Guided Crossover: A New Operator for Genetic Algorithm Based Optimization

Khaled Rasheed
shehata@cs.rutgers.edu

Haym Hirsh
hirsh@cs.rutgers.edu

http://athos.rutgers.edu:80/ shehata/papers.html

Abstract

Genetic algorithms (GAs) have been extensively used in different domains as a means of doing global optimization in a simple yet reliable manner. They have a much better chance of getting to global optima than gradient based methods which usually converge to local sub optima. However, GAs have a tendency of getting only moderately close to the optima in a small number of iterations. To get very close to the optima, the GA needs a very large number of iterations. Whereas gradient based optimizers usually get very close to local optima in a relatively small number of iterations.

In this paper we describe a new crossover operator which is designed to endow the GA with gradient-like abilities without actually computing any gradients and without sacrificing global optimality. The operator works by using guidance from all members of the GA population to select a direction for exploration. Empirical results in two engineering design domains and across both binary and floating point representations demonstrate that the operator can significantly improve the steady state error of the GA optimizer.

1 Introduction

Genetic Algorithms (GAs) [4] are search algorithms that mimic the behavior of natural selection. GAs attempt to find the best solution to some problem (e.g., the maximum of a function) by generating a collection ("population") of potential solutions ("individuals") to the problem. Through mutation and recombination (crossover) operations, better solutions are hopefully generated out of the current set of potential solutions. This process continues until an acceptably good solution is found. GAs have many advantages over other search techniques, including the ability to deal with qualitatively different types of domains, such as continuous variable domains, discrete or quantized variable domains, or mixed-type variable domains. In continuous domains though, local search methods such as gradient based algorithms tend to have an advantage over the GA, and that is their ability to get extremely close to local optima in a relatively small number of iterations. Most GAs, even the ones using floating point representation, need a large number of iterations to get very close to optima. On the other hand, GAs have a much higher probability of reaching the global optimum than local based methods. One classical approach to solve this dilemma is to use a GA optimizer to get to the "good region" and then run a local search method from then on. This approach is indeed very good, but it has many problems. One problem is how to decide that the GA has reached the "good region". The good region may be thought of as the basin of attraction of the global optimum. In the case of highly multi-modal spaces, this region can be very small. In this case, we have to run the local search method many times starting, say, from selected points of the final GA population. This can get very expensive. Another problem is that local search methods suffer from a "curse of dimensionality".

This paper presents a new crossover operator. we name it "Guided Crossover" (GC) capable of improving the steady state error of a GA optimizer (the distance between the best point found and the global optimum of the search space) in a continuous space, without having to use gradient information or gradient based methods.

We demonstrate the merit of our method in the GA optimization of realistic continuous-variable engineering design domains. In such domains (such as the two explored in Section 3) a design is represented by a number of continuous design parameters, so that potential solutions are vectors (points) in a multi-dimensional vector space. Determining the quality ("fitness") of each point involves the use of a simulator that computes relevant physical properties of the artifact and summarizes them into a single measure of merit. The simulator will often take a non-negligible amount of time, ranging from a fraction of a second to, in some cases, many days. Consequently, gradients are extremely expensive to compute. Moreover, the use of table lookup, and the high dimensionality and multi-modality of the search space make it very difficult to use gradient based methods with a high degree of reliability.

The remainder of this paper first presents a more detailed description of the new operator. We then present a number of experiments concerning the use of our operator on two realistic engineering tasks and across both binary and floating point representations. We conclude the paper with a discussion of related efforts and future work.

2 GA Architecture

The GA used in this research is described in detail in [8, 10]. Each individual in the GA population represents a parametric description of an artifact, such as an aircraft, or a process, with each parameter taking on a value in some continuous interval. The fitness of each individual is based on the sum of a proper measure of merit computed by a simulator (such as the takeoff mass of an aircraft), and a penalty function if relevant (such as to impose legal limits on the permissible noise of an aircraft). Operators are applied to elements of the population via some selection scheme. Here selection was performed by rank (rather than via the actual value of the fitness function on each individual) because of the wide range of fitness values caused by the use of a penalty function — rank selection prevents the first discovered evaluable/feasible points from dominating the population. A steady state GA model is used, in which existing points in the population are replaced by newly generated points via some replacement strategy. The replacement strategy used here takes into consideration both the fitness and the proximity of the points in the GA population, with the goal of selecting for replacement a point that both has low fitness and is relatively close to the point being introduced. The GA stops when either the maximum number of evaluations has been exhausted or the population completely loses diversity and practically converges to a single point in the search space. The GA architecture also includes a screening module (SM) and a diversity maintenance module (DMM) [9] which can both be turned on or off. The SM saves time but preventing the GA from evaluating points that are close to previously encountered bad points. It uses a case based learning technique to do this. The DMM maintains diversity by preventing the GA from evaluating candidate point which are extremely close to previously evaluated points. The DMM also attempts to restore diversity through a re-seeding operation if severe loss of diversity is detected in the early stages of the optimization.

The remainder of this section describes the new crossover operator in detail.

2.1 Guided Crossover

Guided Crossover (GC) works as follows:

1. One candidate point is selected from the GA population using the normal selection rule (by rank) and called $candidate_1$.

2. The second candidate point is also selected from the GA population but in a different way: for each point X in the GA population other than $candidate_1$ a quantity Q(X,$candidate_1$) is computed. Where

   $$Q(A,B) = \frac{(fitness(A) - fitness(B))^2}{distance(A,B)^2}$$

   A choice for X that maximizes Q(X,$candidate_1$) is taken to be $candidate_2$.

3. $candidate_1$ and $candidate_2$ are swapped if necessary, to make $candidate_1$ the point that has the higher fitness among the two.

4. The result of the crossover is a point along the line joining $candidate_1$ to $candidate_2$ which is selected at random from the small region around $candidate_1$ (the better point) as follows:

   $$Result = L * candidate_1 + (1 - L) * candidate_2$$

   where L is a uniformly distributed random number in the interval [1-x,1+y] and x and y are functions of the number of elapsed iterations I_e and the total allowed number of iterations I_t such that:

   $$x = \frac{0.2 * (I_t - I_e)}{I_t}$$

   $$y = \frac{0.5 * (I_t - I_e)}{I_t}$$

In words, GC examines all the directions that can be formed by joining the randomly selected first candidate point to all other points in the current GA population. The directions are ranked based on the contribution they give to the objective function when moving between the two end points relative to the distance between the end points. The best direction according to this ranking is chosen, and a small step is taken in this direction in the vicinity of its best end point. The magnitude of the step diminishes as the GA optimization progresses.

The guided crossover operator should not be used as the only crossover operator in a GA architecture, because it is greedy in nature. We propose using it as a substitute crossover operator only a fraction of the time. In the current implementation, a random choice is made in every iteration between GC and more conventional crossover operators. The probability of choosing GC is increased linearly from 0 to its maximum value as the number of iterations increases to its maximum allowed value. The maximum value was 0.2 in all experiments, except in the case of aircraft design using the binary-Coding GA where the maximum value was 0.4. The reason for doing this is that the performance without the GC was very unacceptable.

The intuition behind guided crossover is that it endows the GA with a way to get very close to the optimum once it is already near it — an advantage usually claimed for gradient-based methods over GAs — without the costly computation of gradients using potentially expensive evaluations in high-dimension spaces.

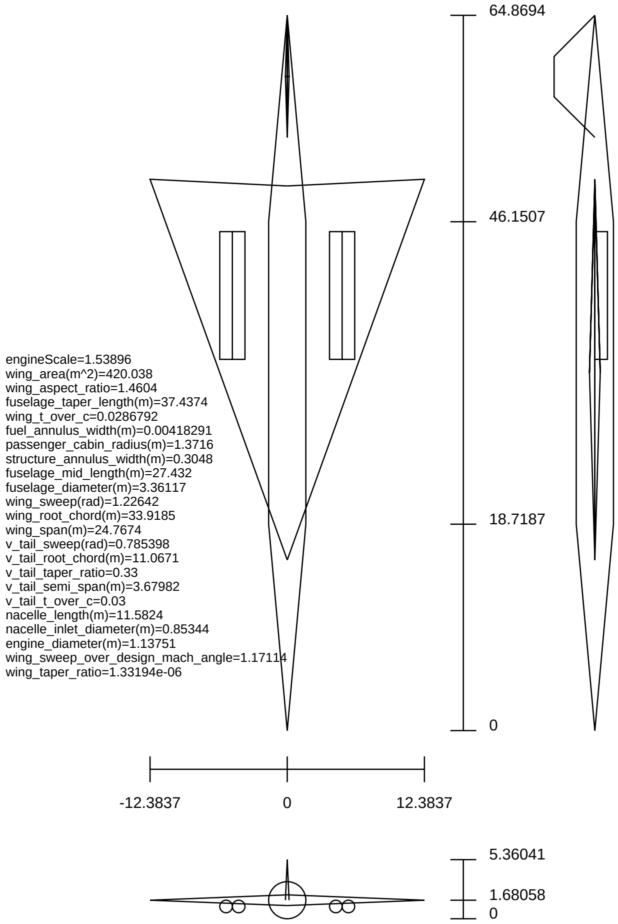

Figure 1: Supersonic transport aircraft designed by our system (dimensions in feet)

3 Experimental Results

To evaluate our GA adaptation we applied it to design problems in two domains, the conceptual design of supersonic transport aircraft, and the design of a three-stage membrane separation process. It should be noted that the superiority of the GA optimization approach to traditional optimization methods (including gradient based methods) in these two domains has been established in [8, 10] and in [7]. In this research we focused on the effect of the proposed crossover operator on the GA performance, rather than the overall GA behavior. This section discusses our results in these two domains.

Table 1: Aircraft Parameters to Optimize

No.	*Parameter*
1	exhaust nozzle convergent length
2	exhaust nozzle divergent length
3	exhaust nozzle external length
4	exhaust nozzle radius(r7)
5	engine size
6	wing area
7	wing aspect ratio
8	fuselage taper length
9	effective structural t/c
10	wing sweep over design mach angle
11	wing taper ratio
12	Fuel Annulus Width

3.1 Supersonic Transport Aircraft Design

Our first domain concerns the conceptual design of supersonic transport aircraft. We summarize it briefly here; it is described in more detail in [3]. Figure 1 shows a diagram of a typical airplane automatically designed by our software system. The GA attempts to find a good design for a particular mission by varying the aircraft conceptual design parameters in Table 1 over a continuous range of values.

The GA evaluates candidate designs using a multidisciplinary simulator. In our current implementation, the GA's goal is to minimize the takeoff mass of the aircraft, a measure of merit commonly used in the aircraft industry at the conceptual design stage. Takeoff mass is the sum of fuel mass, which provides a rough approximation of the operating cost of the aircraft, and "dry" mass, which provides a rough approximation of the cost of building the aircraft. A complete mission simulation requires about 1/5 second of CPU time on a DEC Alpha 250 4/266 desktop workstation.

The aircraft simulation model used is based on both implicit and explicit assumptions and engineering approximations and since it is being used by a numerical optimizer rather than a human domain expert, some design parameter sets may correspond to aircrafts that violate these assumptions and therefore may not be physically realizable even though the simulator does not detect this fact. For this reason a set of constraints has been introduced to safeguard the optimization process against such violations. In particular, a penalty function approach was used to incorporate the effect of constraint violations into the optimization: the penalty function was added to the takeoff mass value returned by the simulator and the resulting sum was the quantity that the optimizer actually minimizes (which therefore also serves as the fitness value assigned to each point of the GA population). The specific penalty function was simply a large constant multiplied by the sum of the amounts of constraint violation for all the violated constraints.

The presence of constraints induced a partition of the search space into three mutually exclusive regions:

1. Unevaluable points: These are points that represent designs that violate the model assumptions so much that the simulator cannot complete the simulation process to produce any significant information. For such points a very large fictitious takeoff mass is generated as the value of the objective function.

2. Infeasible evaluable points: These are points that represent unrealizable aircrafts but the type and extent of model violation is moderate enough for the simulator to complete its work and report the constraint violation information. As described above, a penalty function is added to the takeoff mass returned by the simulator to account for the model violation.

3. Feasible points: The simulator succeeds in evaluating the take off mass for such points and no violations occur in the process. The penalty function for these points is zero.

In the first experiment we used the traditional binary-encoding GA (bit-string representation for individuals, the classical bit-string crossover operator, the classical bit mutation operator). The results of our experiment are shown in Figure 2. Ten random populations of 120 points each were generated, and for each population the GA was allowed to proceed for 24000 iterations (an iteration denotes one call to the simulator, which takes, on average, 0.2 seconds) once with GC and once without it. In the iterations where GC was used a substitute crossover operator, the genotypes were converted to floating point and the genotype of the newborn was then converted back to the binary representation.

The graph plots the percent deviation of the best point found so far, from the global optimum[1] as the GA run progresses. Each curve represents the average of the runs from the ten starting populations.

The figure clearly indicates how GC decreased the steady state error of the GA from an average of around 16% to less than 5%. One other advantage of using GC that is not demonstrated by the figure is that all runs with GC terminate well before the 24000 iteration limit. The average number of iterations for the runs in which GC was used is about 15000. On the other hand, all the runs with no GC terminated by exhausting the 24000 iteration cutoff. The figure also demonstrates how GC not only improved the steady state error, but also the overall performance of the GA. If the search was to be terminated, at say 10000 iterations, the performance of the runs which used GC would still be reasonably acceptable, but not without GC.

The second experiment was very similar to the first experiment (the same starting populations were used). The only difference was that the floating point representation described in [8, 10] was used instead

[1]Note that since the evaluation function is the output of a numerical simulator we refer to the best design found throughout all optimization attempts as the global optimum, where in fact we have no formal proof that it is indeed global.

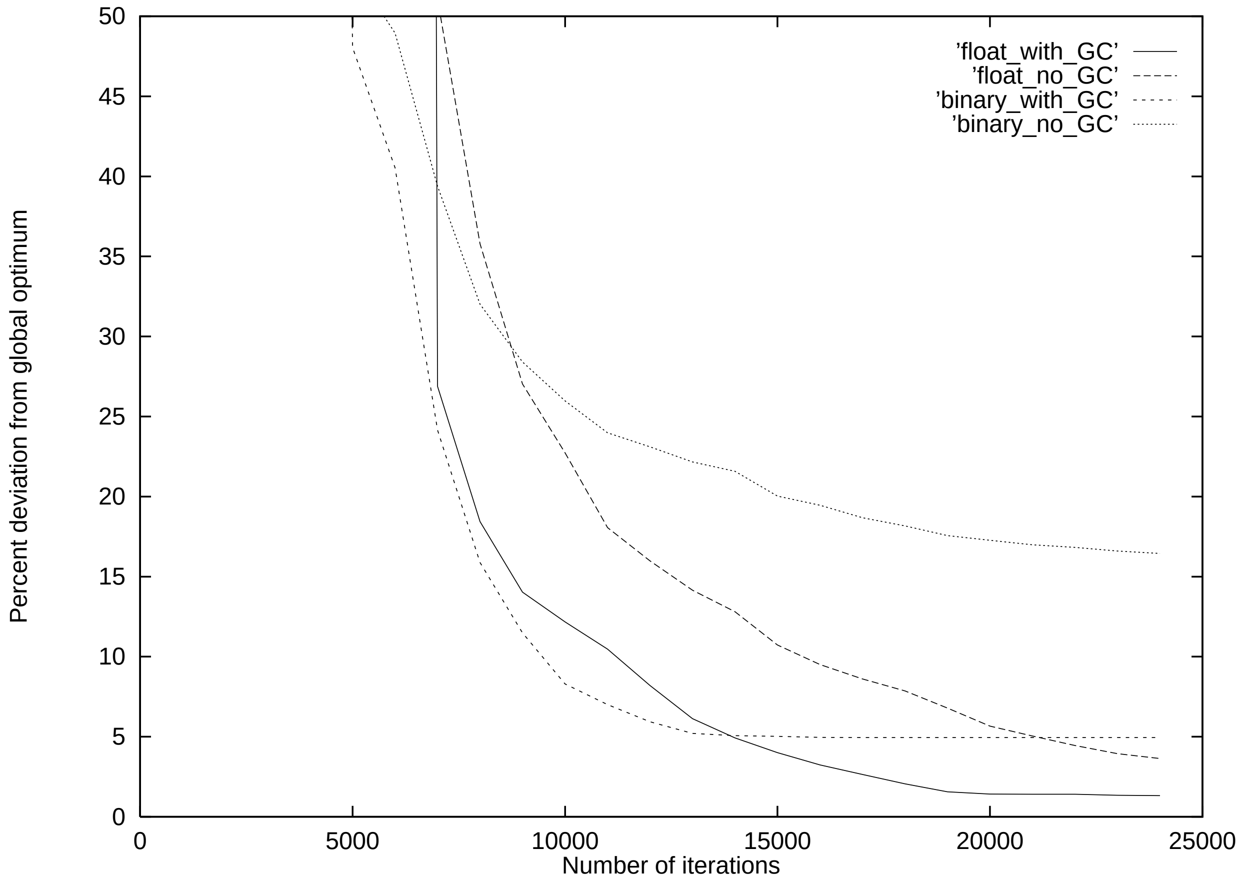

Figure 2: Effect of guided crossover in the Aircraft Design Domain

of the binary representation. The crossover operator was line crossover and the mutation operator was the shrinking window mutation described in [8, 10]. Line crossover works by joining a line between the two parent points and picking a point on that line or its extensions to be the newborn. Shrinking window mutation is done by randomly perturbing the newborn. The perturbation window shrinks as the optimization progresses. The result of this experiment is also shown in Figure 2. The figure again demonstrates the advantage of using GC.

3.2 Three-stage membrane separation process

In this domain the GA attempts to optimize the design of a Three-stage membrane separation process using a mathematical programming model. The problem has 13 continuous domain design variables and 13 inequality constraints. Thus the space is partitioned into feasible and infeasible points. All points are evaluable. A more detailed description of this domain can be found in [11].

Similar to the aircraft design domain, experiments were done using ten random starting populations. The results are shown in Figure 3 and Figure 4 . Again, it is clear in this domain how much GC enhanced the performance in both GA architectures[2]. The reason we include Figure 4 is to show that the diversity maintenance module is not necessary for the success of the GC operator. The SM and the DMM improved the overall performance of the GA in both architectures, but the GC's contribution was still substantial even when SM and DMM were turned off.

4 Final Remarks

A new crossover operator for continuous space search spaces has been presented, based on maintaining a large sample of previously evaluated points. Experimental results demonstrated the merit of using the

[2] The reader is cautioned not to under estimate the overall performance of the GA in this domain. According to [7], gradient based methods stop with more than 60% deviation from the global optimum

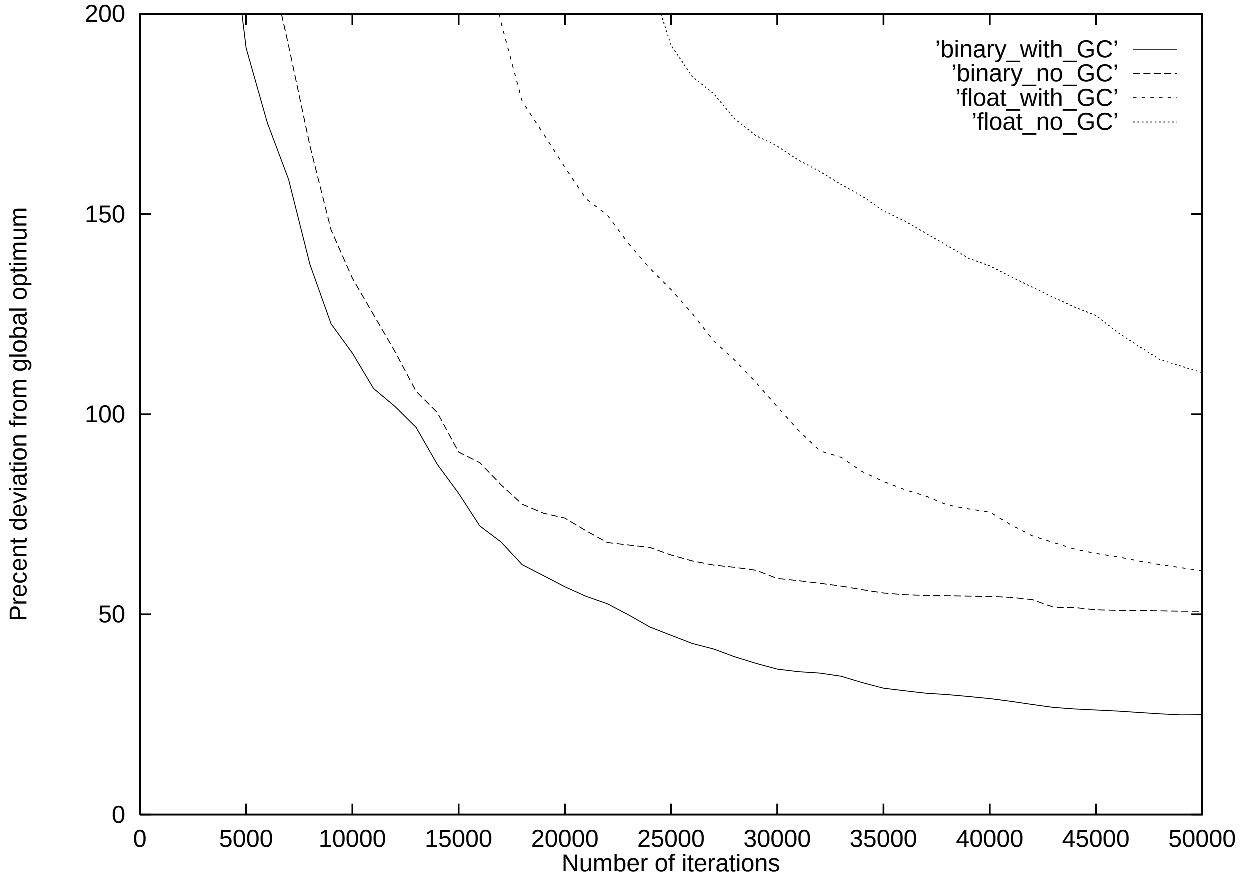

Figure 3: Effect of guided crossover in the Membrane Separation Process Design Domain

new operator in the domains of aircraft design optimization and the design of a membrane separation process and across binary and floating point representations.

Although a great deal of work has been done in the area of numerical optimization algorithms [5], not much has been published about the particular difficulties of attempting to optimize functions defined by large "real-world" numerical simulators. A number of research efforts have combined AI techniques with numerical optimization [12, 6, 7, 1, 13, 2], and although a GA was used in some of these efforts [6, 1, 7], this typically represented the use of an off-the-shelf GA.

The new crossover operator contributed a lot of improvement to the GA optimization. The idea of taking a little step in a promising direction, proved very powerful. In GC, the promising direction is selected by joining lines between points of the GA population and choosing the most promising line. However, a very interesting idea is to create new directions by combining several such lines. We plan on exploring this method in the near future.

Acknowledgments

We thank our aircraft design expert, Gene Bouchard of Lockheed, for his invaluable assistance in this research. We also thank all members of the HPCD project, especially Andrew Gelsey, Donald Smith, and Keith Miyake. This research was partially supported by NASA under grant NAG2-817 and is also part of the Rutgers-based HPCD (Hypercomputing and Design) project supported by the Advanced Research Projects Agency of the Department of Defense through contract ARPA-DABT 63-93-C-0064.

References

[1] E. E. Bouchard. Concepts for a future aircraft design environment. In *1992 Aerospace Design Conference*, Irvine, CA, February 1992. AIAA-92-1188.

[2] G. Cerbone. Machine learning in engineering: Techniques to speed up numerical optimization. Technical Report 92-30-09, Oregon State Uni-

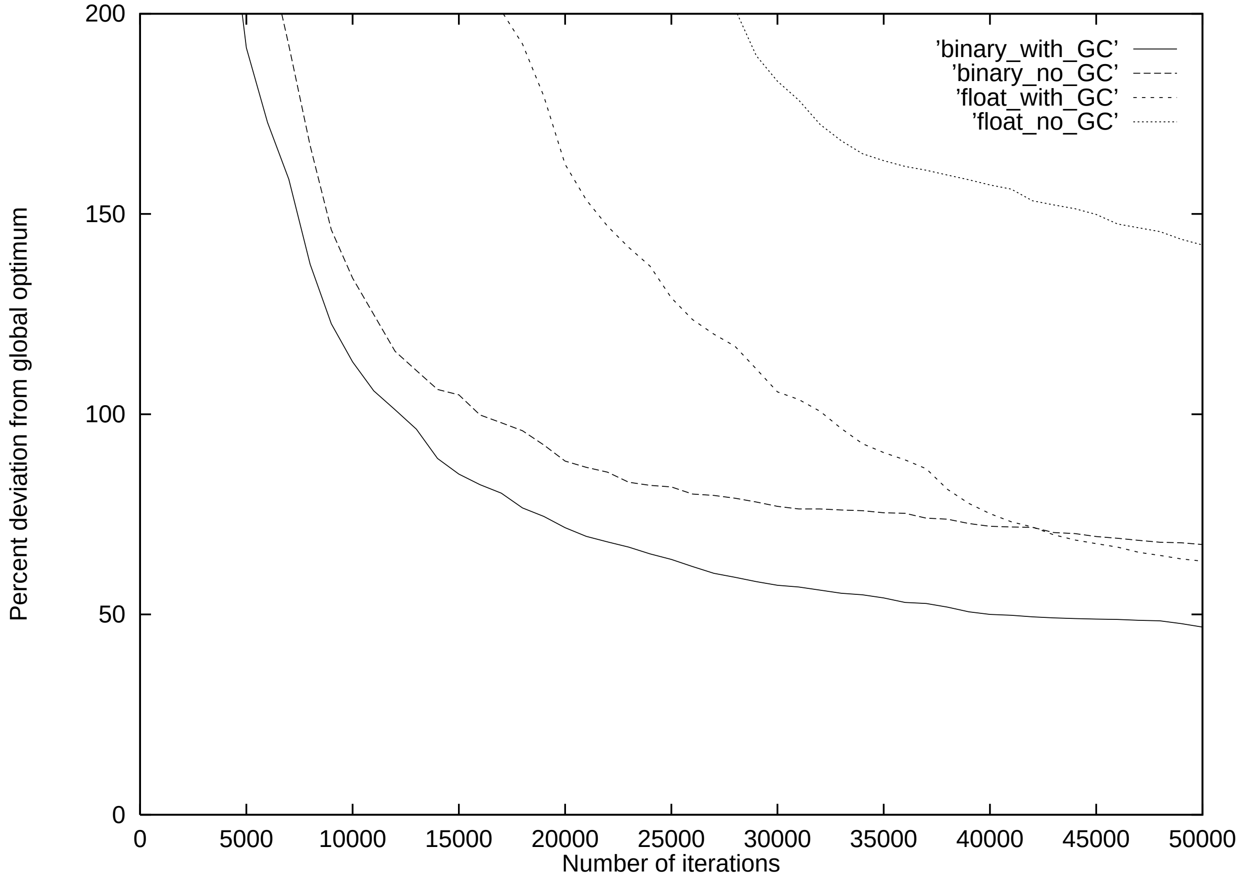

Figure 4: Effect of guided crossover in the Membrane Separation Process Design Domain (no screening or diversity maintenance)

versity Department of Computer Science, 1992. Ph.D. Thesis.

[3] Andrew Gelsey, Mark Schwabacher, and Don Smith. Using modeling knowledge to guide design space search. In *Fourth International Conference on Artificial Intelligence in Design '96*, 1996.

[4] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Mass., 1989.

[5] Jorge J. Moré and Stephen J. Wright. *Optimization Software Guide.* SIAM, Philadelphia, 1993.

[6] D. Powell. Inter-GEN: A hybrid approach to engineering design optimization. Technical report, Rensselaer Polytechnic Institute Department of Computer Science, December 1990. Ph.D. Thesis.

[7] D. Powell and M. Skolnick. Using genetic algorithms in engineering design optimization with non-linear constraints. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 424–431, University of Illinois at Urbana-Champaign, July 1993. Morgan Kaufmann.

[8] Khaled Rasheed and Andrew Gelsey. Adaptation of genetic algorithms for engineering design optimization. In *Fourth International Conference on Artificial Intelligence in Design '96: Evolutionary Systems in Design Workshop*, 1996.

[9] Khaled Rasheed and Haym Hirsh. Using case based learning to improve genetic algorithm based design optimization. In *Submitted under review to ICML'97*, 1997.

[10] Khaled Rasheed, Haym Hirsh, and Andrew Gelsey. A genetic algorithm for continuous design space search. *To appear in: Artificial Intelligence in Engineering.*

[11] E. Sandgren. The utility of nonlinear programming algorithms. Technical report, Purdue University, 1977. Ph.D. Thesis.

[12] Siu Shing Tong, David Powell, and Sanjay Goel. Integration of artificial intelligence and numerical optimization techniques for the design of complex aerospace systems. In *1992 Aerospace Design Conference*, Irvine, CA, February 1992. AIAA-92-1189.

[13] Brian C. Williams and Jonathan Cagan. Activity analysis: the qualitative analysis of stationary points for optimal reasoning. In *Proceedings, 12th National Conference on Artificial Intelligence*, pages 1217–1223, Seattle, Washington, August 1994.