Evaluation of Different Metaheuristics Solving the RND Problem

Miguel A. Vega-Rodríguez[1], Juan A. Gómez-Pulido[1], Enrique Alba[2], David Vega-Pérez[1], Silvio Priem-Mendes[3], and Guillermo Molina[2]

[1] Dep. of Computer Science, Univ. of Extremadura, Caceres, Spain
{mavega,jangomez}@unex.es, dvperez@indra.es
[2] Dep. of Computer Science, Univ. of Malaga, Malaga, Spain
{eat,guillermo}@lcc.uma.es
[3] Polytechnic Institute of Leiria, High School of Technology, Leiria, Portugal
smendes@estg.ipleiria.pt

Abstract. RND (Radio Network Design) is a Telecommunication problem consisting in covering a certain geographical area by using the smallest number of radio antennas achieving the biggest cover rate. This is an important problem, for example, in mobile/cellular technology. RND can be solved by bio-inspired algorithms. In this work we use different metaheuristics to tackle this problem. PBIL (Population-Based Incremental Learning), based on genetic algorithms and competitive learning (typical in neural networks), is a population evolution model based on probabilistic models. DE (Differential Evolution) is a very simple population-based stochastic function minimizer used in a wide range of optimization problems, including multi-objective optimization. SA (Simulated Annealing) is a classic trajectory descent optimization technique. CHC is a particular class of evolutionary algorithm which does not use mutation and relies instead on incest prevention and disruptive crossover. Due to the complexity of such a large analysis including so many techniques, we have used not only sequential algorithms, but grid computing with BOINC in order to execute thousands of experiments in only several days using around 100 computers. In this paper we present the most interesting results from our work, indicating the pros and cons of the studied solvers.

Keywords: RND, PBIL, DE, SA, CHC, Metaheuristics, Evolutionary Techniques.

1 Introduction

The Radio Network Design problem is a kind of telecommunication network design problem. When a set of geographically-dispersed terminals needs to be covered by transmission antennas (also called base station transmitters or base transceiver stations -BTS-), a capital subject is to minimize the number and locations of those antennas and to cover the largest area.

M. Giacobini et al. (Eds.): EvoWorkshops 2007, LNCS 4448, pp. 101–110, 2007.

RND is an NP-hard problem; therefore its solution by means of bio-inspired algorithms is appropriate. In this work we use several different metaheuristics in order to solve this problem: PBIL, DE, SA and CHC.

Finally, since our interest is not only studying these techniques, but also to open new research lines, we have applied grid computing for the realization of our experiments (not in an exhaustive manner for space constraints in this conference paper). In particular, we have used BOINC (Berkeley Open Infrastructure for Network Computing), a very interesting proposal for volunteer computing and desktop grid computing.

The rest of the paper is organized as follows: Section 2 briefly explains the RND problem. After that, in the following sections we introduce the PBIL, DE, SA and CHC algorithms. Then, in Section 7 we show the most interesting results of this work, including comparisons among the different studied techniques, finally leading to the conclusions and future work summarized in the last section.

2 The RND Problem

The RND problem [1,2] consists in covering a largest area with a minimal set of transmitters. In order to mathematically define this problem, let us consider the set L of all potentially covered locations and the set M of all potential transmitter locations. Let G be the graph, (M $\bigcup$ L, E), where E is a set of edges such that each transmitter location is linked to the locations it covers. As the geographical area needs to be discretized, the potentially covered locations are taken from a grid, as shown in Figure 1a. In our case, we focus on a 287×287 point grid representing an open-air flat area (a total of 82,369 different positions) and we will be able to use a maximum of 349 available locations for placing antennas. The vector **x** will be a solution to the problem where $x_i \in \{0,1\}$, and $i \in [1, 349]$, indicates whether a transmitter is used (1) or not (0) in the corresponding location.

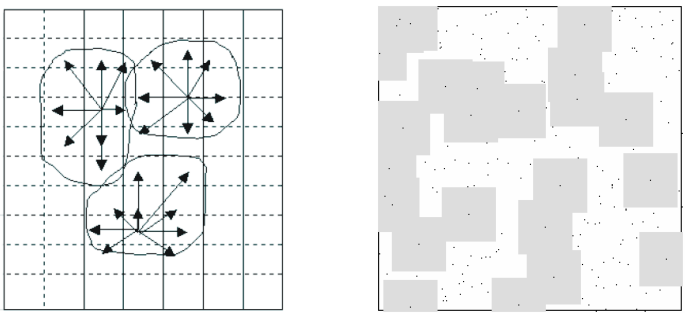

Fig. 1. (a) Three potential transmitter locations and their associated covered cells on a grid. (b) Base station transmitters with square coverage.

The objective of RND is to search for the minimum subset of transmitters that covers a maximum surface of an area, therefore, we are searching for a subset M' $\subseteq$ M such that |M'| is minimum and such that |Neighbours(M', E)| is maximum,

where:

$$\begin{gathered}\text{Neighbours}(M', E) = \{u \in L \mid \exists v \in M', (u, v) \in E\} \\ M' = \{t \in M \mid x_t = 1\}\end{gathered} \tag{1}$$

The fitness function we have used in our experiments is shown in Equation 2 [3].

$$f(x) = \frac{CoverRate(x)^2}{NumberTransmittersUsed(x)} \tag{2}$$

An important constraint in this problem consists in determining the list of available locations for the antennas, because there are some places where the antennas can not be placed (public recreation areas, etc.). In our case, for the predefined set of available locations we selected the one included in [4]. This will make easy the comparisons among the different evolutionary techniques.

In our experiments we consider base station transmitters with square cells (see figure 1b), each transmitter has an associated 41×41 point cell (coverage radius size is 20). Other cell shapes are possible (like circular cells for omnidirectional antennas) but deferred for future work. Let us now briefly present the used algorithms.

3 Population-Based Incremental Learning

Population-Based Incremental Learning (PBIL) is a method that combines a genetic algorithm with competitive learning for function optimization. Instead of applying operators, PBIL infers a probability distribution from the present population and samples the new population from the inferred distribution [5,6].

4 Differential Evolution

Differential Evolution (DE) is an algorithm used for continuous optimization problems in the past with satisfactory results [7,8]. DE is a simple population-based stochastic function minimizer/maximizer, used in a wide range of optimization problems, including multi-objective optimization [9]. It has been modified in this research to work with discrete representations [10].

5 Simulated Annealing

Simulated annealing (SA) is a generic probabilistic meta-algorithm for the global optimization problem, namely locating a good approximation to the global optimum of a given function in a large search space. It was independently invented by S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi in 1983 [11], and by V. Černý in 1985 [12]. SA is a trajectory based optimization technique (i.e., only one tentative solution is manipulated in contrast with the rest of algorithms here, where a population of solutions is used). It is commonly found in industry and provides good results; therefore it constitutes an interesting method for comparison.

6 CHC

The fourth algorithm we propose for solving the RND problem is Eshelman's CHC [13], which stands for Cross-generational elitist selection, Heterogeneous recombination (by incest prevention), and Cataclysmic mutation. CHC is a kind of Evolutionary Algorithm (EA), where mutation is not used. As a mechanism for preventing convergence and maintaining diversity, CHC employs incest prevention [14] plus a special recombination procedure known as HUX, combined with a special restarting mechanism for adding diversity through mutation when stagnation is detected (cataclysmic mutation).

7 Results

In this section we present the most interesting results coming from using each of the evolutionary techniques we have proposed to solve the RND problem.

7.1 Results Using PBIL

In this work we have first evaluated different configuration parameters for our PBIL algorithm in order to solve the RND problem since this application to this problem is relatively new. In particular, the parameters we can adjust in PBIL are: used variant of PBIL (we have 7 variants explained in the next paragraphs), number of samples (the population size), mutation probability, mutation shift (intensity of the mutation that affects the probability vector), learning rate, negative learning rate (only valid for the variant PBIL-NegativeLR), number of M best individuals (only valid for the variants PBIL-M-Equitable, PBIL-M-Relative and PBIL-M-Consensus), and whether an elitist strategy is used or not (the best individual in the previous population is transferred to the current generation unaltered).

The variants of PBIL we have thus studied are the following:

- PBIL-Standard.
- PBIL-Complement: The probability vector is moved towards the complement of the lowest evaluation individual (the worst sample) in each generation.
- PBIL-Different: The probability vector is only moved towards the bits in the best individual which are different than those in the worst individual, in each generation.
- PBIL-M-Equitable: The probability vector is moved equally in the direction of each of the M selected individuals (M best samples) in each generation.
- PBIL-M-Relative: The probability vector is moved on the relative evaluations (fitness functions) of the M best individuals in each generation.
- PBIL-M-Consensus: The probability vector is moved only in the positions in which there is a consensus (the same value) in all of the M best individuals in each generation.
- PBIL-NegativeLR: The probability vector is moved towards the best individual (using the learning rate) and also away from the worst individual (using the negative learning rate) in each generation.

For the rest of configuration parameters we present a wide analysis including the following values (most typical in literature):

- Number of generations: 1000, 2500, 5000, 7500 and 10000.
- Number of samples (the population size): 50, 75, 100, 135, 170 and 200.
- Mutation probability: 0.01, 0.02, 0.03, 0.05, 0.07, 0.09 and 0.10.
- Mutation shift: 0.01, 0.03, 0.04, 0.05, 0.07, 0.09 and 0.10.
- Learning rate:0.01, 0.05, 0.10, 0.20, 0.30 and 0.40
- Negative learning rate: 0.01, 0.02, 0.03, 0.05, 0.07, 0.09 and 0.10.
- Number of M best individuals: 2, 3, 4, 5, 6 and 7.
- Elitism: Yes or no.

The total number of possible combinations is very high, for this reason we have used the middleware system BOINC [15,16] (Berkeley Open Infrastructure for Network Computing) in order to perform massive computations/experiments in a parallel way for PBIL. BOINC is a system for volunteer computing and desktop grid computing. Volunteer computing uses computers volunteered by the general public to do distributed scientific computing.

In our case, we use BOINC in order to perform many different executions of our evolutionary algorithm in parallel. In this way, we can do a deep survey about which are the best parameter values for solving the RND problem with PBIL, which is needed to guide future research after this first study. Researchers interested in learning more about our platform RND-BOINC (RND@home), or wanting to join in this project (volunteer donation of CPU cycles), can access it via the website http://arcoboinc.unex.es/rnd. At present, around 100 computers are available in this project, executing hundreds of experiments at the same time in parallel.

Table 1. The most important results with PBIL for solving the RND problem

	Config. best result
PBIL variant	Standard
# Generations	2500
# Individuals	135
Mutation probability	0.05
Mutation shift	0.07
Learning rate	0.30
Negative learning rate	---
M best individuals	---
Elitism	YES

	Best result	Average results
Fitness function	204.082	204.082
# Transmitters	49	49
Coverage	100%	100%
Execution time	5 minutes, 34 seconds	5 minutes, 56 seconds
Execution on	Pentium IV – 2.8 GHz	
Normalized execution time (P.IV-1.7 GHz)	9 minutes, 10 seconds	9 minutes, 46 seconds
# Evaluations	276,345	306,855

Table 1 shows the most important results using PBIL. In particular, for every combination of parameters, 30 independent runs have been performed for statistical purposes. In this table, the normalized execution time converts the obtained execution time to a virtual standard Pentium IV-1.7 GHz. As we can see, PBIL solved the problem with 100% accuracy, and presents low variability between the best computational effort (276,345 evaluations) and the average (306,855 evaluations). Furthermore, we can observe PBIL is quite fast.

7.2 Results Using DE

In order to compare the results we have obtained from DE with other optimization methods and techniques, it is necessary to apply the experiments on the same predefined set of BS available locations.

We have programmed two types of experiments:

- In the first type, we try to find the optimal set of locations (reaching the 100% of coverage) for a fixed amount of BS transmitters (49). The parameters of DE to be initially set are: population size, crossover function and maximum number of generations to be performed.
- The second type of experiments consists in looking for the minimum number of transmitters to be located in the target area in order to reach a predetermined cover rate. A loop goes on starting from an initial amount of transmitters and increasing it until the wanted coverage is obtained. For every number of BS in this loop, DE is applied in the same manner as in the first type of experiments (see above).

Two crossover functions have been considered: FA and SA. Let be two set of locations (individuals), named A and B (the parents). Let be the S individual (the offspring) obtained from the application of the crossover function to A and B. FA function chooses the first half of A to build the first half of the offspring. The second half of the offspring is then built with the first half of B, but if a repeated location appears, successive locations of the second halves of B and A are taken. SA function chooses the second half of A to build the first half of the offspring, and the second half of the offspring is then built with the second half of B, but if a repeated location appears, successive locations of the first halves of B and A are taken.

With this background we have performed a series of tests for both types of experiments. For every combination of parameters, 30 independent runs have been performed for statistical purposes. We have observed than FA crossover produces better results than SA crossover, and furthermore, these results are obtained in a lower number of generations (conclusions obtained mainly from the second type of experiment). Table 2 shows the most important results for the first type of experiment, considering the same instance of the problem used by the other algorithms presented in this paper. The main conclusion is that the desired optimal coverage (100%) has not been reached. Perhaps this result could be improved with more evaluations. Also, you can observe DE algorithm is fast.

Table 2. The most important results with DE for solving the RND problem

	Config. best result
# Generations	4000
# Individuals	2000
Crossover	FA

	Best result	Average results
Fitness function	163.48	163.48
# Transmitters	49	49
Coverage	89.5%	89.5%
Execution time	3 minutes	3 minutes, 48 seconds
Execution on	Pentium IV –1.7 GHz	
# Evaluations	9,363	11,538

7.3 Results Using SA

SA has been used on the same instance as the previous algorithms, in order for the obtained results to be comparable.

SA has only three parameters the programmer needs to tune:

- The mutation probability. The values employed range from 0.005 to 0.97.
- The length of the Markov chain.
- The temperature decay, α. The values employed range from 0.99 to 0.99999.

The length of the Markov chain and the temperature decay have been proven to work in the same manner, thus to be equivalent. Therefore, we decided to keep the first at a constant value of 50, and allow the tuning of the latter.

For every combination of parameters, 50 independent runs have been performed in order to assure its statistical relevance. Table 3 shows the results. The tests have been performed in a 16 machine cluster named in dedicated mode, and the code has been developed using the MALLBA library [17]. This resource code is available at the web page http://neo.lcc.uma.es/mallba/easy-mallba/index.html.

SA has been able to solve the problem with 100% accuracy, but presents a high degree of variability between the best computational effort (441,141 evaluations) and the average (810,755 evaluations).

Table 3. The most important results with SA for solving the RND problem

	Config. best result
# Evaluations	5,000,000
Mutation probability	0.005
Markov chain length	50
Temperature decay	0.999
Initial temperature	1

	Best result	Average results
Fitness function	204.082	204.082
# Transmitters	49	49
Coverage	100%	100%
Execution time	7 minutes	12 minutes, 10 seconds
Execution on	Pentium IV – 2.4 GHz	
Normalized execution time (P.IV-1.7 GHz)	9 minutes, 53 seconds	17 minutes, 11 seconds
# Evaluations	441,141	810,755

7.4 Results Using CHC

When using the CHC algorithm on the same instance that the previous methods, we have considered two parameters that can be tuned:

- The population size. The values employed range from 50 to 10,000 individuals.
- The cataclysmic mutation probability. Ranging from 0.01 to 0.35.

50 independent runs are performed for every combination of parameters. Table 4 shows the best configuration, and the results obtained. The tests have been performed in a 16 machine cluster named in dedicated mode, and the code has been developed using the MALLBA library [17]. This resource code is available at the web page http://neo.lcc.uma.es/mallba/easy-mallba/index.html.

During the tests we have concluded that the mutation probability (second parameter tuned) has little effect on the algorithm's performance, and can be kept at a value of 35% without any significant loss of efficiency.

CHC solved the problem with 100% accuracy, and presents low variability in the computational effort: there is little difference between the best effort (291,200 evaluations) and the average (380,183 evaluations).

Table 4. The most important results with CHC for solving the RND problem

	Config. best result
# Evaluations	5,000,000
# Individuals	2,800
Mutation probability	0.35
Incest distance	25% vector length
Crossover probability	0.8
Elitism	YES

	Best result	Average results
Fitness function	204.082	204.082
# Transmitters	49	49
Coverage	100%	100%
Execution time	5 minutes, 58 seconds	7 minutes, 8 seconds
Execution on	Pentium IV – 2.4 GHz	
Normalized execution time (P.IV-1.7 GHz)	8 minutes, 25 seconds	10 minutes, 4 seconds
# Evaluations	291,200	380,183

8 Conclusions

In this paper we have solved the RND (Radio Network Design) problem with different metaheuristic techniques. Our aim was to solve the problem efficiently and at the same time research in the results of a wide spectrum of modern techniques. Most of these techniques (PBIL, SA and CHC) have obtained the optimal solution for this problem with square BTS of radius 20 (100% of coverage is attained placing 49 antennas, getting a fitness value of 204.082) except for the DE. However, DE is the evolutionary approach that needs lower times and number of evaluations in order to obtain a reasonable result. The difficulties in DE's accuracy may be originated by its intrinsic continuous nature.

If we look for the optimal solution, the best normalized execution times (around 9 minutes) and the best number of evaluations (below 300,000) is simultaneously attained by PBIL and CHC. Figure 2 shows the best result for each technique, both in normalized execution time (left) and in number of evaluations (right).

Figure 3 presents the same data but using the average results for each technique. In this case, the results are very similar to the previous ones: PBIL and CHC obtain a similar normalized execution time (around 10 minutes), but PBIL needs slightly less evaluations (306,855 vs. 380,183 -statistically similar). On the other hand, SA also reaches the optimal fitness value (204.082) but needs huge computational resources (time and evaluations) to obtain that result.

Future work includes the study of other bio-inspired algorithms, such as genetic algorithms, parallel genetic algorithms... in our quest for more efficient and accurate solvers for this problem. In this line, we will continue using grid computing with

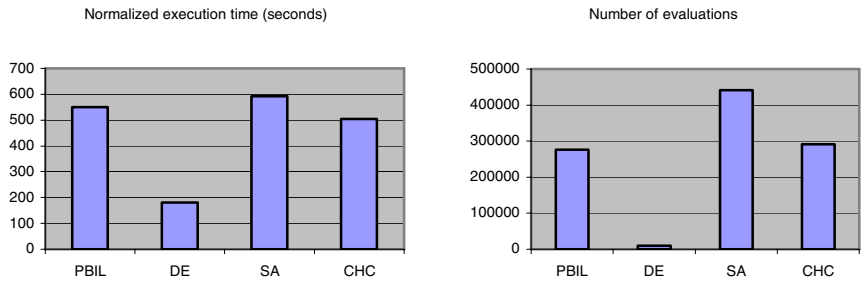

Fig. 2. Best result for each algorithm: (left) Normalized execution time (supposing a Pentium IV-1.7 GHz). (right) Number of evaluations.

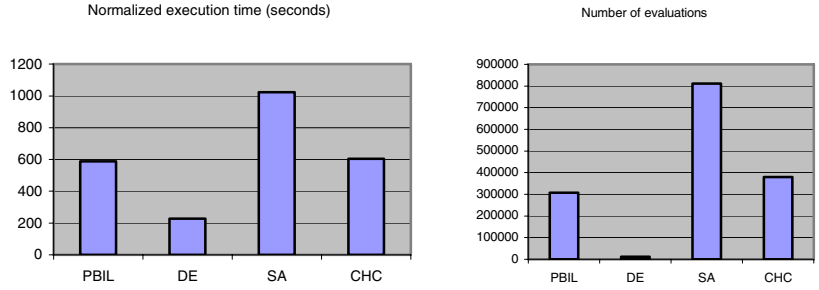

Fig. 3. Average results for each algorithm: (left) Normalized execution time (supposing a Pentium IV-1.7 GHz). (right) Number of evaluations.

BOINC and cluster computing in order to speedup all our experiments and to create more sophisticated algorithms by communicating information among component parallel agents.

Acknowledgments. This work has been partially funded by the Ministry of Education and Science and FEDER under contract TIN2005-08818-C04-01 and TIN2005-08818-C04-03 (the OPLINK project). Guillermo Molina is supported by grant AP2005-0914 from the Spanish government.

References

1. P. Calégari, F. Guidec, P. Kuonen, D. Kobler. Parallel Island-Based Genetic Algorithm for Radio Network Design. Journal of Parallel and Distributed Computing, 47(1): 86-90, November 1997.
2. P. Calégari, F. Guidec, P. Kuonen, F. Nielsen. Combinatorial Optimization Algorithms for Radio Network Planning. Theoretical Computer Science, 263(1): 235-265, July 2001.

3. E. Alba. Evolutionary Algorithms for Optimal Placement of Antennae in Radio Network Design. NIDISC'2004 Sixth International Workshop on Nature Inspired Distributed Computing, IEEE IPDPS, Santa Fe, USA, pp. 168-175, April 2004.
4. OPLINK: http://oplink.lcc.uma.es/problems/rnd.html, November 2006.
5. S. Baluja. Population-based Incremental Learning: A Method for Integrating Genetic Search based Function Optimization and Competitive Learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, June 1994.
6. S. Baluja, R. Caruana. Removing the Genetics from the Standard Genetic Algorithm. Twelfth International Conference on Machine Learning, San Mateo, CA, USA, pp. 38-46, May 1995.
7. K. Price, R. Storn. Differential Evolution – A Simple Evolution Strategy for Fast Optimization. Dr. Dobb's Journal, 22(4): 18–24, April 1997.
8. K. Price, R. Storn. Web site of DE. http://www.ICSI.Berkeley.edu/~storn/code.html, November 2006.
9. H.A. Abbass, R. Sarker. The Pareto Differential Evolution Algorithm. Int. Journal on Artificial Intelligence Tools, 11(4): 531-552, 2002.
10. S. Mendes, J.A. Gómez, M.A. Vega, J.M. Sánchez. The Optimal Number and Locations of Base Station Transmitters in a Radio Network. 3rd Int. Workshop on Mathematical Techniques and Problems in Telecommunications, Leiria, Portugal, pp.17-20, September 2006.
11. S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi. Optimization by Simulated Annealing. Science, 220(4598): 671-680, May 1983.
12. V. Cerny. A Thermodynamical Approach to the Travelling Salesman Problem: an Efficient Simulation Algorithm. Journal of Optimization Theory and Applications, 45: 41-51, 1985.
13. L.J. Eshelman. The CHC Adaptive Search Algorithm: How to Have Safe Search when Engaging in Nontraditional Genetic Recombination. Foundations of Genetic Algorithms, Morgan Kaufmann, pp. 265-283, 1991.
14. L.J. Eshelman, J.D. Schaffer. Preventing Premature Convergence in Genetic Algorithms by Preventing Incest. Fourth Int. Conf. on Genetic Algorithms, San Mateo, CA, USA, pp. 115-122, 1991.
15. BOINC: http://boinc.berkeley.edu, November 2006.
16. D.P. Anderson. BOINC: A System for Public-Resource Computing and Storage. 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, USA, pp. 365-372, November 2004.
17. E. Alba, F. Almeida, M. Blesa, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, C. León, G. Luque, J. Petit, C. Rodríguez, A. Rojas, F. Xhafa. Efficient Parallel LAN/WAN Algorithms for Optimization: The MALLBA Project. Parallel Computing, 32 (5-6): 415-440, June 2006.