A Parallel Implementation of an Artificial Immune System to Handle Constraints in Genetic Algorithms: Preliminary Results

Carlos A. Coello Coello and Nareli Cruz Cortés
CINVESTAV-IPN
Departamento de Ingeniería Eléctrica
Sección de Computación
Av. Instituto Politécnico Nacional No. 2508
Col. San Pedro Zacatenco
México, D. F. 07300
ccoello@cs.cinvestav.mx, nareli@computacion.cs.cinvestav.mx

Abstract- In this paper, we present a parallel version of a constraint-handling technique based on the artificial immune system. The proposed approach does not require penalty factors of any kind, it is relatively simple to implement and it is quite competitive with more sophisticated techniques. Additionally, when parallelized using an island scheme, the approach not only reduces its computational time, but it also improves the quality of the results produced.

1 Introduction

Despite the success of genetic algorithms (GAs) as optimization techniques [6], they are really an unconstrained search technique. Therefore, an additional mechanism is required to incorporate constraints of any type (linear, nonlinear, equality, inequality) into the fitness function.

In recent years, a number of constraint-handling techniques have been proposed, some of which are very elaborate [10, 2]. From those techniques, penalty functions remain as the most popular choice for practitioners [12]. However, the performance of a penalty function depends on the type of function adopted and on the values defined for its parameters (namely, the penalty factor). In this paper, we propose and validate an alternative approach to incorporate constraints into the fitness function of a GA. The proposed approach incorporates an emulation of the immune system and uses genotypic-based distances to move from the infeasible to the feasible region of a problem. The algorithm was originally proposed in [3] and its parallelization is presented in this paper. The parallel version is compared to its sequential counterpart to evaluate the performance gains obtained and the differences in the results produced (because of the type of paradigm adopted to parallelize our algorithm, it was expected that the results of both versions would differ). Our preliminary results indicate that the algorithm remains as a highly competitive approach (in terms of quality of the results produced) with respect to contemporary constraint-handling techniques when parallelized using an island-based scheme. Additionally, the parallelization of the algorithm seems to provide remarkable increases in performance with respect to the sequential version (in some cases, much higher than expected).

2 Problem Statement

The problem that is of interest to us is the general nonlinear programming problem in which we want to:

$$\text{Find } \vec{x} \text{ which optimizes } f(\vec{x}) \tag{1}$$

subject to:

$$g_i(\vec{x}) \leq 0, \quad i = 1, \ldots, n \tag{2}$$

$$h_j(\vec{x}) = 0, \quad j = 1, \ldots, p \tag{3}$$

where $\vec{x}$ is the vector of solutions $\vec{x} = [x_1, x_2, \ldots, x_r]^T$, n is the number of inequality constraints and p is the number of equality constraints (in both cases, constraints could be linear or non-linear).

3 The Proposed Approach

The proposed algorithm emulates the invaders recognition process by combining antibodies' libraries in order to attain antigen specificity.[1] Furthermore, the purpose is to learn to identify the proper antibodies. The search process of our approach is led by a genetic algorithm. Thus, what we propose is a scheme in which a genetic algorithm is embedded into another genetic algorithm. Note however that the computational complexity of the approach is not really (N^2), because the internal genetic algorithm does not evaluate the original fitness function of the problem as we will see later on (the number of fitness function evaluations remains the same as in a simple GA — i.e., Gmax ×M, where Gmax is the maximum number of generations and M is the population size). Our algorithm is the following:

1. Determine the number of processors available and make it equal to the number of subpopulations (called demes) created.

[1] Note that our approach is based on the negative selection algorithm [5], but there are several other models of the artificial immune system. See for example [4].

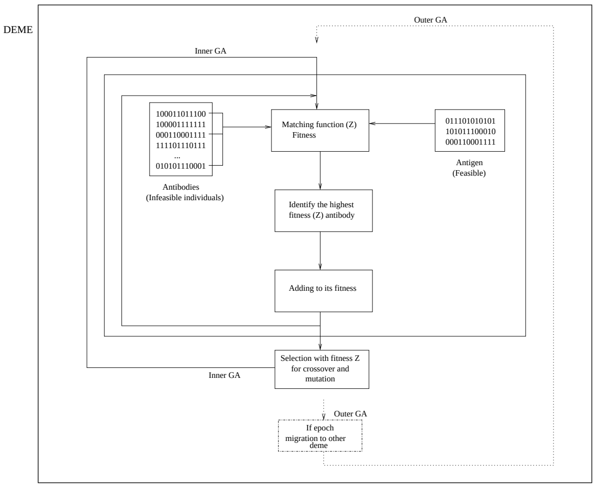

Figure 1: Our algorithm of the immune system.

2. Determine the size of each deme (d) dividing the population size of the original (i.e., sequential) GA by the total number of demes.

3. For each deme do the following:

   (a) Generate randomly an initial population for the GA.

   (b) If the initial population contains a mixture of feasible and infeasible individuals, then the population is divided in two groups. The first contains the infeasible individuals, which are denominated "antibodies". The second contains the feasible individuals, which are called "antigens".

   (c) If none of the individuals in the initial population is feasible, then we use the magnitude of constraint violation of each individual as its fitness. Then, we use as the "antigen" to the best individual in the population, where "best" means the one with the lowest constraint violation. The "antibodies" are a randomly selected portion of the population. For the assignment of antibodies, it is important to consider only a portion of the population, to avoid premature convergence. Under this scheme (i.e., no feasible individuals in the initial population), we run our simulation of the immune system based on minimum constraint violation and we verify at certain intervals if we have reached the feasible region. Whenever that happens, we then use the feasible individuals found as our antigens, and we proceed to divide the population as in the previous step.

   (d) The fitness of all the antibodies is set to zero.

   (e) The fitness of the antibodies is computed according to their similarity with a set of predetermined antigens in the following way:

      - An antigen is randomly selected from the antigens population.
      - From the population of N antibodies, we randomly select a sample of size σ without replacement.
      - Each antibody in the sample is compared against the antigen selected, and we compute the result of the comparison, to which we will call Z (matching magnitude). Z represents a distance (normally but not necessarily Euclidean) measured at the genotype level.
      - We identify the antibody in the population that has the highest matching magnitud (Z). Ties are solved by randomly choosing a winner.
      - The matching magnitude of the winner antibody is added to its fitness. The fitness of the other antibodies remains unaltered.
      - The antibodies are returned to the population, and the process is repeated typically during three times the number of antibodies.

   (f) Based on the fitness computed in the previous step, the population of antibodies is reproduced in a traditional genetic algorithm (using crossover and mutation).

   (g) The process is repeated C times, where C refers to the result of dividing the number of predefined iterations for the sequential immune system by the total number of demes.

   (h) Individuals are returned to the external GA and we proceed in the conventional way.

4. If the epoch was completed, then:

   (a) Migrate the union of the best individual of the deme plus r randomly selected individuals to a different deme at each iteration.

   (b) Receive $r+1$ migrating individuals that are to replace $r+1$ randomly selected individuals within the deme.

5. Repeat from (b) until reaching convergence.

This algorithm is illustrated in Figure 1. The amount of similarity Z of an antibody against an antigen is computed using:

$$Z = \sum_{i=1}^{L} t_i \qquad (4)$$

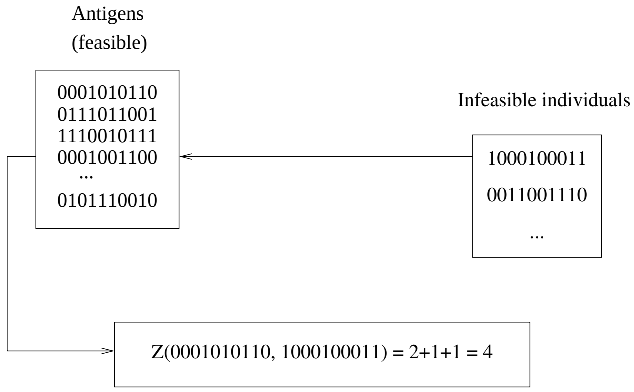

Figure 2: Matching measure Z.

where $t_i = 1$ if there is a matching at position $i = 1, \cdots, L$ (L is the length of the chromosome), or zero if there is not match. This is illustrated in Figure 2.

There are a few issues that need to be mentioned. First, the approach is really using a GA embedded inside another GA which optimizes a certain function. However, the GA that is run with the emulation of the immune system does not use the fitness function directly and, therefore, does not require to evaluate the objective function of the problem. The size of the sample σ of antibodies determines if the antibodies are *generalist* (i.e., able to distinguish a great variety of antigens) or *specialist* (i.e., able to distinguish only a small variety of antigens). If the value of σ (i.e., size of the sample of antibodies) is sufficiently large, then the antibodies tend to be specialists, since there is a larger probability that an antibody can exactly match a certain antigen. Conversely, if the value of σ is small, then the antigens tend to be generalist.

The parallelization of the algorithm took place applying a coarse-grained GA with a distributed memory using MPI (Message Passing Interface). The population is divided in as many demes (subpopulations) as processors are available. Each demes evolves independently from the others exchanging individuals at regular intervals. Since the demes have a lower number of individuals that the population size of the sequential GA, then the number of iterations required for the internal cycle (see Figure 1) of the immune system is proportionally reduced according to its size. This means that the number of iterations that each deme executes is obtained dividing the population size of the sequential version of the algorithm by the total number of demes.

Our algorithm is an extension of the proposal of Hajela & Lee [7]. One of the main limitations of Hajela & Lee's approach is their assumption that the initial population of the GA always contained a mixture of feasible and infeasible individuals. We do not require that in our approach. When the initial population does not have any feasible individuals (a common situation in real-world problems), our approach uses constraint violation information to guide the search (i.e., individuals that violate less restrictions are considered as the best in the population).

4 Examples

The test functions used to validate our approach are the following:

1. **Example 1**:

   Minimize:

$$f(\vec{x}) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i \quad (5)$$

   subject to:

$$\begin{aligned}
g_1(\vec{x}) &= 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0 \\
g_2(\vec{x}) &= 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \leq 0 \\
g_3(\vec{x}) &= 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0 \\
g_4(\vec{x}) &= -8x_1 + x_{10} \leq 0 \\
g_5(\vec{x}) &= -8x_2 + x_{11} \leq 0 \\
g_6(\vec{x}) &= -8x_3 + x_{12} \leq 0 \\
g_7(\vec{x}) &= -2x_4 - x_5 + x_{10} \leq 0 \\
g_8(\vec{x}) &= -2x_6 - x_7 + x_{11} \leq 0 \\
g_9(\vec{x}) &= -2x_8 - x_9 + x_{12} \leq 0
\end{aligned}$$

   where $0 \leq x_i \leq 1$ ($i = 1, \ldots, 9$), $0 \leq x_i \leq 100$ ($i = 10, 11, 12$) and $0 \leq x_{13} \leq 1$. The global minimum is at $\vec{x}^* = (1,1,1,1,1,1,1,1,1,3,3,3,1)$ where six constraints are active (g_1, g_2, g_3, g_7, g_8 and g_9) and $f(\vec{x}^*)$=-15.

2. **Example 2**:

   Minimize:

$$\begin{aligned}
f(\vec{x}) = 5.3578547x_3^2 + 0.8356891x_1x_5 + \\
37.293239x_1 - 40792.141
\end{aligned}$$

   subject to:

$$\begin{aligned}
g_1(\vec{x}) = 85.334407 + 0.0056858x_2x_5 + \\
0.0006262x_1x_4 - 0.0022053x_3x_5 - 92 \leq 0 \\
g_2(\vec{x}) = -85.334407 - 0.0056858x_2x_5 - \\
0.0006262x_1x_4 + 0.0022053x_3x_5 \leq 0 \\
g_3(\vec{x}) = 80.51249 + 0.0071317x_2x_5 + \\
0.0029955x_1x_2 + 0.0021813x_3^2 - 110 \leq 0 \\
g_4(\vec{x}) = -80.51249 - 0.0071317x_2x_5 - \\
0.0029955x_1x_2 - 0.0021813x_3^2 + 90 \leq 0 \\
g_5(\vec{x}) = 9.300961 + 0.0047026x_3x_5 + \\
0.0012547x_1x_3 + 0.0019085x_3x_4 - 25 \leq 0 \\
g_6(\vec{x}) = -9.300961 - 0.0047026x_3x_5 - \\
0.0012547x_1x_3 - 0.0019085x_3x_4 + 20 \leq 0
\end{aligned}$$

where: $78 \le x_1 \le 102$, $33 \le x_2 \le 45$, $27 \le x_i \le 45$ ($i = 3, 4, 5$). The global optimum of this problem is located at $\vec{x}^* = (78, 33, 29.995256, 45, 36.775812)$ where $f(\vec{x}^*) = -30665.539$. Two constraints are active at the optimum: g_1 and g_6.

3. **Example 3**:

   Minimize:

$$f(\vec{x}) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7$$

subject to:

$$g_1(\vec{x}) = -127 - 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 \le 0$$
$$g_2(\vec{x}) = -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 \le 0$$
$$g_3(\vec{x}) = -196 + 23x_1 + x_2^2 + 6x_6^2 - 8x_7 \le 0$$
$$g_4(\vec{x}) = 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \le 0$$

where $-10 \le x_i \le 10$ for ($i = 1, \ldots, 7$). The optimum solution is $\vec{x}^* = $ (2.330499, 1.951372, -0.4775414, 4.365726, -0.6244870, 1.038131, 1.594227) where $f(\vec{x}^*)$= 680.6300573. Two constraints are active (g_1 and g_4).

5 Experimental Setup

The efficiency of a parallel algorithm tends to be measured in terms of its correctness and its speedup. The speedup (SP) of an algorithm is obtained by dividing the processing time of the best serial algorithm (T_s) by the processing time of the parallel version (T_p)[2] [9]:

$$SP = \frac{T_s}{T_p} \tag{6}$$

To obtain the best serial algorithm, we performed a set of experiments to determine the minimum parameters required by our algorithm to operate reasonably well [11]. From these experiments, we determined the following parameters for the sequential version of our algorithm: population size = 30, maximum number of generations = 5000, mutation rate = 0.05, crossover rate = 0.8, number of internal GA cycles = 15. The value of σ was defined by dividing the number of infeasible individuals in the current population by two. If there were three or less infeasible individuals in the current population, then all of them were considered as the value of σ. Additionally, we used binary tournament selection and two-point crossover.

[2]This expression assumes identical processors and identical input sizes (i.e., number of fitness function evaluations in our case).

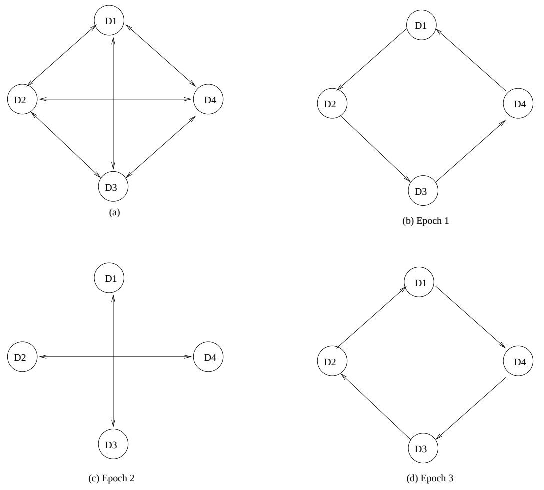

Figure 3: Figure that illustrates the topology and migration policies adopted by the parallel version of our algorithm assuming 4 processors. In (a) we can see the possible interconnections among deme. In (b), (c) and (d) we can see the migration possibilities for each deme. Note how these migration possibilities change at each epoch.

The parallel implementation used the following parameters: number of demes = number of processors available (between 2 and 4), deme size = total population size (30) divided by the number of processors available[3], epoch size = 500 generations, migration rate = 0.25. The migration policy adopted was the following: we select the best individual within a deme to be migrated plus an additional set of individuals randomly selected from the same deme. The candidates for replacement within each deme are also randomly selected. The topology adopted is a diamond in which migration is allowed to one different deme at a time (see Figure 3). Finally, the number of internal cycles in the parallel version is obtained by dividing the number used in the sequential algorithm (15) by the number of processors. Again, if this result is not an integer, the `ceiling` function is applied to it to make it an integer. In order to experiment with different evolutionary characteristics, we decided to use different setups in each deme (these setups were randomly determined) for the following parameters: mutation rate, crossover rate, type of selection (roulette-wheel, stochastic remainder with and without replacement, binary tournament and stochastic universal sampling), type of crossover (one-point, two-point, uniform).

As in our previous work (see [3]), our results are compared to the homomorphous maps of Koziel & Michalewicz [8], which is one of the best constraint-handling techniques known to date. Note that Koziel

[3]When this result is not an integer, we apply the `ceiling` operator to make it an integer value.

	KM	Serial	2 P SP=5.11	3 P SP=10.71	4 P SP=20.17
	$f(x)$	$f(x)$	$f(x)$	$f(x)$	$f(x)$
Mean	-14.7082	-14.5266	-14.7773	-14.5922	-14.8201
Best	-14.7864	-14.7841	-14.9995	-14.9995	-14.9978
Worst	-14.6154	-13.8417	-12.9892	-11.8787	-12.9930
Std. Dev.	N/A	0.2335	0.5103	0.8797	0.4830

Table 1: Comparison of results for the first example. P indicates the number of processors used and SP refers to the speedup achieved. N/A = Not Available.

& Michalewicz [8] performed $70 \times 20,000 = 1,400,000$ fitness function evaluations to solve each of the experiments previously described. Our algorithm, in contrast, performs only $150,000$ fitness function evaluations in each case (for both the sequential and the parallel version).

6 Comparison of Results

Table 1 shows the results produced by the method of Koziel & Michalewicz (KM), the sequential version of our algorithm and three versions of the parallelized algorithm for the first example. Each of these three parallel versions only differ in the number of processors employed. In this case, our serial implementation generates results inferior to KM, but not too far from the optimum. However, the parallel versions improve these results (they practically achieve the optimum on average), but with a higher performance variability (i.e., a higher standard deviation). More remarkable is, however, the considerable speedup achieved as more processors are used.

Table 2 shows the results for the second example. In this case, our serial implementation generates practically the same results that KM. The parallel versions maintain similar results but with a lower standard deviation. The speedup achieved in this case is once more, remarkable.

Table 3 shows the results for the third example. In this case, our serial implementation performs quite similarly to KM. The parallel versions obtain similar results but with a higher standard deviation. The speedup achieved in this case is, again, remarkable.

7 Conclusions and Future Work

We have presented a constraint-handling approach based on emulations of the immune system (particularly, using the negative selection model) that was incorporated into a genetic algorithm used for global optimization. The proposed algorithm does not use a penalty function and it is conceptually very simple to implement. Two versions of the proposed algorithm were compared in this paper: a serial version and a parallel version that uses islands.

Both versions of the proposed approach seem to be competitive with a state-of-the-art constraint-handling technique. Our approach (in all of its variants) produced reasonably good results with respect to the homomorphous maps of Koziel and Michalewicz [8] at a fraction of its computational cost. Whereas the homomorphous maps perform over $1,000,000$ fitness function evaluations per run, our approach only performs $150,000$ fitness function evaluations per run (in all our versions of the algorithm). Furthermore, our parallel versions produced remarkable speedups (in one test function, they performed up to 20 times faster than the serial version) while producing similar results. A note must be made regarding these speedups though. Since we used different selection operators and different crossover and mutation rates, we expected that the parallel version of our algorithm would behave differently. Therefore, the computation of the speedup may be somewhat misleading. Although other evolutionary computation researchers have reported superlinear speedups when using evolutionary algorithms (see for example [1]) mainly produced by increased selection pressure of migration, the speedups obtained in our case seem excessive. This behavior may be explained by the extra diversity introduced in the demes as an effect of the diverse selection schemes and crossover and mutation rates introduced. However, in any case the time gains seem too large at least in some cases and more analysis is still required.

Our current work involves more validation of the algorithm and of its sensitivity to its parameters (more experiments are required, involving more test functions and more detailed statistical analysis).

8 Acknowledgments

The first author acknowledges support from CONACyT through the NSF-CONACyT project number 32999-A. The second author acknowledges support from CONACyT through a scholarship to pursue graduate studies in Computer Science at the Sección de Computación of the Electrical Engineering Department at CINVESTAV-IPN.

	KM	Serial	2 P SP=1.92	3 P SP=4.5	4 P SP=8.87
	$f(x)$	$f(x)$	$f(x)$	$f(x)$	$f(x)$
Mean	-30655.30	-30654.98	-30662.32	-30648.86	-30657.03
Best	-30664.50	-30665.51	-30665.00	-30664.19	-30664.62
Worst	-30645.90	-30517.44	-30652.21	-30604.85	-30634.81
Std. Dev.	N/A	32.67	2.99	18.05	8.96

Table 2: Comparison of results for the second example. P indicates the number of processors used and SP refers to the speedup achieved. N/A = Not Available.

	KM	Serial	2 P SP=2.85	3 P SP=5.8	4 P SP=13.48
	$f(x)$	$f(x)$	$f(x)$	$f(x)$	$f(x)$
Mean	681.16	681.6192	681.6655	682.5332	682.5125
Best	680.91	680.9599	680.7499	680.9414	681.2212
Worst	683.18	683.7651	683.2584	687.1950	686.8906
Std. Dev.	N/A	0.7733	0.6910	1.8856	1.2949

Table 3: Comparison of results for the third example. P indicates the number of processors used and SP refers to the speedup achieved. N/A = Not Available.

Bibliography

[1] Erick Cantú-Paz. Migration Policies, Selection Pressure, and Parallel Evolutionary Algorithms. *Journal of Heuristics*, 7(4):311–334, 2001.

[2] Carlos A. Coello Coello. Theoretical and Numerical Constraint-Handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art. *Computer Methods in Applied Mechanics and Engineering*, 191(11–12):1245–1287, January 2002.

[3] Carlos A. Coello Coello and Nareli Cruz Cortés. Use of Emulations of the Immune System to Handle Constraints in Evolutionary Algorithms. In Cihan H. Dagli, Anna L. Buczak, Joydeep Ghosh, Mark J. Embrechts, Okan Erson, and Stephen Kercel, editors, *Intelligent Engineering Systems through Artificial Neural Networks (ANNIE'2001)*, volume 11, pages 141–146. ASME Press, July 2001.

[4] Leandro N. de Castro and F. J. Von Zuben. The Clocal Selection Algorithm with Engineering Applications. In *Proc. of GECCO'00, Workshop on Artificial Immune Systems and Their Applications*, pages 36–37, 2000.

[5] S. Forrest, A.S. Perelson, L. Allen, and R. Cherukuri. Self-nonself discrimination in a computer. In *IEEE Symposium on Research in Security and Privacy*, pages 202–212, Oakland, CA, May 16-18 1994. Springer-Verlag.

[6] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Co., Reading, Massachusetts, 1989.

[7] P. Hajela and J. Lee. Constrained Genetic Search via Schema Adaptation. An Immune Network Solution. *Structural Optimization*, 12:11–15, 1996.

[8] Slawomir Koziel and Zbigniew Michalewicz. Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

[9] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms.* The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1994.

[10] Zbigniew Michalewicz and Marc Schoenauer. Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[11] Erick Cantú Paz. *Efficient and Accurate Parallel Genetic Algorithms.* Kluwer Academic Publishers, Massachusetts, USA, 2000.

[12] Jon T. Richardson, Mark R. Palmer, Gunar Liepins, and Mike Hilliard. Some guidelines for genetic algorithms with penalty functions. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 191–197, George Mason University, 1989. Morgan Kaufmann Publishers.