Evaluation of Different Optimization Techniques in the Design of Ad Hoc Injection Networks

Bernabé Dorronsoro, Grégoire Danoy, Pascal Bouvry, and Enrique Alba

Abstract— **Injection networks arise as a way to deal with the network partitioning problem in ad hoc networks. In this kind of networks, it is assumed that devices might have other communication interfaces rather than Wi-Fi and/or Bluetooth that allow them to connect to remote devices, such as GSM/UMTS. The problem considered in this work is to establish remote links between devices (called bypass links) in order to maximize the QoS of the network by optimizing its properties to make it small world. Additionally, these bypass links are not free, so the number of this kind of links in the network should be minimized as well. We face the problem with six different GAs and compare their behaviors. These alorithms are two panmictic algorithms, two GAs with the population structured in islands and two cellular GAs. One of the island GAs (a simple distributed GA with steady-state GAs runing in the islands) and the two cellular GAs were applied here for the first time to this problem. The other island GA, a cooperative coevolutionary GA, is considered the current state-of-the-art algorithm for this problem. As a result, we conclude that the two cellular GAs outperform all the compared algorithms, including the CCGA, for the three studied network instances.**

I. Introduction

Mobile multi-hop ad hoc networks most often face the problem of network partitioning. In this work we consider the problem of optimizing *injection networks* which consist in adding long-range links (e.g., using GSM, UMTS or HSDPA technologies) that are also called *bypass links* to interconnect network partitions. To tackle this topology control problem, we use small-world properties as indicators for the good set of rules to maximize the *bypass links* efficiency. Small-world networks [1] feature a high clustering coefficient (γ) while still retaining a small characteristic path length (L). On the one hand, a low characteristic path length is of importance for effective routing mechanisms as well as for the overall communication performance of the entire network. On the other hand, a high clustering coefficient features a high connectivity in the neighborhood of each node and thus a high degree of information dissemination each single node can achieve. This finally motivates the objective of evoking small-world properties in such settings. In order to optimize those parameters (maximizing γ, minimizing L) and to minimize the number of required bypass links in the network, we relied on Evolutionary Algorithms (EAs) and more specifically on Genetic Algorithms (GAs) [2].

This optimization problem was first introduced in [3], where it was solved with two panmictic GAs (generational and steady-state) and a cooperative coevolutionary GA (CCGA), this latter one reporting the best results. However, in that study the CCGA was compared versus two simple panmictic algorithms. In this work we extend this preliminar study by proposing three aditional decentralized GAs, namely a GA distributed in islands running a steady-state GA in each island, and two cellular genetic algorithms: a canonical and a hierarchical one. We consequently compare two panmictic GAs and four decentralized GAs (two island and two cellular GAs, the main kinds of structured GAs) on this complex problem. One important contribution of this paper is the comparison we perform among the CCGA versus other decentralized GAs. As an additional contribution of this work, we found that the two cellular GAs generally outperformed all the compared algorithms, representing the new state of the art for the problem.

The remainder of this paper is organized as follows. In the next section we introduce the injection network problem. Section III provides a brief description of the studied genetic algorithms, as well as the representation used and the fitness function we defined. Then, section IV presents the experiments and discuss the results. The last section contains our conclusions and perspectives.

II. Problem Overview

The problem we study in this article consists in overcoming partitioning in ad hoc networks by optimizing the placement of long range links that we call *bypass links*.

Our initial motivation for the current investigation is based on the assumption that technologies like Bluetooth and Wi-Fi can be used to create ad hoc communication links within the transmission range at no charge. Additional cellular network links such as GSM/UMTS/HSDPA might be employed by appropriately equipped devices to establish supplementary communication links, that we call *bypass links*, between two capable devices. These links will induce additional costs, and they are typically used to connect distant (not in range) devices that are either far away (there is a many-hops communication between them) or not connected (belonging to different clusters) in the network. Practically, a bypass link can be built by using a cellular network as well as by using access points. Nevertheless, in our model a bypass link is counted as a single hop, thus simplifying the real topology behind that bypass link. Devices used for establishing bypass links are called injection points, and self-organizing communication networks based on bypass links and injections points as described here are called *injection networks* (see an example injection network in Fig. 1).

Proceedings of the 2008 High Performance
Computing & Simulation Conference ©ECMS
Waleed W. Smari (Ed.)
ISBN: 978-0-9553018-7-2 / ISBN: 978-0-9553018-6-5 (CD)

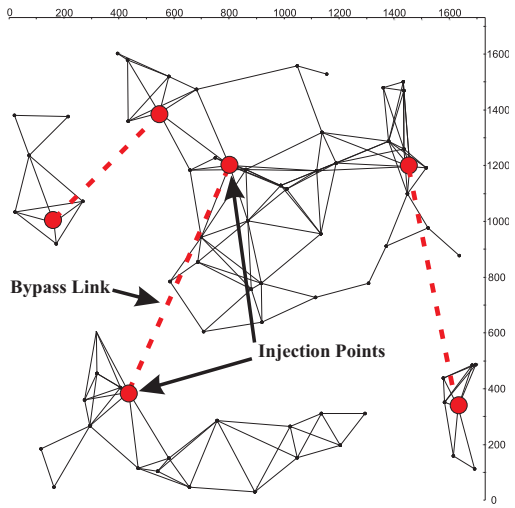

Fig. 1. Example of an Injection Network.

Injection points serve two different purposes: a point where information dissemination starts and where services are being placed (service placement, Herrmann et al. [4]). In the first case, the injection point is of essential importance at the moment of receiving information and passing this information to the neighborhood. The injection point might represent a bottleneck, depending on the amount of data passing through. In addition, injection points become particularly attractive when offering a service. In fact, information dissemination can be seen as such a service that is usable by devices in the injection points surrounding. Hence, the criterion for determining the injection point might highly influence the behavior of the network.

In order to optimize this kind of networks, we consider small-world properties as indicators for the good set of rules to maximize the bypass links efficiency. Small-World networks [1] are a class of random graphs that exhibit a small characteristic path length (L), indicating the degree of separation between the nodes in the graph, and a high clustering coefficient (γ), defining the extent to which nodes in the graph tend to form closely-knit groups that have many edges connecting each other in the group, but very few edges leading out of the group. The challenging aspect in using small-world properties is that small-world networks combine the advantages of regular networks (high clustering coefficient) with the advantages of random networks (low characteristic path length). In order to study the small-world properties of such hybrid networks, we had to rely on some ad hoc network simulator. In our case we used Madhoc [5], an application-level network simulator dedicated to the simulation of mobile ad hoc networks. The main motivation for using Madhoc is its ability to simulate hybrid networks, i.e., mixing different technologies (e.g., bluetooth/Wi-Fi for local connections and UMTS for long distance calls), and its graphical and batch modes of visualization, which greatly help in understanding the network design alternatives.

III. The Algorithms

We compare in this work several algorithms with different population structures (panmictic, cellular, and islands) on the complex problem of optimizing the design of injection ad hoc networks. On the one hand, panmictic algorithms do not consider any structure into the population; so any individual can mate with any other one in the population. On the other hand, in structured, or also called decentralized, populations (e.g., cellular and distributed GAs) individuals can only interact with a subset of the individuals in the whole population. In cellular GAs, a distance measure is defined among all the individuals in the population and only individuals that are close each other can interact. In the case of island (or distributed) GAs, the population is partitioned into several smaller subpopulations that independently evolve, exchanging some information among them during the run.

We present in this section the six algorithms evaluated in this study. Specifically, they are two panmictic GAs, the generational (genGA) and steady-state GAs (ssGA), two cellular ones, a canonical (cGA) and a hierarchical one (HcGA), and two island GAs, a canonical one (dGA) and a coevolutionary GA (CCGA). All these algorithms are briefly described in sections III-A to III-C, while the problem representation and the fitness function are presented in Section III-D.

A. Panmictic Genetic Algorithms

We present in this section the two GAs with centralized population we study in this paper, namely the steady-state (ssGA) and the generational (genGA) genetic algorithms. In panmictic algorithms, any individual in the population can mate with any other one during the breeding loop. These two algorithms perform in a similar way: they iterate a process in which two parents are selected from the whole population with a given selection criterion, they are then recombined, the obtained offsprings are mutated, and finally they are evaluated and inserted back into the population following a given criterion.

The difference between the ssGA and the genGA is the way in which the population is being updated with the new individuals generated during the evolution. In the case of the ssGA, new individuals are directly inserted into the current population (it is a (μ+1)-GA), while in the case of the genGA, a new auxiliary population is built with the obtained offsprings and then, once this auxiliary population is full, it completely replaces the current population (it is a (μ, λ)-GA, with $\mu = \lambda$). Thus, in ssGAs the population is asynchronously being updated with the newly generated individuals, while in the case of genGAs all the new individuals are updated at the same time, in a synchronous way.

B. Cellular Genetic Algorithms

Cellular genetic algorithms (cGAs) [6] are a kind of GA with a structured population in which individuals are spread in a two dimensional toroidal mesh, and they are only allowed

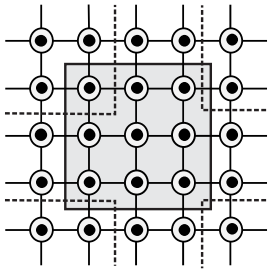

Fig. 2. Example 5×5 population of a cGA with C9 neighborhood

to interact with their neighbors. As an example, we show in Fig. 2 the disposition of the individuals in the population of a cGA, the neighborhood of the center individual (shadowed), and of another individual far from the center, in the upper left corner (dashed line).

A canonical cGA follows the pseudo-code included in Algorithm 1. In this basic cGA, the population is usually structured in a regular grid of d dimensions ($d = 1, 2, 3$), and a neighborhood is defined on it. The algorithm iteratively considers as current each individual in the grid (line 3), and individuals may only interact with individuals belonging to their neighborhood (line 4), so parents are chosen among the neighbors (line 5) with a given criterion. Crossover and mutation operators are applied to the individuals in lines 6 and 7, with probabilities P_c and P_m, respectively. Afterwards, the algorithm computes the fitness value of the new offspring individual (or individuals) (line 8), and inserts it (or one of them) instead of the current individual in the population (line 9) following a given replacement policy. This loop is repeated until a termination condition is met (line 2).

Algorithm 1 Pseudocode for a canonical cGA

```
 1: proc Evolve(cga)        //Algorithm parameters in 'cga'
 2: while ! StopCondition() do
 3:    for individual ← 1 to cga.popSize do
 4:       n_list←Get_Neighborhood(cga,position(individual));
 5:       parents←Selection(n_list);
 6:       offspring←Recombination(cga.Pc,parents);
 7:       offspring←Mutation(cga.Pm,offspring);
 8:       Evaluation(offspring);
 9:       Add(position(individual),offspring,cga);
10:    end for
11: end while
12: end proc Steps_Up;
```

In addition to the previously described canonical cGA, we also study in this paper a hierarchical version of the algorithm, called hierarchical cGA (HcGA) [7]. HcGA is a simple extension of canonical cGAs in which the population structure is augmented with a hierarchy according to the current fitness of the individuals. The basic idea is to put the best current solutions all together in the same region of the population, so that high quality solutions are exploited quickly, while at the same time new solutions are provided by individuals outside this region that keep exploring the search space. This algorithmic variant is expected to increase the convergence speed of the cGA algorithm and to maintain the diversity given by the distributed layout. In [7], the HcGAs were proposed for the first time and were validated in a theoretical and empirical comparison versus an equivalent canonical cGA.

C. Distributed Genetic Algorithms

In addition to the cellular model, there is another common way for structuring the population of GAs. It consists of splitting the whole population into several subpopulations in which isolated GAs are evolving, and these subpopulations exchange some information among them during the run. We study in this paper two algorithms following this model, namely dGA, a simple distributed GA with an ssGA running in every island, and CCGA, a cooperative coevolutionary GA that represents current state of the art for this problem.

The main idea behind coevolutionary algorithms is to consider the coevolution of subpopulations of individuals representing specific parts of the global solution, instead of considering a population of similar individuals representing a global solution, like classical genetic algorithms do. The quality of this kind of algorithms have been reported in a large number of papers in the literature. As an example, two different coevolutionary GAs were applied in [8] on a number of test functions known in the area of evolutionary computation, and they were demonstrated to clearly outperform a sequential GA.

Cooperative (also called symbiotic) coevolutionary genetic algorithms (CCGA) involve a number of independently evolving species which together form complex structures, well-suited to solve a problem (see a pseudocode in Algorithm 2). The fitness of an individual depends on its ability to collaborate with individuals from other species. In this way, the evolutionary pressure stemming from the difficulty of the problem favors the development of cooperative strategies and individuals. The CCGA considered here is based in the model proposed by Potter and DeJong [9], in which a number of populations explore different decompositions of the problem. In this system, each species represents a subcomponent of a potential solution. Complete solutions are obtained by assembling representative members of each of the species (populations). The fitness of each individual depends on the quality of (some of) the complete solutions it participated in,

Algorithm 2 Pseudocode of the CCGA

```
 1: gen = 0
 2: for all species_s do
 3:    Pop_s(gen) = randomly initialized population
 4:    evaluate fitness of each individual in Pop_s(gen)
 5: end for
 6: while termination condition = false do
 7:    gen = gen + 1
 8:    for all species_s do
 9:       select Pop_s(gen) from Pop_s(gen − 1) based on fitness
10:       apply genetic operators to Pop_s(gen)
11:       evaluate fitness of each individual in Pop_s(gen)
12:    end for
13: end while
```

thus measuring how well it cooperates to solve the problem. The evolution of each species is controlled by a separate, independent evolutionary algorithm. In the initial generation *(t=0)* individuals from a given subpopulation are matched with randomly chosen individuals from all other subpopulations. A fitness for each individual is evaluated, and the best individual in each subpopulation is found. The process of *cooperative coevolution* starts form the next generation *(t=1)*. For this purpose, in each generation a cycle of operations is repeated in a round-robin fashion. Only one current subpopulation is active in a cycle, while the other subpopulations are frozen. All individuals from the active subpopulation are matched with the best values of frozen subpopulations. When the evolutionary process is completed a composition of the best individuals from each subpopulation represents a solution of a problem.

D. Problem Encoding and Fitness Functions

Solution encoding is a major issue in this kind of algorithms since it will determine the choice of the genetic operators applied for exploring the search space. We have used a binary encoding of the solution in which each gene encodes an integer on 15 bits, that corresponds to one possible bypass link in the half-matrix of all possible links. For instance, if the maximum number of bypass links fixed a priori for the network that is optimized is 10, then a chromosome will have 10 genes of 15 bits. Figure 3 shows the example of a chromosome composed of 2 genes (thus the maximum number of created bypass links is 2) on a network of 5 stations. The 5×5 half-matrix represents all the possible links in the network including the already existing local links in the network (i.e. the existing Wi-Fi connections). In the example showed in Figure 3, the first gene (circled) with the integer value 2 stands for the connection between station 1 and station 3 in the corresponding half-matrix (also circled).

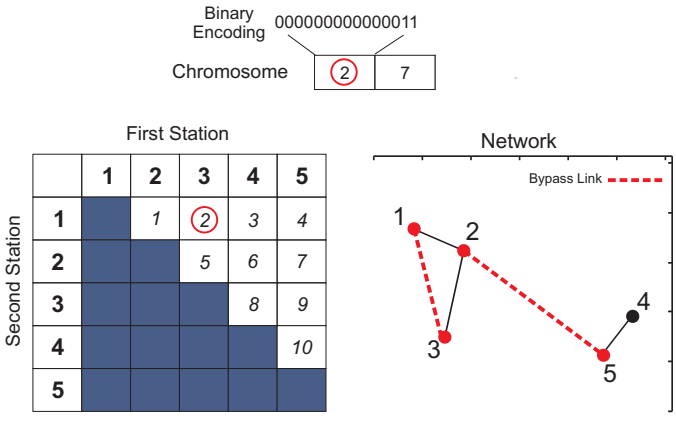

Fig. 3. Solution encoding example

In order to assign a fitness value to the candidate solutions (i.e. sets of possible bypass links) of our algorithms, we use a unique cost function F which combines the two small world measures (L and γ) and the number of created bypass links. The calculation of the characteristic path length L imposes that there exists a path between any given nodes a and b. Consequently, the computation of the fitness function requires that we first test if the network is partitioned.

If the optimized network is still partitioned (the bypass links defined do not achieve to connect all the partitions), the fitness value is assumed to be a weighted term of the number of partitions in the network.

On the contrary, if the optimized network is no longer partitioned, the fitness value is assumed to be a linear combination of the clustering coefficient, of the characteristic path length, and of the difference between the number of bypass links defined and the maximum number allowed.

The aim of the optimization process is to maximize the clustering coefficient, and to minimize both the characteristic path length and the number of bypass links. By using this fitness function we now face the maximization problem defined in Algorithm 3.

Algorithm 3 Fitness Function

1: **if** Graph connected **then**
2: F = α * γ - β * (L - 1) - δ * (bl - bl_{max})
3: **else**
4: fitness = ξ * P
5: **end if**

With weights experimentally defined:
α = 1
β = 1 /(N -2)
δ = 2 / (N * (N-1)) - WifiConnections
ξ = 0.1

where bl is the number of bypass links created in the simulated network by one solution, bl_{max} (defined a priori) is the maximum number of bypass links that can be created in the network, P is the number of remaining partitions in the whole network after the addition of bypass links and N is the number of stations in the global network. Finally, WifiConnections is the number of existing Wi-Fi connections in the network.

IV. Experiments

This section presents the results obtained on the injection network optimization problem using the different GAs presented in Section III. We first describe the parameters used for the genetic algorithms. Next, the configuration of the network simulator is introduced and, finally the results obtained using the six GAs are analyzed and compared.

A. Parameterization

In Table I, we show the parameters used for all the proposed algorithms. All of them have a single population of 100 individuals, except for the two distributed algorithms: CCGA (10 populations of 50 individuals), and dGA (5 subpopulations of 100 individuals), having both of them a total population of 500 individuals. The termination condition is achieving 50,000

TABLE I

PARAMETERS USED FOR THE STUDIED GAS

Number of Subpopulations	10 for CCGA
	5 for dGA
(Sub)Population size	100 (genGA, ssGA, dGA)
	10×10 (cGA, HcGA)
	50 (CCGA)
Termination Condition	50,000 function evaluations
Selection	Binary tournament (BT)
	Current individual + BT in cGA and HcGA
Neighborhood	C9 in cGA
	C13 in HcGA
Crossover operator	DPX, p_c=0.8
Mutation operator	bit flip, p_m = 1/chrom_length
Elitism	1 individual (not for ssGA)

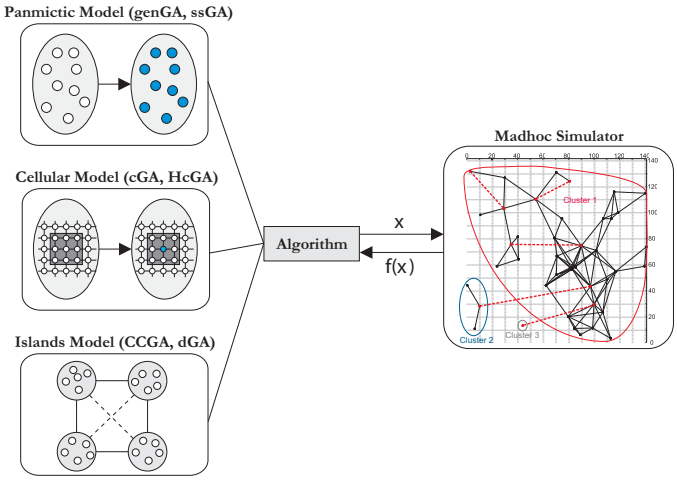

Fig. 4. Components of the experimental study

fitness function evaluations, common to all the algorithms, as well as the recombination (the two points crossover –DPX–) and mutation (bit-flip) operators, and their probabilities: $p_c = 0.8$ and $p_m = 1/\text{chrom\_length}$, respectively.

The two parents are selected using a binary tournament, except for the two cellular algorithms, for which one of them is considered to be the current individual itself. A specific parameter of these cellular models is the neighborhood. We used C9 (9 closest individuals measured in Manhattan distance –see Fig.2–) for cGA and C13 for HcGA. The reason of using a different neighborhood for HcGA is that it maintains a higher diversity in the population than the cGA, and thus we can use a more exploitative neighborhood. Finally, all the algorithms follow an elitist strategy, with the exception of ssGA.

B. Madhoc Configuration

As stated before, the Madhoc simulator was used for managing the complex scenarios posed by this injection network problem. Fig. 4 shows how the genetic algorithms interact with Madhoc.

The parameters we have used in Madhoc for defining our problem instances are shown in Table II. We have defined a square simulation area of 0.2 km^2 and tested three different densities of 150, 210 and 350 devices per square kilometer. Each device is equipped with both Wi-Fi (802.11b) and UMTS technologies. The coverage radius of all mobile devices ranges between 20 and 40 meters in case of Wi-Fi.

TABLE II

PARAMETERIZATION USED IN MADHOC

	1 Cluster	3 Clusters	5 Clusters
Surface	0.2 km^2	0.2 km^2	0.2 km^2
Node Density	350 nodes/km^2	210 nodes/km^2	150 nodes/km^2
Number of Nodes	70	42	30
Partitions	1	3	5
Possible Links	2189	745	400

The studied networks, as presented in Fig. 5, here represent a snapshot of mobile networks in the moment in which a single set of users moved away from each other creating the clusters of terminals, that were obtained using the graphical mode of Madhoc. As an example, the network with 3 clusters (center of Fig. 5) consists in 42 stations located in three partitions, the first partition has 38 stations, the second one 3, and the third one has a single station. The number of possible connections in this 3-clusters network is $\frac{N*(N-1)}{2}$ = 861. The number of existing Wi-Fi connections in this network is 116, thus the number of possible bypass links is 861-116 = 745. The clusters are selected purposely to be different and thus challenging.

C. Results

In Table III we show the averaged results, the best ones, and the total computational time for all 30 runs for each algorithm. Additionally, we show the results of the statistical tests in the comparison of the different algorithms for each cluster instance in order to obtain concluding results from the comparison made. For performing these statistical tests, we first check whether the data follow a normal distribution or not using the Shapiro-Wilks test. Then, if the data are normally distributed we perform an ANOVA test. In the other case, we use the Kruskal-Wallis test. This statistical study allows us to assess if there are meaningful differences among the compared algorithms with 95% probability or not.

Symbol '+' in Table III stands for existing statistical differences in the comparison of the algorithms. Grey background means that the result is the best one with statistical confidence (if there are more than one result with grey background for the same problem it means that there are no statistical difference between them, but they are better than the others with statistical significance).

As we can see in Table III, the two cellular models and the CCGA are clearly the three best compared algorithms. The two panmictic GAs and dGA obtain the worst results with statistical significance with respect to the best performing algorithm for the three studied instances. If we now compare the CCGA and the two cellular models, we can see that the celullar alorithms outperform CCGA in the case of the 5 clusters instance with statistically significant differences. For the other two instances (1 and 3 clusters), CCGA obtains better

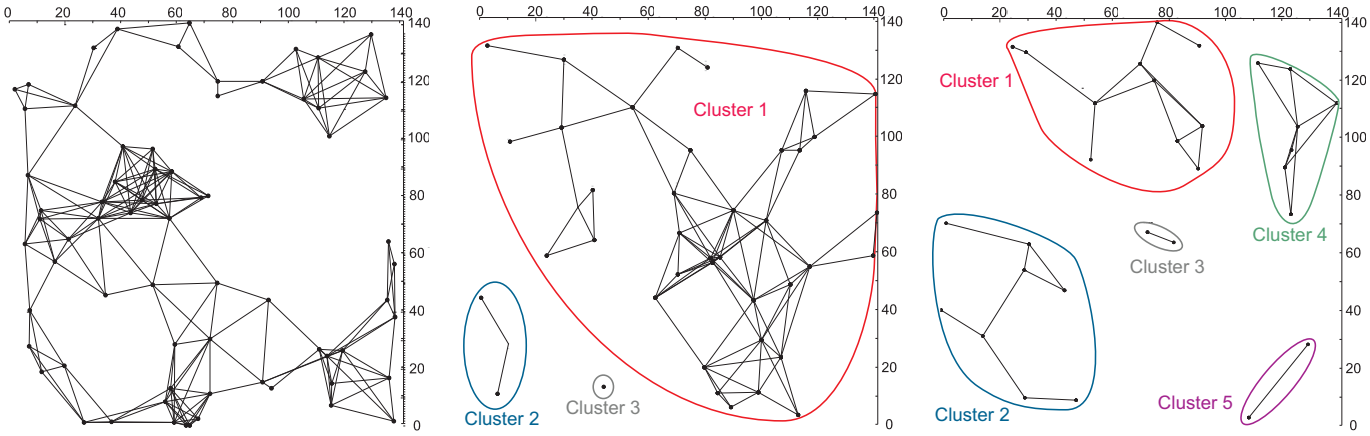

Fig. 5. Studied Networks with 1, 3 and 5 clusters

TABLE III
RESULTS OF ALL EXPERIMENTS

Network	GA	Avg. Result	Best Result	Time (s)	p-value
1 Cluster	genGA	0.6833	0.6920	**8256**	+
	ssGA	0.6736	0.6855	8395	
	dGA	0.6760	0.6856	11311	
	CCGA	**0.6911**	0.6925	17784	
	cGA	0.6887	0.6924	10760	
	HcGA	0.6893	**0.6926**	10366	
3 Clusters	genGA	0.6685	0.6782	4486	+
	ssGA	0.6489	0.6651	3289	
	dGA	0.6555	0.6664	5703	
	CCGA	**0.6739**	**0.6757**	6793	
	cGA	0.6727	**0.6757**	3563	
	HcGA	0.6724	0.6754	**2883**	
5 Clusters	genGA	0.5634	0.5848	1672	+
	ssGA	0.5527	0.5809	1717	
	dGA	0.5576	0.5783	3563	
	CCGA	0.5652	0.5864	8674	
	cGA	**0.5790**	0.5923	1713	
	HcGA	0.5779	**0.5931**	**1569**	

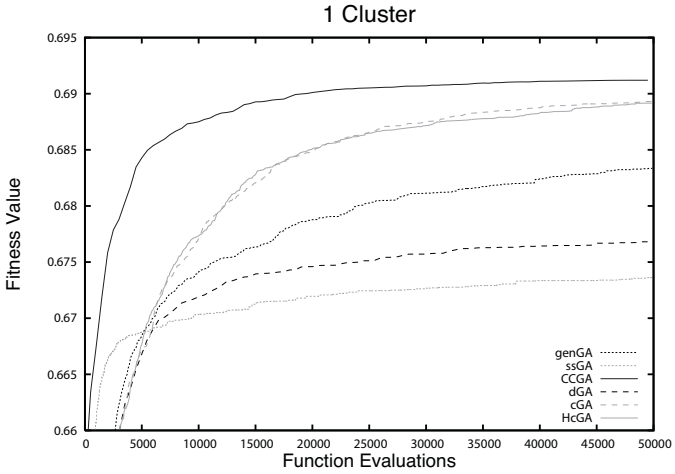

Fig. 6. Evolution of the average best fitness value (30 executions) during the run. One cluster instance

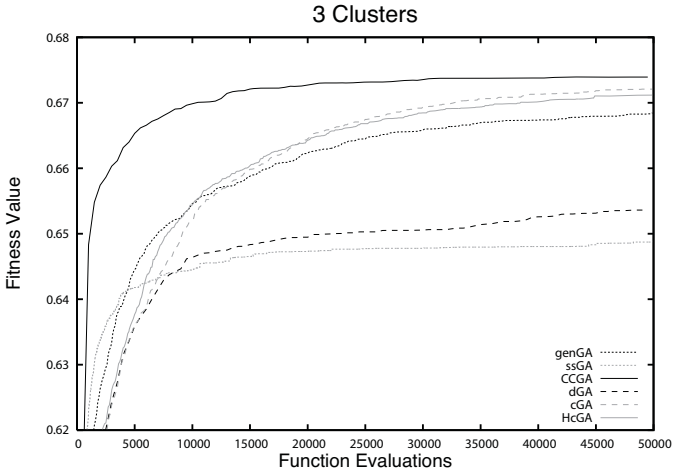

Fig. 7. Evolution of the average best fitness value (30 executions) during the run. Three clusters instance

average results, but the differences with the cellular models are not statistically significant. If we now pay attention to the best solution found in the 30 runs, HcGA finds the best result for instances of 1 and 5 clusters, while in the case of the 3 clusters problem, both the cGA and CCGA achieve the same best value.

Regarding the computation time, the HcGA is the fastest algorithm among the three best ones. Indeed it is the fastest algorithm out of the six compared ones for instances of 3 and 5 clusters, being only improved by the two panmictic algorithms (wich find much worse results) in the largest instance.

With the goal of better understanding the behavior of the different compared algorithms, we now analyze the evolution of the population during the run for all the alorithms. So, we plot in figures 6 to 8 the evolution of the average of the best fitness values during the execution in the 30 runs for the six algorithms and the three studied problem instances. As it can be seen, the behavior of the algorithms is similar for the three problem instances.

As a general rule, all the algorithms experiment a high improvement of the fitness value during the first function evaluations, but then they get stuck and the improvement of the fitness is very low. The reason for this too slow evolution of the fitness value is that the population (or populations) of the

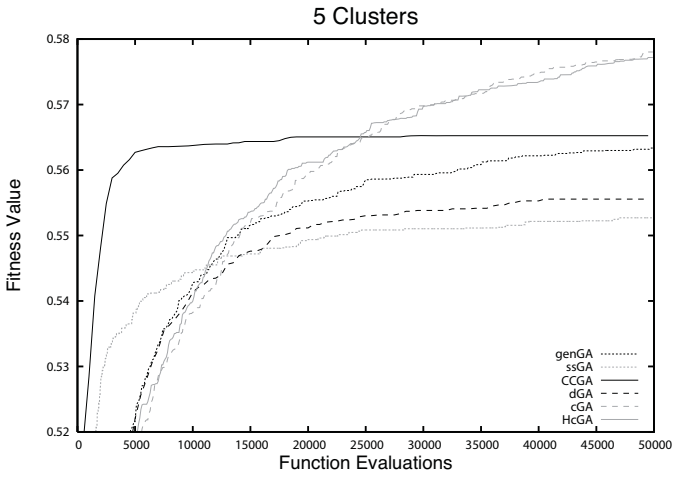

Fig. 8. Evolution of the average best fitness value (30 executions) during the run. Five clusters instance

algorithms prematurely converged in the first steps of the run, loosing the diversity of individuals, and thus making difficult the improvement of the current individuals with the application of the variation operators.

As an exception, the reader can see how the convergence of the cellular GAs is slower from the beginning of the evolution, and it does not get stuck as in the case of the other compared algorithms. The reason is that the cellular model preserves the diversity of the population for longer [6] with respect to the other compared GAs. It thus makes possible to improve the current solutions with the application of the variation operators to the individuals in the population during the breeding loop. We can clearly see this effect in Fig. 8: the fitness value is continuously growing during the whole evolution for the two cellular models, while the other algorithms hardly improve the fitness value in the second half of the run.

The CCGA experiments a really fast convergence during the first generations, but after that the convergence becomes much slower, slightly improving the solution. The two cellular models show a similar behavior in the three problem instances. The convergence of these two algorithms is slower than for the other ones, consequently they need more funtion evaluations to achieve good results, but they maintain the diversity of the population for longer avoiding local optimal solutions.

Finally, it can be seen how the island and the two panmictic GAs prematurely converge to local optimal solutions, from which they find difficulties to escape.

V. Conclusions and Further Works

We have compared in this paper six different GAs on the problem of designing ad hoc injection networks. The compared algorithms are two panmictic algorithms, the generational and steady-state GAs, two algorithms with population structured in islands, a distributed GA with steady-state GAs in the islands and a coevolutionary GA, and two cellular GAs: the canonical and the hierarchical cGA.

Our main conclusion from our comparison study is that the cellular models outperform the other compared ones for the three studied instance problems. The CCGA reports similar average results than the cellular algorithms for the two largest intances, but the cellular models are better in terms of the best solution found and the computational time required. As future works, we propose the study of more realistic instances of the problem. This can be achieved by either considerably increasing the size of the problems or by studying dynamic networks varying with time.

References

[1] D. J. Watts, *Small Worlds – The Dynamics of Networks between Order and Randomness*. Princeton, New Jersey: Princeton University Press, 1999.

[2] T. Bäck, D. Fogel, and Z. Michalewicz, Eds., *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[3] G. Danoy, E. Alba, P. Bouvry, and M. R. Brust, "Optimal design of ad hoc injection networks by using genetic algorithms," in *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2007, pp. 2256–2256.

[4] K. Herrmann and K. Geihs, "Self-Organization in Mobile Ad hoc Networks based on the Dynamics of Interaction," Erlangen, Germany, 2003, frühjahrstreffen der GI-Fachgruppe Betriebssysteme. [Online]. Available: http://www.kbs.cs.tu-berlin.de/publications/fulltext/gi0403.pdf

[5] L. Hogie, P. Bouvry, F. Guinand, G. Danoy, and E. Alba, "Simulating Realistic Mobility Models for Large Heterogeneous MANETS," in *Demo proceeding of the 9th ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM'06)*. IEEE, October 2006.

[6] E. Alba and B. Dorronsoro, *Cellular Genetic Algorithms*, ser. Operations Research/Compuer Science Interfaces. Springer-Verlag Heidelberg, 2008.

[7] S. Janson, E. Alba, B. Dorronsoro, and M. Middendorf, "Hierarchical cellular genetic algorithm," in *Evolutionary Computation in Combinatorial Optimization (EvoCOP)*, ser. Lecture Notes in Computer Science (LNCS), J. Gottlieb and G. Raidl, Eds., vol. 3906. Budapest, Hungary: Springer-Verlag, Heidelberg, April 2006, pp. 111–122.

[8] F. Seredynski, A. Zomaya, and P. Bouvry, "Function optimization with coevolutionary algorithms," in *Proc. of the International Intelligent Information Processing and Web Mining Conference*. Springer, 2003.

[9] M. Potter and K. De Jong, "A cooperative coevolutionary approach to function optimization," in *Parallel Problem Solving from Nature (PPSN III)*. Springer, 1994, pp. 249–257.