Parallel Evolutionary Algorithms in Telecommunications: Two Case Studies

E. Alba, C. Cotta, F. Chicano, and A.J. Nebro

Dept. Lenguajes y Ciencias de la Computación, ETSI Informática,
University of Málaga, Campus de Teatinos, 29071 - Málaga - SPAIN
{eat, ccottap, antonio}@lcc.uma.es

Abstract. Sequential and parallel evolutionary algorithms (EAs) are developed and evaluated on two hard optimisation problems arising in the field of Telecommunications: designing error correcting codes, and finding optimal placements for antennas in radio networks. Different EA models (generational, steady-state and cellular) are compared on these two problems, both in sequential and parallel versions. We conclude that the cellular EA is a very effective technique, consistently finding the optimum, although it is slower than a steady-state EA. A distributed steady-state EA is shown to be the best approach, achieving the same success rate than the cellular EA in much lower time. Furthermore, it is shown that linear speedups are possible when using separate processors.

Keywords: Parallel Evolutionary Algorithms, Telecommunications, Error Correcting Codes, Radio Network Design, Efficiency

1 Introduction

An important symbol of our present information society are Telecommunications. With a rapidly growing number of user services, Telecommunications is a field in which many open research lines are challenging the research community. Many of the problems found in this area can be formulated as optimisation tasks. Some examples are assigning frequencies in radio link communications [9, 12], predicting bandwidth demands in ATM networks [17], developing error correcting codes for transmission of messages [7, 10], and designing the telecommunication network [8, 13, 14, 19].

In practice, most of these optimisation tasks are unaffordable with exact techniques. Hence, the use of heuristic approaches is in order. In this sense, evolutionary algorithms have constituted a popular choice. However, the high complexity of real-world intances of these tasks requires the computational power of several machines working together to find acceptable solutions. This gives rise to the application of parallel algorithms for efficiently tackling these problems.

In this work, our goal is to analyse different parallel evolutionary algorithms (PEAs) with application to problems in the domain of Telecommunications. We will be interested in some numerical issues, such as the computational effort to locate a solution, or the advantages of using multiple pools of solutions rather than a single centralised population. To this end, we will consider a test suite composed of two different optimisation tasks from this domain: designing error correcting codes (ECC), and finding optimal placements for antennas in a radio network (RND).

The remainder of the paper is organised as follows. First, a general inspection at the algorithms we have considered is made from a unified point of view in Section 2. Next, the test suite is described, and application details are outlined in Section 3. Subsequently, empirical results are presented and analysed both for sequential and parallel versions of the algorithms (Section 4). Finally, some concluding remarks and future research lines are drawn in Section 5.

2 Parallel Evolutionary Algorithms: a Unified View

In this section we intend to provide a quick overview of PEAs in order to classify and explain the class of algorithms we are using in the paper. For this purpose, it is interesting to focus on the way the population of solutions is handled in the algorithm.

It is usual for many EA families to manipulate the population as a single pool of individuals, i.e., there is just one global competition for reproduction and surviving, and mating is possible between any two individuals in the population. This scenario is usually termed *panmixia*. On the contrary, most parallel EAs found in the literature utilise some kind of spatial disposition for the individuals, and then parallelise the resulting chunks in a pool of processors. We must stress at this point of the discussion that parallelisation is achieved by first structuring the panmictic algorithm, and then parallelising it. For this reason, we distinguish between structuring a population and making a parallel implementation, since the same structured EA can admit many different implementations.

There exists a long tradition in using EAs with structured populations, especially in the works associated to parallel implementations. Among the most widely known types of structured EAs, *distributed* [18] (dEA) and *cellular* [16] (cEA) algorithms are two very popular optimisation procedures. Decentralising a single population can be achieved by partitioning it into several sub-populations performing sparse exchanges –migration– of individuals (distributed EAs), or in the form of overlapped neighbourhoods within a global topology (cellular EAs).

In distributed EAs, additional parameters controlling when migration occurs and how migrants are selected/incorporated from/to the source/target islands are needed [2, 5, 18]. This model (either running on separate processors or not) is usually faster than a panmictic EA due to the separate search in several regions of the solution space. As to cellular EAs, the existence of overlapped small neighbourhoods (e.g., a 2D grid [15]) helps in exploring the search space, due to the fact that good solutions are smoothly propagated through the underlying topology [4]. These two kinds of PEA seem to provide a better sampling of the search space and improve the numerical and runtime behaviour of the basic algorithm in many cases [3, 11].

In Figure 1 we plot a 3D representation of structured algorithms based on the number of sub-algorithms, the number of individuals in each one, and the coupling among them. While a distributed EA has a large sub-population ($\gg 1$), a cEA has typically one single individual in every sub-algorithm. The sub-algorithms are loosely connected in dEAs, while they are tightly coupled in cEAs. Additionally, there exist only a few sub-algorithms in dEAs, while a large number of them exist in cEAs. We use this cube to provide a generalised way for classifying structured EAs.

In this work we will use three basic search algorithms to solve the problems under consideration: steady-state (ssEA), generational (genEA) and cellular (cEA) evolutionary algorithms.

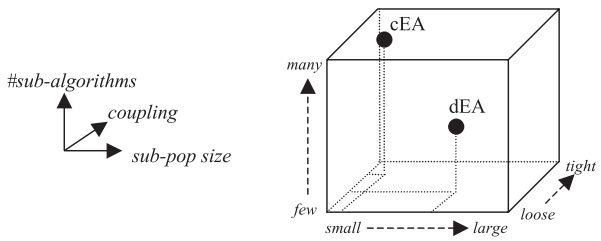

Fig. 1. The structured-population evolutionary algorithm cube.

The first two ones are sub-classes of panmictic algorithms, while the latter one is a sub-class of structured EA. Additionally, we have developed a distributed version (dEA) for any of these three models that can be run in parallel on any number of processors in a LAN of workstations. We note the resulting algorithms as dxxEA, where xx $\in$ {gen, ss, c}.

3 Description of the Problems

After having described the algorithms used in our experiments, we now turn to the test suite from the domain of Telecommunications we have chosen. This test suite comprises two problems (ECC and RND) that are both hard to solve, and exhibit a clear interest from a practical point of view.

3.1 Designing Error Correcting Codes (ECC)

As a basic part of any communication system, the designer must solve the problem of finding a suitable code for transmitting messages through a noisy channel as reliably and quickly as possible. As we often find in optimisation, we must fulfill two conflicting requirements to solve this problem. A first goal is to find minimum length codewords, so that we can ensure quickly transmission of the messages encoded using these codewords. On the other hand, the Hamming distance between codewords must be maximised in order to guarantee a high level of error correction in the receiver part. The underlying idea is the following: if all codewords are separated by d bits, then any modification of at most $(d-1)/2$ bits in a valid codeword can be easily reverted by taking the most similar codeword. Take a look to Figure 2 (left) to understand our basic problem statement.

There are many kinds of error correcting codes; our work will focus on binary linear block codes. Such a correcting code can be represented as a three-parameter vector (n, M, d), where n is the number of bits in each word in a code, M is the number of words in the code, and d is the minimum Hamming distance between any pair of words $C(i)$ and $C(j)$ $(i \neq j)$ in the code.

An optimal code is one that maximises d, given n and M. In this paper we solve the problem for a $M = 24$-word code, with $n = 12$-bit length codewords as in [7]. As an indication, the search space is $\binom{2^n}{M}$, which in our case means approximately 10^{87}. Exhaustive search is clearly out of question.

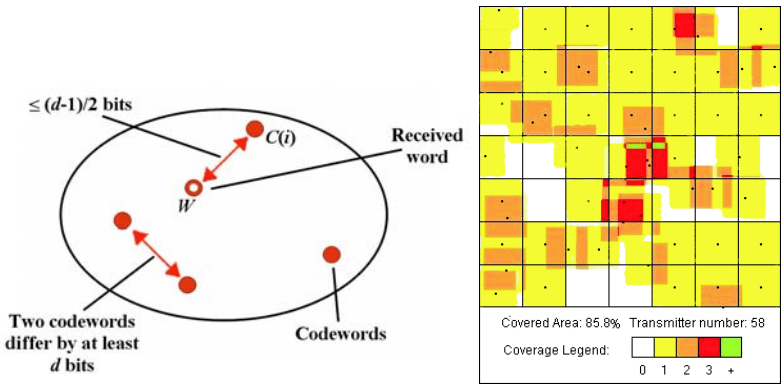

Fig. 2. (Left) Graphic interpretation of a codeword system. (Right) Graphical representation a solution for the RND problem.

A first representation of the problem would assign an $M \times n$ bit string to individuals, i.e., the genotype would be the concatenation of the M codewords of length n. This representation directly maps the problem parameters into the chromosome of each individual, providing an easy and efficient implementation of the algorithm. We have added an additional constraint though: $M/2$ codewords (we assume M to be even) must be obtained by performing a full bitwise-inversion of the remaining $M/2$ ones. By doing so, the genome length is halved with the subsequent advantage of faster convergence. Furthermore, some preliminary tests have shown that no degradation in the quality of the results takes place when adding this constraint. Actually, a plain representation is clearly outperformed by the proposed one.

As to the fitness function, we have followed the proposal of Dontas and de Jong [10], i.e.,

$$Eval(\mathbf{x}) = f(\mathbf{y}) = \frac{1}{2\sum_{i=1}^{M-1}\sum_{j=i+1}^{M} d_{ij}^{-2}} \ , \tag{1}$$

where $\mathbf{y}$ is the concatenation of $\mathbf{x}$ and $\bar{\mathbf{x}}$, and d_{ij} is the Hamming distance between y_i and y_j. This function measures how well the M words are placed in the corners of an n-dimensional space by considering the minimal energy configuration of M particles, as it is done in Physics.

3.2 The Radio Network Design Problem (RND)

The radio coverage problem amounts to covering an area with a set of transmitters. The part of an area that is covered by a transmitter is called a cell. In the following we will assume that the cells and the area considered are discretised, that is, they can be described as a finite collection of geographical locations (taken from a geo-referenced grid, for example). The computation of cells may be based on sophisticated wave propagation models, on measurements, or on draft estimations. In any case, we assume that cells can be computed and returned by an *ad hoc* function.

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 $(M \cup L, E)$, where E is a set of edges such that each transmitter location is linked to the locations it covers. Let vector $\mathbf{x}$ be a solution to the problem where $x_i \in \{0, 1\}$, and $i \in M$ indicates whether a transmitter is being used or not. As the geographical area needs to be discretised, the potentially covered locations are taken from a grid, as shown in Figure 2 (right).

Searching for the minimum subset of transmitters that covers a maximum surface of an area comes to searching for a subset $M' \subseteq M$ such that $|M'|$ is minimum and $|N(M', E)|$ is maximum, where $N(M', E) = \{u \in L \mid \exists v \in M', (u, v) \in E\}$, and $M' = \{t \in M \mid x_t = 1\}$. Representing such subsets in the EA is straightforward: every potential transmitter location is assigned a bit in a binary string; an 1 in a certain string position means that the associated transmitter will be used in the placement. A total number of 149 transmitter sites is considered (49 are uniformly distributed in a 7×7 grid, and 100 are located at random positions).

We have used the fitness function proposed in [6], i.e.,

$$Eval(\mathbf{x}) = \frac{Cover(\mathbf{x})^{\alpha}}{Ones(\mathbf{x})} \ , \quad \text{where} \quad Cover(\mathbf{x}) = 100\frac{N(M', E)}{N(M, E)} \tag{2}$$

and $Ones(\mathbf{x})$ is the number of active bits in $\mathbf{x}$, i.e., the number of selected transmitters. Thus, this fitness function is intended to both maximising coverage and minimising the cost, being the parameter α responsible for setting the appropriate tradeoff between these two conflicting objectives.

4 Empirical Results

In this section we present the results of performing an assorted set of tests by using sequential and parallel EAs to solve both the ECC and the RND problems.

First of all, let us consider the ECC problem. Initially, we have performed 30 runs of two panmictic EAs, an elitist generational $(\mu, \mu-1)$-EA and a steady-state $(\mu, 1)$-EA, and a cellular EA (using a square 2D-grid with NEWS –North, East, South, West– neighbourhood). The parameterisation is the same for all of them in order to get meaningful results: a population of 64 individuals (a rather small and resource-saving setting), roulette wheel selection of each parent, uniform crossover with application probability 1.0 and a 60%-bias to the fittest parent, and traditional bit-flip mutation with probability $p_m = (M \times n)^{-1}$. The replacement step eliminates always the worst string in the case of steady-state EA, generates a new population preserving the best individual from generation to generation in the genEA, and replaces the considered central individual with the newly generated one only if it is better in the cellular EA. In all cases, the runs are stopped after finding an optimal solution (minimum Hamming distance $d = 6$) or reaching a maximum number of evaluations $(5 \cdot 10^5)$.

In Table 1, we show the results for the three models. From a numerical point of view, the genEA is clearly the worst algorithm in the number of successful runs. The cEA is the slowest one, requiring a considerably number of evaluations due to its more exhaustive sampling behaviour (a salient feature of cEA algorithms). Nevertheless, it is the more successful algorithm, finding the optimum solution in 100% of the runs. The ssEA offers an intermediate performance: it is the faster one, but only achieves a moderate success rate (40%). Except for the cEA-time vs. genEA-time comparison, all results are statistically significant (see the middle and rightmost sections of Table 1).

Table 1. Results for non-distributed models in the ECC problem. (Leftmost) Percentage of successful runs, mean number of evaluations, and mean time in successful runs. (Middle and righmost) Statistical significance of the results (p-values are shown).

	Numerical Results			*Significance (#evals)*			*Significance (time)*		
	%Success	*#evals*	*time (sec.)*	*ssEA*	*genEA*	*cEA*	*ssEA*	*genEA*	*cEA*
ssEA	40%	15476	85.37	-	0.0008	8.69e-8	-	1.92e-5	1.94e-5
genEA	10%	32756	217.47	0.0008	-	0.0001	1.92e-5	-	0.115
cEA	100%	52757	280.00	8.69e-8	0.0001	-	1.94e-5	0.115	-

Next, distributed versions of the above algorithms are considered. In this case, we have 8 islands, each one with 16 individuals performing in parallel the corresponding basic evolutionary step, with an added migration operation. The migration will occur in a unidirectional ring topology, sending one randomly chosen individual to the neighbouring sub-population. The target population incorporates this individual only if it is better than its presently-worst solution. The migration step is asynchronously performed every 512 evaluations in every island. Each machine has an Ultra Sparc 1 CPU at 143 Mhz with 64Mb RAM. The results are shown in Table 2.

Notice that the distributed algorithms sample a larger number of points for finding the optimum as a consequence of the parallel search. The advantage is again for dssEA and dgenEA, with a smaller numeric effort than dcEA. With respect to run times, dssEA is the faster one, closely followed by dgenEA; the dcEA is clearly the slower algorithm (all these conclusions are statistically assessed in the middle and rightmost sections of Table 2). As it can be also seen, the distributed version of ssEA has notably improved its efficacy (100%) with respect

Table 2. Results for distributed models in the ECC problem. (Leftmost) Percentage of successful runs, mean number of evaluations, and mean time in successful runs. (Middle and rightmost) Statistical significance of the results (p-values are shown).

	Numerical Results			*Significance (#evals)*			*Significance (time)*		
	%Success	*#evals*	*time (sec.)*	*ssEA*	*genEA*	*cEA*	*ssEA*	*genEA*	*cEA*
$dssEA_8$	100%	36803	22.3	-	0.280	5.05e-6	-	0.0529	3.49e-7
$dgenEA_8$	9%	42317	28.9	0.280	-	2.01e-5	0.0529	-	8.09e-6
$dcEA_8$	100%	89598	62.0	5.05e-6	2.01e-5	-	3.49e-7	8.09e-6	-

to its serial version (40%), thus supporting the idea that dssEA is the more suited algorithm for this problem among all the tested algorithms. In fact, our dssEA_8 (36803 evaluations) outperforms other existing algorithms such as parallel genetic simulated annealing [7] which needs more than 256000 evaluations of the fitness function. Furthermore, and despite the fact that, strictly speaking, run-times of the sequential and parallel versions cannot be directly compared because they are completely different search algorithms [1], we can informally state that changing from sequential to parallel algorithms has speeded up the search in a factor of 3.8/7.5/4.5 for dssEA/dgenEA/dcEA respectively. This is a good indication of the potential benefits of parallelising EAs.

We now turn to the RND problem. Taking into account the results obtained on the ECC problem, we have focused on steady-state models, since they seem to provide the best tradeoff between efficacy and efficiency. We have considered a sequential model with 512 individuals in the population, roulette wheel selection, double-point crossover with probability 1.0, bit-flip mutation with probability $1/n_s$ (n_s is the number of transmitter sites), and replacement of the worst string. As to the distributed model, we have 8 islands (each one with 64 individuals performing in parallel the basic evolutionary step mentioned above), with the same migration scheme as for the previous problem (except that migration takes place every 2048 evaluations). Again, results correspond to series of 30 runs.

The results are shown in Table 3. In this case, the distributed version has been tested both on physically separate CPUs (dssEA_8), and on a single CPU (dssEA_1). In all cases (including the sequential model), a 100%-success rate is achieved, i.e., the optimum is always found.

Table 3. Results for the RND problem. (Leftmost) Percentage of successful runs, mean number of evaluations, and mean time in successful runs. (Middle and righmost) Statistical significance of the results (p-values are shown).

	Numerical Results		*Significance (#evals)*			*Significance (time)*		
	#evals	*time (hours)*	*ssEA*	$dssEA_1$	$dssEA_8$	*ssEA*	$dssEA_1$	$dssEA_8$
ssEA	173013	3.86	-	2.86e-11	6.02e-14	-	2.32e-11	4.55e-9
$dssEA_1$	491496	10.87	2.86e-11	-	0.7380	2.32e-11	-	3.48e-14
$dssEA_8$	505624	1.38	6.02e-14	0.7380	-	4.55e-9	3.48e-14	-

Several interesting facts can be inferred from Table 3. Firstly, it is clear that the sequential ssEA is the best algorithm in terms of the number of evaluations, since it samples roughly a third of the points sampled by dssEA (in either of the two versions). However, dssEA_8 is better in an average wall-clock sense, since it performs very quickly (1.38 hours) when compared to the ssEA (3.86 hours), or to the dssEA_1 (10.87 hours). This is an expected result: the distributed search samples more points in the search space, but it can greatly benefit from physical parallelisation in order to perform much faster computations. Notice also that comparing computational times for dssEA_8 and dssEA_1 results in a almost perfectly linear speedup (7.87), a very satisfactory result. In all cases we obtain the optimum fitness value for this problem (100% efficacy), which

is a clear improvement on the existing results [6], where the best found solution has a fitness of 193.8 (95% of optimum).

We provide a final experiment in order to obtain further evidence regarding the attainable speedup when using physically separate processors. The termination criterion has been here relaxed with respect to the previous tests. More precisely, a run is stopped when a solution whose quality is at least 61% of the optimum is found, as it is done in [6]. The results are shown in Table 4.

Table 4. Results for *RND* 61% *dssEA*.

	Number of Evaluations	*Time (min.)*	*Fitness*
$dssEA_1$	3940	6.78	127.19
$dssEA_8$	4021	0.85	126.79
$IPGA_1$[6]	40000	?	125.3
$IPGA_{10}$[6]	∼4000	?	∼126

In this case, the speedup between the dssEA_1 and dssEA_8 is 7.97, even better than when 100%-success was sought. This result confirms that the linear speedup is not a result of a rather strict termination criterion (finding the optimum), but accurately reflects the behaviour of dssEA_1 vs. dssEA_8 for this problem.

5 Conclusions

In this paper we have tackled the solution of two important problems in the domain of Telecommunications (designing error correcting codes, and finding optimal placements for antennas in radio networks) via parallel evolutionary algorithms. We selected steady-state and generational EAs as members of the panmictic (single population) class of algorithms, and cellular EAs as distinguished members of the structured (decentralised) class of EAs. Since we are interested not only in numerical results, but also in the search time, we made distributed implementations of these three algorithms.

Regarding the ECC problem, our conclusions are encouraging. The distributed ssEA is the overall best performing algorithm, both in time and number of evaluations. The genEA and its distributed version provided slightly worse results in time and evaluations, and exhibited a clear drawback: the percentage of runs finding the optimum is around 10%, which is unacceptable from our point of view. Finally, the cEA and its distributed version has revealed itself as quite an effective (100% success) technique, especially in its sequential version where it has shown a competitive behaviour. As to the wall-clock time, cEA is somewhat slow, since it is generation-driven, as opposed to the individual-driven basic step of ssEA; however, its features of broad exploration are salient, while its disadvantages of larger run times can be solved by using a parallel version (that performs better for larger sub-populations).

As to the RND problem, steady-state models provide very good sampling of the search space, although a sequential version of this algorithm has been shown to be somewhat slow. This drawback has been overcome by using a distributed multi-population version running on a network of workstations that –despite making a larger sampling of the search space– provides a much faster functioning, reducing the waiting time for a solution to just about one hour (with 8 processors). Furthermore, a nearly linear speedup has been obtained when moving from a single-processor implementation to an 8-processor implementation. This speedup has been confirmed in two different scenarios, thus indicating a clear trend of behaviour.

From a global point of view, the results have been very satisfactory for the two problems, thus emphasizing the fact that PEAs can be a very adequate tool for dealing with the hard optimisation tasks that arise in the domain of Telecommunications. Our future work will address other instances of both ECC and RND, as well as other telecommunication problems with interest in real world applications. A further understanding on the numerical behaviour of the algorithms is already in progress, in order to export our conclusions to similar problems. Finally, we are also searching combinations of EAs and other techniques (i.e., hybrid algorithms) that could outperform our current algorithms.

Acknowledgements

The authors want to acknowledge the funds from the Spanish national CICYT project TIC1999-0754-C03-03 that partially supported this work.

References

1. E. Alba. Parallel evolutionary algorithms can achieve superlinear performance. *Information Processing Letters*, 82(1):7–13, April 2002.
2. E. Alba and J. M. Troya. Influence of the migration policy in parallel distributed GAs with structured and panmictic populations. *Applied Intelligence*, 12(3):163–181, 2000.
3. E. Alba and J. M. Troya. Analyzing synchronous and asynchronous parallel distributed genetic algorithms. *Future Generation Computer Systems*, 17:451–465, January 2001.
4. S. Baluja. Structure and performance of fine-grain parallelism in genetic search. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 155–162. Morgan Kaufmann, 1993.
5. T. C. Belding. The distributed genetic algorithm revisited. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 114–121. Morgan Kaufmann, 1995.
6. P. Calégari, F. Guidec, P. Kuonen, and D. Kobler. Parallel island-based genetic algorithm for radio network design. *Journal of Parallel and Distributed Computing*, 47:86–90, 1997.
7. H. Chen, N.S. Flann, and D.W. Watson. Parallel genetic simulated annealing: A massively parallel SIMD algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 9(2):126–136, 1998.
8. C.H. Chu, G. Premkumar, and H. Chou. Digital data networks design using genetic algorithms. *European Journal of Operational Research*, 127:140–158, 2000.
9. C. Cotta and J.M. Troya. A comparison of several evolutionary heuristics for the frequency assignment problem. In J. Mira and A. Prieto, editors, *Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence*, volume 2084 of *Lecture Notes in Computer Science*, pages 709–716. Springer-Verlag, Berlin Heidelberg, 2001.
10. K. Dontas and K. De Jong. Discovery of maximal distance codes using genetic algorithms. In *Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence*, pages 905–811, Herndon, VA, 1990. IEEE Computer Society Press, Los Alamitos, CA.
11. V. S. Gordon and D. Whitley. Serial and parallel genetic algorithms as function optimizers. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 177–183. Morgan Kaufmann, 1993.
12. A. Kapsalis, V.J. Rayward-Smith, and G.D. Smith. Using genetic algorithms to solve the radio link frequency assigment problem. In D.W. Pearson, N.C. Steele, and R.F. Albretch, editors, *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, pages 37–40. Springer-Verlag, 1995.
13. N. Karunanithi and T. Carpenter. Sonet ring sizing with genetic algorithms. *Computers and Operations Research*, 24(6):581–591, 1997.
14. S. Khuri and T. Chiu. Heuristic algorithms for the terminal assignment problem. In *Proceedings of the 1997 ACM Symposium on Applied Computing*, pages 247–251. ACM Press, 1997.
15. J. Sarma and K. A. De Jong. An analysis of the effect of the neighborhood size and shape on local selection algorithms. In H. M. Voigt, W. Ebeling, I. Rechenberg, and H. P. Schwefel, editors, *Parallel Problem Solving from Nature (PPSN IV)*, volume 1141 of *Lecture Notes in Computer Science*, pages 236–244. Springer-Verlag, Heidelberg, 1996.
16. P. Spiessens and B. Manderick. A massively parallel genetic algorithm. In L. B. Booker R. K. Belew, editor, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 279–286. Morgan Kaufmann, 1991.
17. N. Swaminathan, J. Srinivasan, and S.V. Raghavan. Bandwidth-demand prediction in virtual path in atm networks using genetic algorithms. *Computer Commnunications*, 22(12):1127–1135, 1999.

18. R. Tanese. Distributed genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 434–439. Morgan Kaufmann, 1989.
19. C. Vijayanand, M. S. Kumar, K. R. Venugopal, and P. S. Kumar. Converter placement in all-optical networks using genetic algorithms. *Computer Communications*, 23:1223–1234, 2000.