the Problem

A network can be modeled by a probabilistic undirected graph [1] $G = (V, L, p)$, where:

- V is the set of nodes.
- L is the set of links (arcs). The cardinality of L is also the number of possible links and can be expressed as

$$n = |L| = \frac{|V|(|V|-1)}{2} \quad (1)$$

- and p is the reliability of links.

A network design problem consists in choosing the communication links of different characteristics (or technologies) between a given set or location centers (nodes). The resulting network should acquire a certain set of values for the objectives (as cost and reliability) and complied with another set of requirements (as maximum cost or minimum reliability).

From the above definitions, it is obvious that the problem of a backbone network design optimization can be expressed as a multi-objective optimization problem. As the problem can be as big as a designer states it (i.e., he can choose as many objectives as he wants and as many kind of links as technology and budget lets him), there is a need to place limits on it, according to the available computer resources he has access to. In the present work, the design problem is stated as the optimization of only two objectives (k =

2): reliability and cost. The fact that every network topology must be connected is expressed by restricting reliability to positive values. Then, the proposed solutions must meet a single and very simple reliability requirement ($m = 1$). It is assumed one bi-directional link between each pair of nodes (redundancy is not allowed). Thus the potential links between every pair of nodes are the decision variables. Every decision variable x is composed of a tuple $(x_1, x_2, ..., x_n)$.

The constraints on redundancy and number of objectives are only apparent and do not make the problem less general, as the addition of new objectives is a trivial problem, even though it may require more computational resources. Also, redundant links can be treated as another kind of link, with its own cost and reliability [2]. Then, the design problem is stated as:

$$\begin{aligned} &\textit{Optimize} \quad \boldsymbol{y = f(x) = (f_1(x), f_2(x))} \qquad (2) \\ &\textit{subject to} \quad \boldsymbol{e_1(x) > 0} \end{aligned}$$

where:

- $\boldsymbol{x} = (x_1, x_2, ..., x_n) \in \boldsymbol{X}$ is the decision vector; every $x_i \{0, 1, ..., t\}$ represents a (type of) link between a pair of nodes and t is the number of different link types, while 0 is used to indicate the absence of connection;
- $\boldsymbol{y} = (y_1, y_2) \in \boldsymbol{Y}$ is the objective vector;
- $f_1(\boldsymbol{x})$ is the reliability corresponding to a configuration $\boldsymbol{x}$;
- $f_2(\boldsymbol{x})$ is the cost function of the same configuration $\boldsymbol{x}$;
- $e_1(\boldsymbol{x})$ refers to the minimum acceptable reliability.

Although parameters like delay, speed, capacity and throughput are important for innovative applications, the main network design objectives are still cost and reliability [2]. Both functions were studied in almost all papers found referring to design optimization problems. Sometimes, the problem was declared as the minimization of cost subject to a reliability constraint; while some others as the maximization of reliability subject to a cost constraint. Even some times [1] a weighted sum approach of both objectives was suggested. But the multi-objective nature of the problem has not been previously evidenced.

The concept of reliability depicts the probability of a system to have an expected performance over a time interval. So, the reliability of a system depends on its configuration and the reliability of its components. There are many methods and metrics to measure reliability. For our instance of the problem, to ensure that there is always a communication path between every pair of nodes in the network, the all-terminal reliability metric was chosen (i.e. the network forms at least a spanning tree) [2, 3]. The reliability calculation is done via Monte Carlo simulations because there are not other methods that can give good results in acceptable time (the problem of computing the reliability of a network is, in its context, NP-Hard [3]).

The cost of each configuration is calculated adding up the costs of every link added to the topology. Each link has a cost that is the product of the distance it covers and its cost per distance unit, given that only fiber optic links are considered in this paper. However, there is no difficulty in considering a given cost per link depending on the technology to be considered (microwave, satellite, etc.).

In order to solve the problem the following assumptions are necessary [8]:

- nodes are perfectly reliable (failure of nodes can be simulated by a failure of its incident links);
- the cost and reliability of each potential link are known;
- links can be in only one of two possible states: operational or failed;
- links fail independently, i.e. the failure in a link does not imply the failure of another one;
- no repair is considered, i.e. when a link fails it is not repaired and does not enter into operation afterwards.

3 Test Problem

The test problem is based on the expansion of the *ULAK-NET network*, first published in [8]. It is a simplified version of a real network design problem conceived to link, using distinct types of fiber, 19 universities and research centers located in 9 different cities of Turkey. It was chosen because it is the largest published example found during our research. Besides, the results of this example were available and they were used to compare with our experimental results.

The distance matrix in kilometers for each pair of nodes is presented in the Appendix (Matrix 1). Three types ($t = 3$) of fiber optic links are considered for each pair of nodes; their costs and reliabilities are (333 \$/km, 96%), (433 \$/km, 97.5%) and (583 \$/km, 99%) respectively. Then, the size of the search space is in the order of 10^{114} individuals of the form (x_1,…, x_{171}) with their corresponding cost and reliability.

4 Description of the Implementation

For the application of the *Multi-Objective Evolutionary Algorithms (MOEAs)*, each possible solution $\boldsymbol{x} = (x_1, x_2, ..., x_n)$ was coded using a string of integer numbers, $x_i \in \{0, 1, ..., t\}$. To obtain the string an adjacency matrix of the graph that models the network was written [14]. Since this matrix is symmetrical, only the upper triangular part was inserted into the chromosome. For example, to code the network of figure 1, the matrix of figure 2 was used.

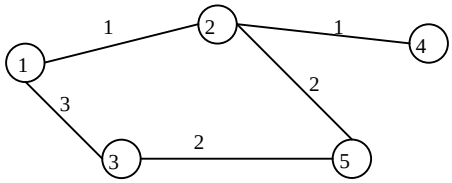

Figure 1. Graphical representation of a computer network backbone.

	1	**2**	**3**	**4**	**5**
1	0	1	3	0	0
2	1	0	0	1	2
3	3	0	0	0	2
4	0	1	0	0	0
5	0	2	2	0	0

Figure 2. Adjacency matrix for network of Figure 1.

The final representation x taken from the upper triangular part (and not considering the diagonal) is the string *1300012020*.

Continuing with the test problem, the calculation of reliability is accomplished with Monte Carlo simulations [2, 15]. Only 10000 replications were made due to the high computational cost. Solutions not achieving the minimum reliability requirement are not inserted in the external population, even though they may be part of the purported Pareto set. In any case, they remain in the current population because feasible children can be generated from them. Since it was formerly stated, the total cost of a network is the sum of the costs of its links.

Following the definition of SPEA [10], two populations of individuals are kept, the first one (depicted as *P*) is known as the current population, while the second one or external non-dominated set *P'* maintains every non-dominated individual found so far.

NSGA is implemented in two versions, the first one following the original formulation suggested by the authors [11], and the second one with an external non-dominated backup population *P'*, as was explained above.

The process of finding the non-dominated individuals in *P* is based on the concept of dominance [12]. We say that a solution χ_a dominates another solution χ_b if it is a better solution in at least one objective function without being worse in any other objective function. For example, if the solution has a better reliability without costing more. With this dominance concept, the implemented MOEAs implement elitism in the following way: every time a new non-dominated individual is found using an evolutionary algorithm, it is compared against the members in *P'*; if it is a new solution, it is inserted into *P'* erasing any dominated suboptimal solution that was kept in *P'*. The number of individuals in *P* is *N* and remains constant during the whole generational loop for both algorithms, but the number of individuals in *P'* may change from one generation to the other. To avoid a computer overflow, *P'* can not have more individuals than a previously stated number of *N'*. If the size of *P'* is greater than *N'* clustering should be performed to eliminate any danger of overflow. The process of clustering [10, 16] has been implemented, but was never used in our experimental computations because the maximum size of the external population was never reached.

SPEA and NSGA differ from the traditional genetic algorithm only in the way fitness is assigned to individuals. The computation of the fitness value follows the procedure explained in [10] and [11, 13], respectively.

In SPEA, every member of *P'* has a fitness equal to the number of individuals in *P* it dominates plus one; while every member of *P* has as its fitness the sum of fitness of the members of *P'* that dominates him. In this way, it is ensured that members of *P'* have a better fitness value than members of *P*. Notice that this is in the context of fitness minimization. The fitness assignment process, as well as clustering induces the maintenance of diversity [10].

In NSGA, fitness is assigned after a classification of individual into ranks. First of all, the non-dominated solutions are identified. All these non-dominated solutions belong to the first rank and the same high fitness value is assigned to them. To maintain diversity, these solutions undergo a fitness sharing procedure. After sharing their fitness value, the solutions of the first rank are temporarily ignored to continue with the classification of the other solutions. The same routine is applied again and a second level of ranks is determined. The solutions in this rank receive an original fitness value that is slightly lower than the worse fitness value assigned to the solutions of the first rank. Again this fitness is shared between all the individuals of this rank. This iterative process continues until an adequate fitness value is assigned to every member of the population.

Selection is implemented with binary tournaments, and the next generation is created via one point crossover. The mutation operator takes *m%* individuals from the population and changes every allele from its chromosome with probability 0.3.

The parameters of the algorithms used in our experiments are the following:

- Population size (*N*): 100 individuals.
- External non-dominated set size (*N'*): 100 individuals.
- Maximum number of generations (g_{max}): 10000.
- Crossover probability (p_c): 1.
- Mutation rate (r_m): 0.3.
- Percentage of population mutated in each generation (*m%*): 5%.

The initial population for the algorithm was generated probabilistically using a heuristic algorithm where individuals with fewer links have a greater probability of being inserted into the initial population. This approach has shown its usefulness to speed up convergence.

A stop criterion has also been implemented. The algorithm continues with its generational cycle if new individuals are being inserted into *P'* every 50 generations, or if the maximum number of generations (10000) has not been reached. Those numbers were chosen for the first implementation and proved to be very good for the test problem, but a complete study still should be done.

When the algorithm stops, it has its solutions in P', which is called the known Pareto set $\boldsymbol{X}_{known}$. The corresponding objective vectors $\boldsymbol{Y}_{known} = \boldsymbol{f}(\boldsymbol{X}_{known})$ is the known Pareto front.

5 Parallel Versions

Since the calculation of objective values, especially the computation of reliability, is extremely time consuming, the execution time of the proposed algorithms can be improved running them in a distributed environment. Moreover, the total implementation costs can be reduced significantly if we use a network of inexpensive personal computer instead of a massively parallel supercomputer.

The implementations for both algorithms consist of two kinds of processes, an organizer or master process and several slave processes. There is only one organizer, with the responsibility of creating all the slaves and collecting the final results. The slaves do the real work.

The Pseudocode1 is for the organizer process.

```
Procedure Organizer()
Begin
 Spawn H slaves
 flag_counter = 0
 While flag_counter < H
  Wait flag from H processes
  If a flag is received
    Collect results from process that sent a
    flag and kill him
    flag_counter=flag_counter+1
  End If
 End While
 Do union operation over sets obtained from
 slaves
 Apply Pareto dominance to obtain X_known
 Calculate Y_known
 Print final result.
End
```

Pseudocode 1. Organizer Procedure.

Straight away, the slaves are discussed. Given a distributed system with *H* processors, in each processor *h*, $h \in \{1, .. , H\}$, two populations are kept $P_h(g)$ and $P'_h(g)$. The population $P_h(g)$ contains the members generated by crossover in the

previous generation *g-1*; while $P'_h(g)$ is the external set of non-dominated solutions found from the beginning of the generational loop until generation *g* is reached.

Each processor *h* runs its own version of the evolutionary algorithm. Once new solutions for $P'_h(g)$ are found, at generation *g*, processor *h* broadcasts them to all the other processors. This procedure is known as *migration* and consists on sending (and receiving) good solutions known as migrants. The receiving processors accept all the migrants, as long as their memory capacity is not exceeded.

For the sequential version the population is composed of *N* individuals. As the parallel version is implemented in *H* identical processors, the size of each population P_h will be *N/H*. When migrants are received, the population grows; returning to its normal level after the genetic operators (as selection) is applied. The Pseudocode2 is for the Slave Procedure:

This parallelization scheme is completely different from other suggested approaches because it includes the independent evolution of many sub-populations that exchange information about good individuals, at every generation, in an asynchronous environment; while other proposals are based in the parallelization of portions of the generational loop, like the computation of objectives of the fitness assignment. Experimental results demonstrate that this new proposal is effective and efficient. This empirical conclusion is very important mostly because other suggested approaches have not yet been tested in any kind of problems.

For the parallelization of the NSGA without external population, the selected scheme is very similar, except for the selection of migrants, which is straightforward. In every generation, all non-dominated solutions are broadcasted. Again, all migrants are received and are placed in the current population. The size of the population is kept stable through the subsequent application of the selection genetic operator.

```
Procedure Slave()
Begin
 Read initial input parameters
 Read initial population P
 Gen_Count = 1
 While StopCondition is not reached and
 Gen_Count < g_max
  Compute values of objectives for each
    individual
  Receive migrants from other processes
    and add them to current population P
  Find non-dominated individuals in P
  Update external non-dominated set P'
  Broadcast new solutions from P'
  If number of external stored solutions
  exceeds N'
    Prune P' by means of clustering
  End If
  Calculate fitness of individuals in P and P'
  Select individuals from the union set P+P'
    until the mating pool is filled
  Generate new set P applying
  crossover & mutation
  Gen_Count = Gen_Count + 1
 End While
 Send flag to Organizer informing process
 is done
 Send individuals from P' to the Organizer
 Wait for a kill signal sent by the Organizer
End.
```

Pseudocode 2. Slave Procedure.

6 Performance Metrics

To evaluate experimental results of the two algorithms, an appropriate test suit metrics is used because no single metric can entirely capture performance, effectiveness

and efficiency for multi-objective evolutionary algorithms.

Since most of these metrics reflect the likeness between the true Pareto optimal front $\boldsymbol{Y}_{true}$ and the computed Pareto front $\boldsymbol{Y}_{known}$, a good approximation of the true Pareto optimal front is built by gathering all non-dominated individuals from all computed sets. In other words, for the following results, the real Pareto Optimal front is approximated by the best known solutions of all our experiments.

The test suit was taken from [12] and comprises the following metrics:

1) Overall *Non-dominated Vector Generation (ONVG)*, that simply counts the number of solutions in the Pareto front $\boldsymbol{Y}_{known}$

$$ONVG \stackrel{\Delta}{=} |Y_{known}|_c \tag{3}$$

where $|\ |_c$ denotes cardinality.

2) Overall true *Non-dominated Vector Generation (OTNVG)*: counts the number of solutions in the Pareto front $\boldsymbol{Y}_{known}$ that are also in the true Pareto optimal front $\boldsymbol{Y}_{true}$.

$$OTNVG \stackrel{\Delta}{=} \left|\{y \mid y \in Y_{known} \ \wedge \ y \in Y_{true}\}\right|_c \tag{4}$$

3) Overall *Non-dominated Vector Generation Ratio (ONVGR)*:

$$ONVGR \stackrel{\Delta}{=} \frac{ONVG}{|Y_{true}|_c} \tag{5}$$

It denotes the ratio between the number of solutions in $\boldsymbol{Y}_{known}$ to the number of solutions in the true Pareto front $\boldsymbol{Y}_{true}$. Since the objective is to obtain a set $\boldsymbol{Y}_{known}$ as similar to the true Pareto front as it is possible, a value near to 1 is desired.

4) Error Ratio (*E*):

$$E \stackrel{\Delta}{=} \frac{\sum_{i=1}^{N} e_i}{ONVG} \tag{6}$$

where:

$e_i = 0$ if a vector in Y_{known} is also in the true Pareto Front Y_{true}

$= 1$ *otherwise*

This ratio reports the proportion of objective vectors in $\boldsymbol{Y}_{known}$ that are not members of $\boldsymbol{Y}_{true}$. Therefore, an error ratio close to 1 indicates a poor correspondence between the obtained and the true Pareto front, i.e. $E = 0$ is desired.

5) Generational Distance (*G*) [12]:

$$G \stackrel{\Delta}{=} \frac{\left(\sum_{i=1}^{N} d_i^2\right)^{1/2}}{ONVG} \tag{7}$$

where d_i is a distance (in objective space) between each objective vector **F** in $\boldsymbol{Y}_{known}$ and its nearest correspondent member in the true Pareto front $\boldsymbol{Y}_{true}$. The Euclidean distance is recommended in [12]. A large value of G indicates $\boldsymbol{Y}_{known}$ is far from $\boldsymbol{Y}_{true}$ being $G = 0$ the ideal situation.

7 Experimental Results

The results presented here were obtained from successive runs over a 10 Mbps Ethernet network composed of up to 8 personal computers, each one with AMD K6-2 350 MHz processor, with 128 MB of RAM. The program code is entirely written in C, and the parallel implementation was done using PVM (Parallel Virtual Machine) running over LINUX (Mandrake 7.0).

A summary of our experimental results using SPEA and NSGA with external populations are shown in Tables 1 and 2,

where the first column identifies a given run, the second column gives the number of processors running in parallel, columns 3 to 6 presents the above defined performance metrics, column 7 gives the running time (in hours), while the last column presents a rank considering as performance metrics of a running columns 3 to 7, being rank 1 the set of optimal runs.

Table 1. Experimental Result and Performance Metrics of the SPEA algorithm for 10 runs, using P = 1, 2, 4 and 8 processors

#	P	ONVG	OTNVG	ONVGR	E	Time	Rank
1	1	41	0	0.323	1	8.64	7
2	1	42	0	0.331	1	8.712	7
3	1	46	0	0.362	1	8.95	7
4	1	46	0	0.362	1	8.45	6
5	1	51	0	0.402	1	9.003	5
6	1	44	3	0.346	0.932	8.35	6
7	1	43	1	0.339	0.977	8.96	7
8	1	51	0	0.402	1	8.472	5
9	1	51	2	0.402	0.961	8.269	3
10	1	57	0	0.449	1	8.726	4
11	2	45	4	0.354	0.911	5.946	5
12	2	47	6	0.370	0.872	5.267	4
13	2	52	0	0.409	1	5.002	4
14	2	56	2	0.440	0.964	5.637	3
15	2	57	0	0.449	1	5.891	3
16	2	41	5	0.323	0.878	5.236	4
17	2	47	7	0.370	0.851	5.189	3
18	2	50	2	0.394	0.96	4.968	3
19	2	54	0	0.425	1	4.256	3
20	2	46	3	0.362	0.935	4.781	3
21	4	49	1	0.386	0.980	2.78	3
22	4	52	1	0.409	0.981	2.64	2
23	4	56	2	0.441	0.964	2.859	2
24	4	41	10	0.323	0.756	3.05	1
25	4	59	3	0.465	0.949	2.956	2
26	4	47	3	0.370	0.936	2.567	3
27	4	49	8	0.386	0.837	2.368	2
28	4	54	3	0.425	0.944	2.945	2
29	4	53	2	0.417	0.962	2.847	2
30	4	51	12	0.402	0.765	2.369	1
31	8	55	0	0.433	1	1.498	1
32	8	50	1	0.394	0.98	1.486	1
33	8	49	6	0.386	0.878	1.56	1
34	8	56	11	0.441	0.804	1.689	1
35	8	53	9	0.417	0.830	1.67	1
36	8	61	3	0.480	0.951	1.547	1
37	8	47	2	0.370	0.957	1.689	1
38	8	51	6	0.402	0.882	1.487	1
39	8	58	5	0.457	0.914	1.694	1
40	8	59	4	0.465	0.932	1.567	1

In general, the ranks were built in such a way that any run of a given rank *k>1* is dominated by at least one run of rank (*k-1*).

Results using the original NSGA without an external population are not presented because it is clearly worse than its version with elitism.

Table 2. Experimental Result and Performance Metrics of the NSGA algorithm with external population for 10 runs, using P = 1, 2, 4 and 8 processors

#	P	ONVG	OTNVG	ONVGR	E	Time	Rank
1	1	32	4	0,224	0,875	11.894	8
2	1	41	0	0,287	1	9.956	10
3	1	29	0	0,203	1	9.036	3
4	1	31	5	0,217	0,839	10.894	11
5	1	45	0	0,315	1	11.563	7
6	1	44	0	0,308	1	10.547	8
7	1	34	2	0,238	0,941	11.451	12
8	1	36	3	0,252	0,917	11.354	5
9	1	44	1	0,308	0,977	10.256	5
10	1	46	2	0,322	0,957	11.378	6
11	2	35	0	0,245	1	5.946	2
12	2	44	6	0,308	0,864	6.784	2
13	2	42	8	0,294	0,810	5.781	8
14	2	33	14	0,231	0,576	6.124	8
15	2	31	0	0,217	1	5.787	3
16	2	45	8	0,315	0,822	6.003	5
17	2	39	0	0,273	1	5.961	9
18	2	42	5	0,294	0,881	5.649	7
19	2	50	0	0,350	1	5.891	2
20	2	43	2	0,301	0,953	5.678	7
21	4	39	2	0,273	0,949	2.78	2
22	4	46	11	0,322	0,761	3.208	6
23	4	51	8	0,357	0,843	2.859	3
24	4	47	0	0,329	1	3.05	1
25	4	39	3	0,273	0,923	2.956	2
26	4	37	5	0,259	0,865	3.103	3
27	4	45	6	0,315	0,867	2.864	2
28	4	35	0	0,245	1	2.945	4
29	4	39	4	0,273	0,897	2.847	2
30	4	45	0	0,315	1	2.965	5
31	8	39	0	0,273	1	1.598	2
32	8	47	6	0,329	0,872	1.678	1
33	8	52	3	0,364	0,942	1.697	3
34	8	46	0	0,322	1	1.7	3
35	8	43	18	0,301	0,581	1.764	4
36	8	41	4	0,287	0,902	1.767	1
37	8	42	3	0,294	0,929	1.77	1
38	8	39	1	0,273	0,974	1.801	2
39	8	48	5	0,336	0,896	1.807	4
40	8	56	4	0,392	0,929	1.853	1

Comparing tables 1 and 2, it is clear that the SPEA algorithm is the one with the best performance given that it founds the larger number of solutions (ONVG) with a fewer percentage of suboptimal solutions (not included in $\boldsymbol{Y}_{true}$), using for its calculation less time than its NSGA counterpart.

At the same time, parallelism proved to be very efficient in this context, as can be noted in tables 1 and 2, given that the average rank of a run improves with the number of processors for all the implemented algorithms.

8 Conclusions and Further Work

This paper first proposes the use of MOEAs to solve the network design problem. Several MOEAs alternatives were tested, being the SPEA the one with the best performance running in a network of personal computers, showing an excellent scalability in parallel executions. To run the experiments, a near real world test problem first published in [8] was used, obtaining a solution with a cost of 1,755,474 M$ and a reliability of 0.991, the best known so far (considering as best already published result the solution with a cost of 1,987,805 M$ and the same reliability [2]).

As future work, the authors are implementing more objective functions as maximum throughput, minimum delay, minimum number of links, between other objectives. At the same time, other MOEAs are being tested in the context of the network design problem, as well as a larger number of processors, especially in a heterogeneous network of computers and communication devices, as Internet.

References

1. Dengiz B., Smith A. B. and Altiparmak F. “Local search genetic algorithm for optimal design of reliable networks”. IEEE Transactions on Evolutionary Computation, Vol. 1, N° 3, September 1997.
2. Laufer F. and Barán B. “Topological Optimization of Reliable Networks using A-Teams”. In Proceedings of the 5th International Conference on Information Systems Analysis and Synthesis. Vol 5. Florida. 1999.
3. Colbourn C. J. Reliability Issues in Telecommunications Network Planning. Department of Computer Science, University of Vermont. Burlington. USA.
4. Tanembaum A. S. Computer Networks. Prentice-Hall, Englewood Cliffs, New Jersey 1981.
5. Boorstyn R. and Frank H. “Large-Scale Network Topological Optimization”. IEEE Trans. on communication, vol. COM-25, nº 1, pp. 29-47. 1977.
6. Pierre S. and Elgibaoui A. “A taboo search approach for designing computer network topologies with unreliable components”. IEEE Trans. on reliability, vol. 46, nº 3, pp. 350-359. 1997.
7. Pierre S., Hyppolite M. A., Bourjolly J. M. and Dioume O. “Topological design of computer communication networks using simulated annealing”. Engineering application of artificial intelligence, vol. 8, nº 1, pp. 61-69. 1995.
8. Deeter D. L., and Smith A. B. “Economic Design of Reliable Networks”. IIE Transactions, vol. 30, in print. 1999.
9. Konak A. and Smith A. “A Hybrid Genetic Algorithm Approach for Backbone Design of Communication Networks”. Proceedings of the 1999 Congress on Evolutionary Computation, Washington D. C., IEEE, 1999.
10. Zitzler E. and Thiele L. “Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach”. IEEE

transactions on evolutionary computation, vol 3, n° 4. November, 1999.
11. Deb K. Non-linear Goal Programming Using Multi-Objective Genetic Algorithms. Technical report TR CI-60/98, University of Dortmund, Germany: Department of Computer Science/XL 1999.
12. Van Veldhuisen D. A. Multiobjective Evolutionary Algorithms: Classifications, Analyses and New Innovations. PhD thesis, Department of Electrical and Computer Engineering. Graduate School of Engineering. Air Force Institute of Technology. Ohio, EE. UU. May, 1999.
13. Deb K. Evolutionary Algorithms for Multi-Criterion Optimization in Engineering Design. In Kasia Miettinen, Marko M. Mäkelä, Pekka Neittaanmäki, and Jacques Periaux, editors, Evolutionary Algorithms in Engineering and Computer Science, chapter 8. John Wiley & Sons, Chichester, U.K. 1999.
14. Horowitz E. and Sahni S. Fundamentals of Data Structures. Reprint edition. W H Freeman & Co. June 1983.
15. Jan R. H. "Design of Reliable Networks". Computers and operations research, vol. 20, nº 1, pp. 25-34. 1993.
16. Morse J. N. "Reducing the Size of the Nondominated Set: Pruning by Clustering. Comput. Oper. Res., vol 7, n° 1 y 2. 1980

Appendix

Matrix 1: Distance Matrix of the Test Problem (ULAK-NET 19 nodes design problem)

	v2	***v3***	***v4***	***v5***	***v6***	***v7***	***v8***	***v9***	***v10***	***v11***	***v11***	***v12***	***v13***	***v14***	***v15***	***v16***	***v17***	***v18***	***v19***
v1	*111*	*126*	*120*	*122*	*115*	*116*	*132*	*346*	*968*	*343*	*343*	*344*	*106*	*107*	*105*	*454*	*613*	*828*	*1261*
v2	-	*15*	*15*	*17*	*5*	*6*	*243*	*458*	*1079*	*454*	*454*	*456*	*10*	*11*	*5*	*565*	*724*	*939*	*1342*
v3		-	*15*	*17*	*13*	*14*	*258*	*473*	*1094*	*469*	*469*	*471*	*25*	*26*	*23*	*580*	*740*	*954*	*1357*
v4			-	*2*	*5*	*6*	*248*	*460*	*1082*	*456*	*456*	*457*	*12*	*13*	*15*	*570*	*730*	*943*	*1353*
v5				-	*8*	*9*	*251*	*463*	*1085*	*459*	*459*	*460*	*15*	*16*	*18*	*573*	*733*	*946*	*1355*
v6					-	*1*	*246*	*457*	*1080*	*454*	*454*	*455*	*10*	*9*	*12*	*568*	*728*	*940*	*1350*
v7						-	*245*	*456*	*1079*	*453*	*453*	*454*	*9*	*8*	*11*	*567*	*727*	*939*	*1351*
v8							-	*384*	*383*	*380*	*380*	*381*	*235*	*236*	*240*	*322*	*542*	*831*	*1301*
v9								-	*766*	*3*	*3*	*4*	*450*	*451*	*453*	*580*	*542*	*487*	*920*
v10									-	*763*	*763*	*764*	*1074*	*1075*	*1077*	*1345*	*1307*	*972*	*624*
v11										-	-	*1*	*450*	*451*	*453*	*582*	*544*	*489*	*921*
v12												-	*449*	*450*	*452*	*583*	*545*	*490*	*922*
v13													-	*1*	*4*	*560*	*720*	*932*	*1337*
v14														-	*3*	*561*	*721*	*933*	*1338*
v15															-	*563*	*723*	*934*	*1340*
v16																-	*469*	*898*	*1424*
v17																	-	*553*	*1079*
v18																		-	*526*