Genetic Algorithms and Simulated Annealing: A Marriage Proposal

Dan Adler
Tudor Investment Corp.,
One Liberty Plaza,
NY, NY 10006
adan@tudor.com

Abstract—Genetic Algorithms (GA) and Simulated Annealing (SA) have emerged as the leading methodologies for search and optimization problems in high dimensional spaces. Previous attempts at hybridizing these two algorithms have been cumbersome and required major changes to both. In this paper we propose a simple scheme of using Simulated-Annealing Mutation (SAM) and Recombination (SAR) as operators in a standard GA environment. The operators use the SA stochastic acceptance function internally to limit adverse moves. This is shown to solve two key problem in GA optimization: populations can be kept small, and hill-climbing in the later phase of the search is facilitated. The implementation of this algorithm within an existing GA environment is shown to be trivial, allowing the system to operate as pure SA (or iterated SA), pure GA, or in various hybrid modes. Performance of the algorithm is tested on various large-scale applications, including DeJong's functions, a 100-city traveling-salesman problem, and the optimization of weights in a feed-forward Neural Network. The hybrid algorithm is seen to improve on pure GA in two ways: better solutions for a given number of evaluations, and more consistency over many runs.

1. Introduction

Over the last decade, *Genetic Algorithms* (GA) [1] have emerged as a leading tool for optimization of arbitrary functions and for guided search problems in high dimensional spaces. GA's are typically comprised of two types of operations: *mutation* and *crossover* which are repeatedly applied to a population of *chromosomes*, each of which encodes a possible solution to the given problem. GA's have been successfully applied to many theoretical optimization problems [2] and several [3] industrial applications.

Simulated Annealing (SA) [4] is another algorithm which is popular in heuristic optimization. SA belongs to a class of algorithms called *probabilistic hill-climbing* [5] which dynamically alter the probability of accepting inferior solutions. The SA algorithm is especially popular in the field of VLSI design [6] where it has been successfully applied to the optimization of extremely high-dimensional problems such as placement and global routing of interconnect layers in VLSI chips [7] which contain *tens or hundreds of thousands* of parameters to be optimized.

Since neither of the two algorithms seems to be universally preferred for all problems, researchers have often resorted to building a large battery of optimization algorithms [8] and finding, through experimentation, which tool best fits the problem at hand. This provides the basic motivation for trying to merge GA and SA into a single software module, which can be configured to run as pure GA, pure SA (or iterated SA), as well as a variety of hybrid modes.

2. A Framework for Comparison

A Genetic Algorithm may be specified as follows:

$$GA = (N_{pop} , N_{gen} , \Omega , f_{eval} , f_{sel})$$

where N_{pop} is the number of elements in the population, N_{gen} is the number of generations, Ω is the set of operators and their probabilities, f_{eval} is the evaluation or fitness function, and f_{sel} is a reproduction selection rule. We are not concerned here with the actual encoding of a solution as a chromosome. We assume that the *same* encoding of solutions can be used in the GA and the SA algorithm. One possible implementation of a GA is as follows:

```
repeat N_gen times {
   fitness_rank(pop_g)
   repeat N_pop times {
      s1 = select(pop_g)
      s2 = select(pop_g)
      op = op_select(Ω)
      pop_{g+1} ← op(s1, s2)
   }
   pop_g ← pop_{g+1}
}
```

The *fitness_rank* function calls f_{eval} to rank the whole population for selection, and the *select* function uses f_{sel} to select members of the population for reproduction.

Similarly, a Simulated Annealing algorithm might be specified as follows:

$$SA = (N_{pop} , \tau , f_{eval} , \mathbf{q})$$

where $\tau=T_0,T_1, \ldots ,T_K$ is the annealing schedule, f_{eval} is

0-7803-0999-5/93/$03.00 ©1993 IEEE

the evaluation function (which may be identical to the function used by the GA to find the relative fitness of a solution) and **q** is an exploration matrix. Each entry in the matrix represents a transition probability q_{ij} between two possible solutions to the problem, defined as:

$$q_{ij} = P(s_{n+1}=j \mid s_n=i) \tag{2.1}$$

which is the probability of moving to solution j from solution i at the current temperature. The algorithm proceeds by generating solutions $s_0, s_1, \ldots, s_n, \ldots$ such that a Markov chain is formed at each temperature level. In most implementations of the SA algorithm $N_{pop} = 1$. The following pseudo-code fragment [6] is a template for SA optimization:

```
T = T0
for(k=0; k<K; k++) {
   repeat {
      foreach s in pop_g {
         s' = mutate(s, T)
         if (accept(s', s, T)) pop_{g+1} ← s'
         else pop_{g+1} ← s
      }
      pop_g ← pop_{g+1}
   } until "local convergence"
   T = adapt(T_k)
}
```

The *mutate* function in SA optimization is an interpretation of the probability distribution P in equation (2.1). It represents the joint multivariate distribution of all possible states of the system, which, based on arguments from statistical physics [4] was originally taken to be the *Boltzman Distribution*. In the case of discrete parameters (combinatorial optimization), most authors [6] pay no attention to the distribution properties of the mutation (or perturbation) operations. The candidate solution, s' is obtained from s by a problem-specific heuristic. For the continuous case, s' is typically obtained as $s + \Delta s$ where Δs is generated from an underlying distribution based on the current temperature. Classic SA (CSA) typically uses the Boltzman distribution, mentioned earlier, to generate Δs, whereas Fast SA (FSA) uses a multivariate Cauchy [9] distribution.

The underlying distribution places a lower bound [10] on the annealing schedule: T should change no faster than $T_0/\ln(k)$ for CSA and T_0/k for FSA, and the system must reach a steady-state between adaptations. This is required to guarantee ergodic sampling of the parameter space. In *homogeneous* SA's—an explicit convergence test is applied (resulting in a series of Markov chains—each representing a single temperature), whereas in *inhomogeneous* SA's the inner loop might simply be executed some fixed number of times. The asymptotic convergence of the latter case is proved [10] based on the theory of in-homogeneous Markov chains, where the exploration matrix varies with temperature.

The most important, and intuitively appealing, aspect of the SA algorithms is the conditional acceptance function, where the probability of accepting an inferior solution s' over s is given by: $exp(-\Delta f/T)$, where T is the current temperature and Δf is the difference between $f_{eval}(s')$ and $f_{eval}(s)$ in the up-hill direction (for a global minimum problem). This allows the SA algorithm to escape from local extrema at the early stages of the search, and to efficiently hill-climb as the temperature approaches zero.

3. Limitations of pure GA's

While GA's are very popular as arbitrary function optimizers over low-dimensional spaces, they are not as popular as SA in the VLSI CAD community. This is mainly due to the GA's need to maintain a large population of solutions. For example, each solution in a VLSI place-and-route application [7] is an encoding of the placement of all cells on a chip and the routing of all interconnect on the chip. This may consume *mega-bytes* of memory for the encoding of *a single solution*. It would be impractical to manipulate a large population of candidate solutions. The same may be true when using a GA to optimize the weights in a large Neural Network, or to optimize the structure of the network.

Another problem frequently found in GA optimization is premature convergence. This is typically the result of the extreme reliance on crossover. The dominance of crossover can result in stagnation as the population becomes more homogeneous, and the mutation rate is too low to move the search to other areas.

Another well-known problem with GA optimizers is that they are quite poor at hill-climbing. This manifests itself in low accuracy in many real-valued problems. The main reason for this is, again, the extremely low mutation-rate, typically [2] one to two orders of magnitude lower than the crossover rate.

Ackley [11] has addressed this problem by developing SIGH, a connectionist algorithm that performs a form of iterated stochastic hill-climbing in a genetically-encoded environment. His analysis shows that the hill-climbing component can significantly improve the speed and accuracy of the search.

4. Previous Work on Merging GA with SA

Sirag and Weisser [12] have proposed a thermodynamic genetic operator that incorporates an annealing schedule into the control loop of a GA. In this scheme, the probability of mutation is given by $exp(-\theta_m/T)$, where θ_m is a constant threshold value, and T is the current temperature. Similarly, other operators have their own threshold

values. This method uses the annealing schedule to control the probability of *applying* an operator rather than the probability of *accepting* the mutated solution based on its merit. Thus, both the GA and the SA algorithms used are non-standard, and do not retain their separate identities. The above work falls into the category of adapting operator probabilities with time. More recent work on this subject was presented by Davis [13] where operator probabilities evolve based on assignment of credit from previous generations. These approaches do not solve the basic need of the GA to move in large leaps early on (to avoid stagnation), and to move in small leaps later on (to allow hill-climbing).

Controlling the magnitude of mutation is a central theme in *Evolution Strategies* [14] (ES). This class of algorithms can be classified as a real-valued GA, where Davis' heuristic "creep" operators [3] are replaced by SA-flavored Gaussian mutations $\Delta s = N(0,\sigma)$, and σ is dynamically altered based on the population fitness. The algorithm is shown [15] to outperform standard GA's in many cases, and has been extended to use recombination operators as well as mutations.

A different approach called SAGA [16] separates each generation into two stages: a GA stage and an SA stage. First, the GA is used to evolve a set of solutions, and then a modified SA is applied to further refine the solution. Both the GA and the SA algorithms used in SAGA are extensively modified and non-standard.

Ackley's work [11] mentioned earlier, can also be viewed as a hybridization of GA with SA—with a connectionist twist. The algorithm consists of processing units which perform a stochastic acceptance function over binary-coded input vectors to generate reinforcement or inhibition signals to other units. Ackley demonstrates that the algorithm out-performs both GA and SA on a large variety of problems. However, it is very difficult to identify the exact relation to either algorithm—and the components cannot be separated.

The current work tries to merge SA and GA in a more standard setting. Each algorithm maintains its own identity, and each can be viewed as an extension of the other. We are not concerned here with the specific form of the mutation or recombination operators. Rather, we focus on a method that can be used either with real-valued mutations such as "creep", CSA, FSA or ES mutations, or standard bit-string mutations and recombinations—although, we will show later that there are theoretical reasons to prefer some forms over others.

Our goal is to relieve some of the pressure from the genetic selection mechanism and transfer some of the selectivity to the operators themselves, thus allowing the algorithm to adjust its degree of selectivity as the search progresses. This allows us to use a very high mutation-rate—which improves the hill-climbing ability of the algorithm, and reduces the chances of "getting stuck" at local extrema. We use the classic interpretation of SA as adapting the probability of *accepting* an inferior solution, and consider the consequences of extending the standard GA operators by using the SA *accept()* function *inside* the operators.

5. Simulated-Annealing Operators: SAM and SAR

The method we have used to combine SA with GA is by replacing all (or some) of the mutation and recombination operators by SA operators: SAM (SA-Mutation) and SAR (SA-Recombination). The SAM operator works exactly like a standard mutation operator: it gets a solution as input, mutates it and returns a solution as output. The difference is that internally, the SA operator can call the evaluation function, and use the result to decide whether to accept the mutated solution, or just stay with the previous solution:

```
SAM(s, T) {
    s' = mutate(s, T)
    if (accept(s', s, T)) return s'
    else return s
}
```

The function *accept*, besides applying the standard SA acceptance condition, does two additional things: it sets $s'.eval$ to the value of the evaluation function $f_{eval}(s')$, and sets a flag $s'.needs\_eval$ to 0 - so that the *fitness_rank* function will not call the evaluation function again for this solution. This guarantees that the hybrid algorithm will call the evaluation function the same number of times as the pure GA, for a given N_{pop} and N_{gen}. The annealing temperature is lowered in an in-homogeneous way: between generations. Thus, every one (or more) generations, the temperature is lowered, making the SA operators more selective about the mutations it accepts. The SAR operator is very similar. First the crossover is applied, and then each of the children is compared to the best of the two parents for acceptance.

As GA operators, SAM and SAR can completely replace the existing operators, or coexist with them. This can be controlled by their relative probabilities of selection. To understand the effects of this scheme it is useful to consider some extreme cases. First, note that setting the probability of the SAM/SAR operator to 0 leaves us with the standard GA, as does disabling the annealing schedule. Second, note that setting all the other operator probabilities to zero or, alternatively, disabling the selection mechanism, leaves us with a standard SA algorithm. With $N_{pop} > 1$, this becomes an iterated SA algorithm.

The interesting property of this implementation is that between these extreme cases, we can easily and con-

tinuously get a variety of intermediate situations. For practical purposes, we consider each separate component of the algorithm as a mere heuristic, which can be mixed-and-matched with other heuristics to produce a hybrid algorithm suited to a specific problem (since both GA's and SA's are *ultimately* just heuristics due to finite computation times). However, with some restrictions, the resulting hybrid algorithm can be made to satisfy all the theoretical assumptions of each of the two underlying algorithms, thereby avoiding the need to develop a new theoretical foundation.

6. Analysis of Hybrid Algorithm

The SAM/SAR operators, while trivial to implement, offer some important improvements over the standard GA algorithm. First and foremost, these operators bring *hill-climbing* into the realm of GA's without any explicit cross-generational breeding as in Eshelman's CHC [17] algorithm. As the temperature decreases, the operators will inevitably only pass through small adverse changes—while freely accepting improvements. This allows us to set the SAM selection probability to be very high. There is no need to make it as small as in standard GA's.

An interesting property of the merged algorithm is the interaction between the GA reproduction-selection mechanism and the SA algorithm carried out by the operators. From the point of view of the GA, the SAM/SAR operators behave the same as their non-SA counterparts. Nothing in the analysis of *schemata* presupposes any specific behavior on the part of the operators. So, clearly, the GA's convergence properties are not adversely affected.

The more interesting point of view is from the SA's side. Instead of evolving a single solution, it is now dealing with a collection of related solutions. Furthermore, these solutions are presented to the SAM/SAR operators based on the fitness selection mechanism. Each solution is presented to the operators several times, based on its relative fitness. This defines a neighborhood around each such solution. The size of the neighborhood varies according to its *average* fitness. The concept of an optimal neighborhood around a solution is a central problem in the SA literature [18] and one of the heuristics for solving it [19] is based on a mechanism similar to fitness selection. Thus, the selection mechanism of the GA guides the SA operators into working within neighborhoods of useful solutions, and lets the operators search those neighborhoods efficiently.

The convergence properties of the SA operators are not influenced by the GA control loop around them. Each member of the current generation is created from one or more of the previous generation's solutions—so the Markov chain is not broken. The case for recombination is not quite so straightforward to analyze since the Markov process must now be defined over pairs of states—but the principle is the same. The formal analysis of the merged algorithm is similar to that of a *parallel* SA algorithm [20] that represents N_{pop} independent periodically-interacting searches. In this case the searches are much less independent: both the GA selection mechanism and the recombination operations improve the efficiency of the search.

Since the convergence properties of the SA algorithm are closely tied to both the annealing schedule and the method of mutation—the hybrid algorithm is better geared towards a real-valued representation. Thus far, most implementations of real-valued GA's (with the exception of Evolution Strategies) have used heuristic mutation operators inspired by the more traditional bit-mutations. The merging of SA with GA in the current algorithm suggests that the exact form of mutation *is* important, and may have a dramatic effect on how the algorithm navigates through the problem-space.

For example, FSA, mentioned earlier, uses an underlying Cauchy distribution. Since a multivariate Cauchy *cannot* be generated by independently varying each dimension as a univariate Cauchy distribution, the mutation must consider all dimensions at once. One way to derive a Cauchy mutation operation [21] is by first generating an n-dimensional uniform vector and multiplying it by $\sqrt{\frac{X}{(1-X)}}$, where $X \sim \text{Beta}(\frac{n}{2},\frac{1}{2})$ is a scalar random variable, and by σT_k, a multiple of the current temperature.

A final point to consider is population size. The merged algorithm can work with any size population. A small population will accentuate the SA properties of the algorithm (with the added benefit of crossover), while a large population will accentuate the exponential preference for better schemata.

7. Performance

We have compared the proposed algorithm against a standard GA on a large variety of problems. The results given below were generated using the test suite of objective functions provided by Baeck [22] as part of the GENEsYs package (using his C-code for the functions, but *not* for the algorithms). Since we are interested in how the SAM/SAR operators can enhance an existing GA, we did not run both algorithms to convergence. Rather, we decided a-priori on a number of generations, and ran both algorithms for the same N_{gen}, which results in the same number of objective-function evaluations. We stored the best solution at the end of each run, and averaged them over 50 different runs. Since we are interested in large problems, we intentionally set the population to a low number (3). The dimension of the problems was kept high, mostly at 50 (a notable exception is F10 which is a 100-

city Traveling Salesman Problem). We only ran the functions for which the true minimum solution is given in [22] so that we can compare how far both algorithms are from the global optimum.

The annealing schedule used for SAR/SAM was linear k-period annealing [10] based on the number of generation the algorithm is asked to run, with T_0 set as 100 times the worst solution in the first generation. The probabilities for SAM and SAR were set such that recombination occurs twice as often as mutation. In the pure GA version, recombination was more than an order of magnitude more likely than mutation. In all experiments we used an *elitist* strategy, copying the best member of the population to the next generation.

TABLE I
Pure GA vs. hybrid algorithm.

	Dim	Min	μ(GA)	σ(GA)	μ(HYB)	σ(HYB)
F1	50	0	10^{-7}	10^{-7}	10^{-10}	10^{-10}
F2	50	0	48.3	0.8	48.1	0.5
F3	50	0	268.7	2.3	270.3	0.1
F4	50	0	31.3	1.2	30.2	0.7
F5	2	1	0.9	10^{-6}	0.9	10^{-7}
F6	50	0	380.6	264.5	265.6	155.5
F7	50	0	53.3	28.7	43.9	7.9
F10	100	21285	64400.0	7245.0	56865.0	4441.0
F12	50	0	34.5	15.4	37.2	3.2
F15	50	0	10^{-6}	10^{-6}	10^{-8}	10^{-9}
F18	4	0	6.2	2.8	5.1	2.8

$N_{pop} = 3$, $N_{gen} = 5000$. (for F10 $N_{gen} = 10000$)
μ and σ are calculated over 50 runs.

As expected, the hybrid algorithm dramatically outperforms pure GA when there are hills to be climbed, e.g. F1 ($\sum x_i^2$), F15 ($\sum i x_i^2$). On F3, which is a step-function, both algorithms are still equally far from the solution. F4 is ($\sum i x_i^4 + rand(0,1)$, which means its minimum is disguised by noise. F5 is Shekel's Foxholes (only defined for 2 dimensions).

In almost all cases, the standard deviation of the solutions was considerably tighter for the hybrid algorithm. While the pure GA occasionally finds better solutions, the hybrid algorithm is more consistent. For example, on F12, which is the Hamming distance from the zero vector, the hybrid solution is worse than the pure GA, but the standard deviation is an order of magnitude better.

F10 is Krolak's 100-city Traveling-Salesman Problem. This is by far the largest problem in the suite. In this case, clearly, the number of iterations performed was not high enough to approach the optimum—but this is characteristic of many real-life problems where there is no hint as to how many iterations are "enough". In this case, after 10000 generations, the relative error of the hybrid solution is 35% better than that of the pure GA, with a 30% lower standard deviation over 50 runs.

As a more realistic example, we considered the problem of training a feed-forward Neural network to predict the chaotic logistic map: $x_{t+1} = 4x_t(1-x_t)$. The network has two inputs: x_t and x_{t-1}, one fully-connected hidden-layer of 10 sigmoidal units, and one output: x_{t+1}. This involves optimizing about 50 weights. Backpropogation is able to optimize the weights in 450 iterations through 250 input vectors, achieving an out-of-sample R^2 of 0.98.

The following table shows the results of using GA with and without the SAM/SAR algorithms to optimize the weights. The objective function in this case is the mean-squared-error of running the network forward with the given weights, compared to the true output for the input sample. After the learning phase, we ran the network forward on an out-of-sample set of 750 vectors.

TABLE II
Out-of-sample R^2 for Neural Network. $N_{pop} = 3$.

Gen	R^2(GA)	R^2(HYB)
250	0.08	0.81
500	0.75	0.94
750	0.88	0.95
1000	0.93	0.98

In this case, the hybrid algorithm's hill-climbing ability gives it a strong advantage over the pure GA. After 250 generations, which is 750 evaluations, the hybrid algorithm's out-of-sample performance is an order of magnitude better than the pure GA. The advantage of using the GA algorithm with SAM/SAR over Backpropagation is that non-differentiable error measures and activation functions can be used.

8. Conclusions

We have presented a method of hybridizing GA's with SA's by replacing the standard mutation and recombination operators by their SA versions: SAM and SAR. These operators perform the classic SA acceptance function, and can otherwise be made identical to the standard operators. The hybrid algorithm can be configured to operate as pure GA, pure SA (or iterated SA), as well as a

variety of hybrid modes.

The analysis has shown that this results in a hybrid algorithm which preserves the advantages of GA while emphasizing hill-climbing at the later stages of the search, thus improving its accuracy on many real-valued problems. This was found to be especially true for high dimensional problems, using a small population of solutions (3-5). Even in cases where the accuracy is not improved over a standard GA, the hybrid algorithm exhibits more consistency across multiple runs.

The analysis also suggests that for real-valued GA's, there is much to be gained by using CSA/FSA mutations instead of simple heuristic operators. These operators are known to improve the convergence and speed of pure SA, and should be coupled with an appropriate annealing schedule.

We believe this work may contribute to making GA's more popular in VLSI place-and-route problems, and in optimization of weights in large Neural networks that use non-standard error measures.

References

1. John Holland, *Adaptation in Natural and Artificial Systems,* University of Michigan Press, Ann Arbor, 1975.
2. D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning,* Addison Wesley, 1989.
3. Lawrence Davis, *Handbook of Genetic Algorithms,* Van Nostrand Reinhold, New York, 1991.
4. S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by Simulated Annealing," in *Science*, vol. 220, pp. 671-680, May, 1983.
5. F. Romeo and A. Sangiovanni-Vincentelli, *Probabilistic Hill-Climbing Algorithms: Properties and Applications,* Computer Science Press, Chapel Hill, NC, 1985.
6. S. Nahar, S. Sahni, and E. Shragowitz, "Simulated Annealing and Combinatorial Optimization," in *Proc. of the 23th Design Automation Conf.*, pp. 293-299, June, 1986.
7. C. Sechen and A. Sangiovanni-Vincentelli, "TimberWolf3.2: A New Standard Cell Placement and Global Routing Package," in *Proc. of the 23th Design Automation Conf.*, pp. 432-439, June, 1986.
8. M. Bramlette and E. Bouchard, "Genetic algorithms in Parametric Design on Aircraft," in *Handbook of Genetic Algorithms*, pp. 109-123, Van Nostrand Reinhold, New York, 1991.
9. H. Szu and R. Hartley, "Fast Simulated Annealing," in *Physics Letters A*, vol. 122(3-4), pp. 157-162, 1987.
10. P. van Laarhoven and E. Aarts, *Simulated Annealing: Theory and Applications,* Kluwer Academic, 1987.
11. D. Ackley, *Stochastic Iterated Genetic Hillclimbing,* Ph.D. Diss., Dept.Comp. Sc. CMU-CS-87-107, Carnegie Mellon University,, March 1987.
12. D. Sirag and P. Weisser, "Toward a Unified Thermodynamic Genetic Operator," in *Proc. of the 2nd International Conf. on Genetic Algorithms*, pp. 116-122, 1987.
13. L. Davis, "Adapting Operator Probabilities in Genetic Algorithms," in *Proc. of the 3rd International Conf. on Genetic Algorithms*, pp. 61-70, 1989.
14. T. Baeck, F. Hoffmeister, and H.P. Schwefel, "A survey of Evolutionary Strategies," in *Proc. of the 4th International Conf. on Genetic Algorithms*, pp. 2-9, 1990.
15. D. Fogel, *System Identification Through Simulated Evolution,* Ginn Press, 1991.
16. D. Brown, C. Huntley, and A. Spillane, "A Parallel Genetic Heuristic for the Quadratic Assignment Problem," in *Proc. of the 3rd International Conf. on Genetic Algorithms*, pp. 406-415, 1989.
17. L. Eshelman, "The CHC Adaptive Search Algorithm," in *Foundations of Genetic Algorithms*, ed. G. Rawlins, pp. 265-283, Morgan Kaufman , 1991.
18. L. Goldstein and M. Waterman, "Neighborhood Size in the Simulated Annealing Algorithm," in *Simulated Annealing (SA) and Optimization: Modern algorithms with Applications*, ed. M. Johnson, pp. 409-424, American Sciences Press, 1988.
19. C. Tovey, "Simulated Simulated-Annealing," in *Simulated Annealing (SA) and Optimization: Modern algorithms with Applications*, ed. M. Johnson, pp. 389-408, American Sciences Press, 1988.
20. R. Azencott, *Simulated Annealing Parallelization Techniques,* Wiley, 1992.
21. L. Devroye, *Non Uniform Random Variate Generation*, Springer-Varlag, New York, 1986.
22. T. Back, *A User's Guide to GENEsYs 1.0,* University of Dortmund, Dept. of CS, (email: baeck@ls11.informatik.uni-dortmund.de), July, 1992.