Learning to be Selective in Genetic-Algorithm-Based Design Optimization

Khaled Rasheed   Haym Hirsh

Department of Computer Science
Rutgers, The State University of New Jersey
New Brunswick, NJ 08903, USA
Phone:(732) 445–2001, FAX –5691
krasheed,hirsh@cs.rutgers.edu

Abstract

In this paper we describe a method for improving genetic-algorithm-based optimization using search control. The idea is to utilize the sequence of points explored during a search to guide further exploration. The proposed method is particularly suitable for continuous spaces with expensive evaluation functions, such as arise in engineering design. Empirical results in several engineering design domains demonstrate that the proposed method can significantly improve the efficiency and reliability of the GA optimizer.

Keywords: genetic algorithms, design optimization, machine learning

To appear in Artificial Intelligence in Engineering, Design, Analysis and Manufacturing

1 Introduction

Genetic Algorithms (GAs) [Goldberg 1989] are search algorithms that simulate the process of natural selection. GAs attempt to find a good solution to some problem (e.g., finding the maximum of a function) by randomly generating a collection ("population") of potential solutions ("individuals") to the problem and then manipulating those solutions using genetic operators. Through selection, mutation, and re-combination (crossover) operations, better solutions are hopefully generated out of the current set of potential solutions. This process continues until an acceptable solution is found. GAs have many advantages over other search techniques in complex domains. They tend to avoid being trapped in local sub-optima and can handle different types of optimization variables (discrete, continuous and mixed). A large amount of literature exists about the application of GAs to optimization problems [Michalewicz 1996].

This paper concerns the application of GA optimization in realistic continuous-variable engineering design domains. In such domains a design is represented by a number of continuous design parameters, so that potential solutions are vectors (points) in a multidimensional vector space. Determining the quality ("fitness") of each point usually involves the use of a simulator or some analysis code that computes relevant physical properties of the artifact represented by the vector, and summarizes them into a single measure of merit and, often, information about the status of constraints. For example, the problem may be to design a supersonic aircraft capable of carrying 70 passengers from Chicago to Paris in 3 hours. The goal may be to minimize the takeoff mass of the aircraft. The constraints may include something like "the wings must be strong enough to hold the plane in all expected flight conditions".

Some of the problems faced in the application of GAs (or any optimization technique for that matter) to such problems are:

- Not all points in the space are legitimate designs — some points in the search space ("unevaluable points") cause the simulator to crash, and others ("infeasible points"), although evaluable by the simulator, do not correspond to physically realizable/desirable designs. For example a plane with too thin or too small wings is undesirable even though its takeoff mass may be very small. We have seen domains in which more than 99% of the space is like this.
- The simulator will often take a non-negligible amount of time to evaluate a point. Engineering simulation run time can range often from a fraction of a second to, in some cases, many days.
- The structure of the subspace of good designs may be very difficult to search. For example, when evaluable points form thin slab-like regions that are not parallel to any of the principal axis some crossover operators such as point crossover may not yield satisfactory results [Wright 1990].

This paper presents a modification of GAs specifically intended to improve performance in realistic continuous-variable engineering design domains of this sort. The idea is to accumulate a large enough sample of the points evaluated during the course of the GA optimization. After the search has progressed sufficiently far to have a reasonable degree of confidence that the sample is representative of the search space, a new point is only evaluated if it is not closely surrounded by some of the previously generated points that proved to be bad. Since the time spent in finding the neighbors of a new point is negligible compared to the time it takes to run a simulator, this policy protects the GA from wasting time exploring regions of the space that proved to be fruitless.

We conducted our investigations in the context of GADO [Rasheed 1998b, Rasheed *et al.* 1997], a GA that was designed with the goal of being suitable for use in engineering design. It uses new operators and search control strategies that target the domains that typically arise in such applications. GADO has been applied in a variety of optimization tasks that span many fields. It demonstrated a great deal of robustness and efficiency relative to competing methods.

In GADO, each individual in the GA population represents a parametric description of an artifact, such as an aircraft or a missile. All parameters take on values in known continuous ranges. The fitness of each individual is based on the sum of a proper measure of merit computed by a simulator or some analysis code (such as the takeoff mass of an aircraft), and a penalty function if relevant (such as to impose limits on the permissible size of an aircraft). A steady state GA model is used, in which operators are applied to two parents selected from the elements of the population via some selection scheme, one offspring point is produced, then an existing point in the population is replaced by the newly generated point via some replacement strategy. Here selection was performed by rank because of the wide range of fitness values caused by the use of a penalty function. The replacement strategy used here is a crowding technique, which takes into consideration both the fitness and the proximity of the points in the GA population. The population size is an external parameter with a default value of 10 times the dimension of the search space. The GA stops when either the maximum number of evaluations has been exhausted or the population loses diversity and practically converges to a single point in the search space. Floating point representation is used. Several crossover and mutation operators are used, most of which were designed specifically for the target domain type. GADO also uses a search-control method that saves time by avoiding the full evaluation of points that are unlikely to correspond to good designs. The following section describes the screening module, the focus of this paper, in more detail.

In addition to the screening module discussed in this paper, GADO includes a number of other novel ideas, including:

- **The Diversity Maintenance Module:** The goal of the diversity maintenance module is to ensure that the GA population stays representative of the search space (i.e., the population includes points from many different parts of the search space). This is very important for the success of the optimization and to avoid being trapped in local optima. The idea is to monitor the degree of diversity of the GA population. If at any stage it is discovered that the population elements became very similar to one another, thus losing track of most of the search space, the diversity maintenance module rebuilds the population using previously evaluated points in a way that restores diversity. The diversity maintenance module also rejects proposed points that are extremely similar to previously evaluated points in the belief that they contain redundant information and will promote loss of diversity. The rejection happens before the expensive simulator is used to evaluate these points.

- **Guided Crossover:** This is a new crossover operator which endows the GA with gradient-like capabilities without actually computing any gradients. The idea is to form different search directions by joining pairs of previously evaluated points, rank these directions and take a small step in the best direction. The directions are ranked by the ratio of the difference in objective values between the two end points to the distance between them. This operation can viewed as a very crude way of computing gradients which does not, however, use the expensive simulator to evaluate any new points.

- **Adaptive Penalty:** GADO uses a penalty approach for handling constraints. It uses a novel technique [Rasheed 1998a] for computing penalties which makes sure that the penalty coefficient is neither too large nor too small at any stage of the optimization. The idea is to start with a relatively small penalty coefficient and increase it whenever the search seems to give too little attention to feasibility (if the point with best fitness in the GA population is infeasible). The penalty coefficient may also decrease if the search seems to give too much attention to feasibility (if all the points in the GA population are feasible). This is done to insure proper search of the regions adjacent to constraint boundaries as in many cases the optima lie there.

The remainder of this paper first presents a detailed description of GADO's screening module. We then present a number of experiments concerning the use of the screening module on two realistic engineering design tasks and several engineering design benchmarks. We conclude the paper with a discussion of related efforts and future work.

2 Screening module

In many optimization tasks to date, the functions to optimize were analytical expressions that take a negligible amount of time to compute. In the case of realistic engineering design, however, the optimization function is more often than not a computationally expensive piece of code. We encountered cases in which one evaluation of the optimization function takes CPU hours on a powerful workstation. Consequently, search control became an appropriate tool since it is fast relative to the cost of evaluating the fitness function and in fact — as we will demonstrate in subsequent sections — search control proved to be one of the major contributers to the success of GADO.

Evaluating an individual can be time consuming, and thus it can be beneficial to only select for evaluation points that seem promising. The screening module (SM) decides whether a point is likely to correspond to a good design without invoking any simulator to do this by extrapolating from points evaluated earlier in the search. In particular, the screening module uses a simple K-nearest neighbor approach that maintains a relatively large random sample of the points encountered in the search so far (the sample size is an external parameter with a default value of 30 times the size of the GA population — the size of the sample should in general be selected based on the speed of the simulator and domain knowledge, if available). Before a candidate point generated by crossover and possibly mutation is evaluated, the module finds the K nearest neighbors of the point among the sample (K is an external parameter with a default value of two — in general, the value should increase if the space is suspected to have needle-like optima or a large number of local optima and should decrease if the space is known to be well behaved). If at least one of those nearest neighbors has a fitness that is better than some threshold, the candidate point is evaluated and added to the GA population, otherwise with probability .99 the candidate point is discarded. (We evaluate the point with probability .01 to avoid the theoretical possibility of deadlock if the screening module keeps rejecting every point and to give small regions of good points surrounded by bad points a chance to be discovered.)

A good choice of the threshold is very important for the success of the whole search. The default is to use the fitness of the second worst member of the current GA population as this threshold (and thus it is a function of the current status of the search). In general, a more expensive evaluation function increases the need for the search to be more focused and therefore a higher threshold may be more appropriate. The screening action starts only after 25% of the maximum allowed number of evaluations have been done. This ensures to some extent that the sample is representative of the search space and hopefully gives several of the regions containing good points a chance to be discovered.

3 Experimental results

Our belief was that being more selective in fitness function evaluations through the use of the screening module will improve the overall efficiency of the optimization process. To demonstrate the utility of the screening module we compared GADO with its default setup to a modified version where the screening module was turned off but with all other parameters intact. We then compared the two systems in several domains: two new domains from real tasks in aerodynamic design, plus others from an existing set of engineering design benchmarks [Sandgren 1977].

3.1 Application domain 1: Supersonic transport aircraft design domain

3.1.1 Domain description

Our first domain concerns the conceptual design of supersonic transport aircraft. We summarize it briefly here; it is described in more detail elsewhere [Gelsey *et al.* 1996]. 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 twelve of the aircraft conceptual design parameters in Table 1 over a continuous range of values.

An optimizer evaluates candidate designs using a multidisciplinary simulator. In our current implementation, the optimizer's goal is to minimize the takeoff mass of the aircraft, a measure of merit commonly used in the aircraft industry at the conceptual

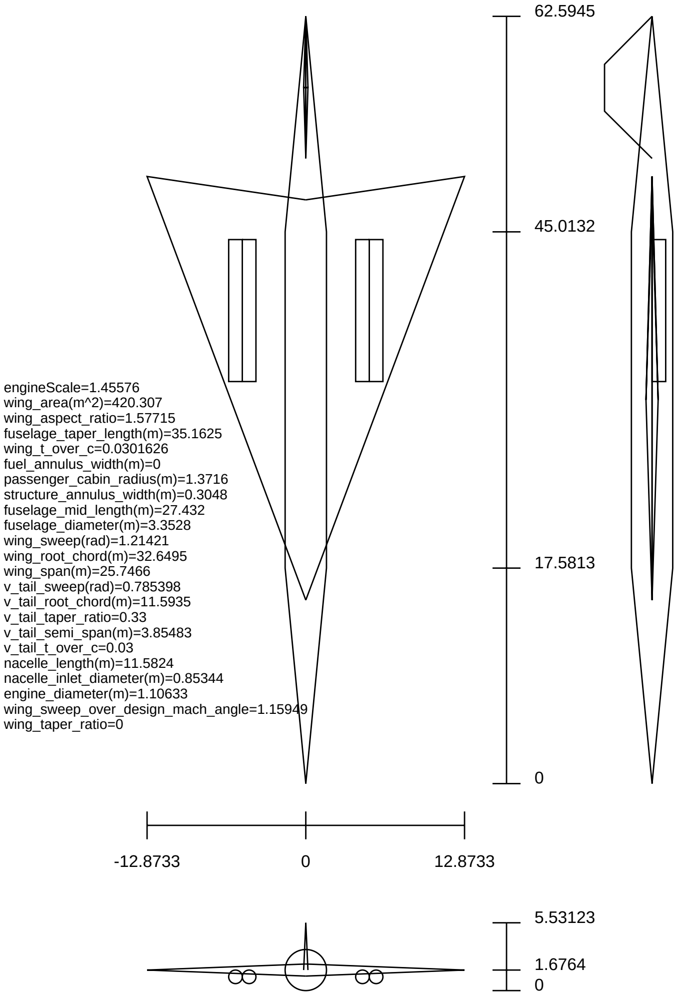

Figure 1: Supersonic transport aircraft designed by our system (dimensions in feet)

Table 1: Aircraft Parameters to Optimize

No.	*Parameter*
1	exhaust nozzle convergent length(l_c)
2	exhaust nozzle divergent length(l_d)
3	exhaust nozzle external length(l_e)
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

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 0.2 CPU seconds 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 aircraft that violate these assumptions and therefore may not be physically realizable even though the simulator does not detect this fact. We refer to these designs as *infeasible points*. For this reason a set of constraints has been introduced to safeguard the optimization process against such violations. We also have the notion of *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.

Figure 2 shows a three dimensional projection of the search space on the first three parameters (l_c, l_d and l_e) of Table 1. The blank regions of the curve are unevaluable points. The figure illustrates the "slab-shaped" evaluable region in this three dimensional subspace. The figure shows how difficult it is for traditional GAs to do optimizations in this domain — for example, the classical point-crossover operation [Goldberg 1989] may well use two evaluable parents and produce unevaluable children.

In summary, the problem has 12 parameters and 37 inequality constraints. 0.6% of the search space is evaluable. No statistics exist regarding the fraction of the search space that is feasible because it is extremely small. A detailed description of the experiments comparing GADO to other optimizers in this domain can be found in [Rasheed 1998b]. Those experiments strongly demonstrated the merit and competitiveness of GADO compared to both stochastic and non-stochastic search methods. Here our focus is on studying the effect of the screening module on performance.

3.1.2 Experiments and results

Figure 3 demonstrates the utility of the screening module in domain 1 (aircraft design). The figure shows the average of 15 runs of GADO both with (labeled "GADO_Aircraft") and without (labeled "GADO_Aircraft_noSM") the screening module. All other parameters were kept at their default values. The figure plots the average (over the 15 runs) of the best measure of merit found so far in the optimization as a function of the number of iterations. (From now on we use the term "iteration" to denote an actual evaluation of the objective function, which is usually a call to a simulator or an analysis code. This is consistent with our goal of understanding how the SM affects the number of calls to the objective function in problems where the SM over-

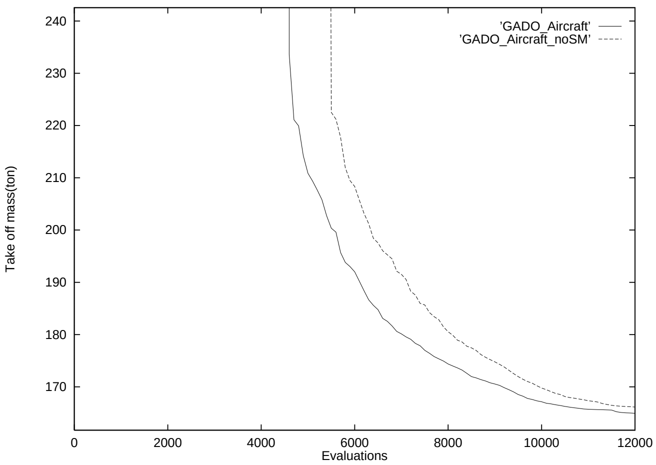

Figure 3: Effect of screening on average performance in application domain 1 (aircraft design)

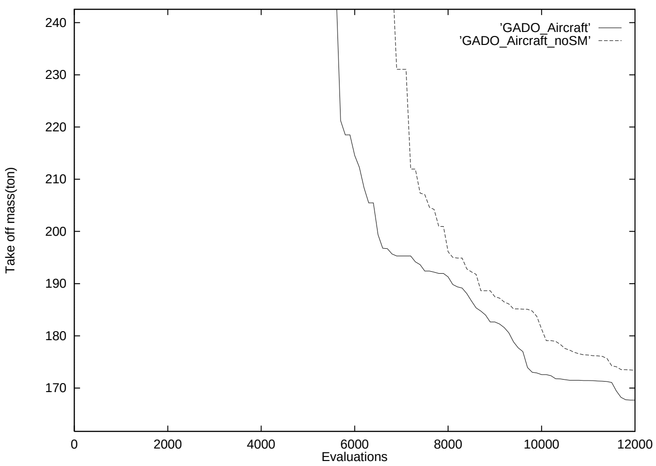

Figure 4: Effect of screening on worst-case performance in application domain 1 (aircraft design)

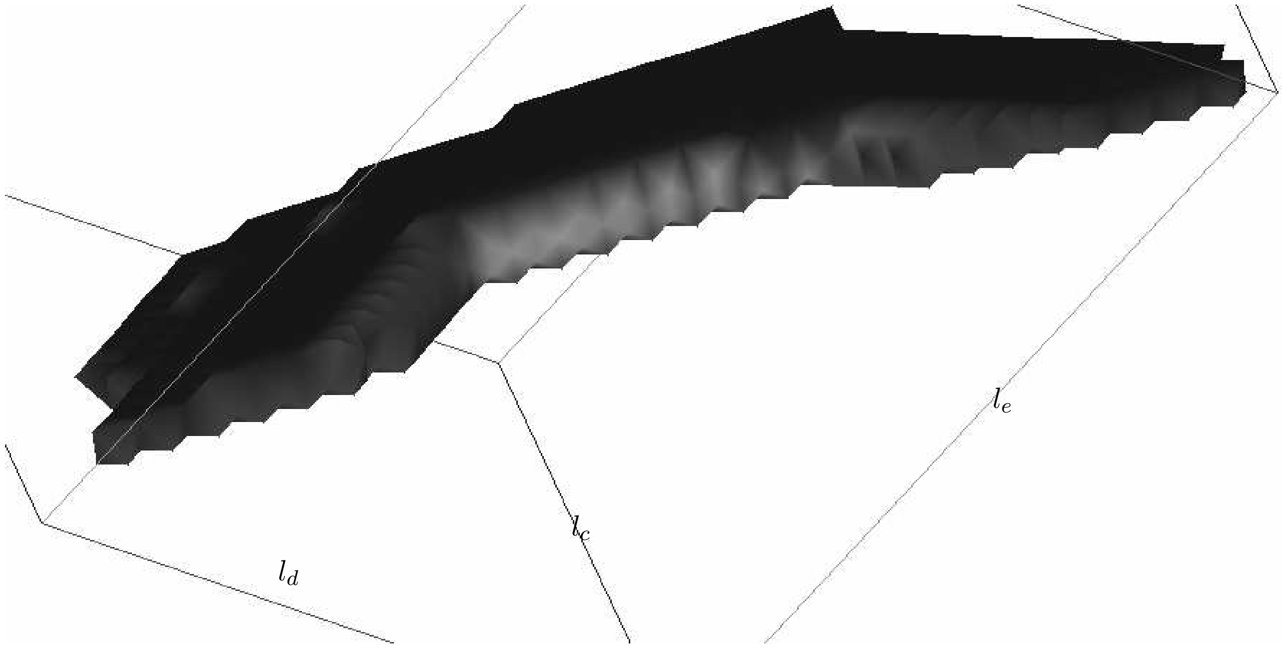

Figure 2: The "slab-shaped" evaluable region.

head is minuscule compared to the runtime of each objective function evaluation, as was the case here. This also helps us avoid the pitfalls of basing evaluations on runtimes, which can vary widely — for example across platforms and even across runs due to variations in memory available and hence caching effects.) The figure shows that the screening module improved the performance in all stages of the search. The feasible region was reached faster with the screening module.[1] Figure 4 shows the effect of the screening module on worst-case performance — very often an engineer will only have time to run the optimizer just once due to the large runtime of the objective function, so an optimizer with bad worst-case performance but excellent average performance may be undesirable. The figure plots the worst (among the 15 runs) of the best measure of merit found so far in the optimization as a function of the number of iterations. The figure is qualitatively the same as Figure 3.

[1]The almost vertical leading parts of the curves are where the feasible region was reached. These parts are so steep because of the large penalty term that goes to zero for feasible points.

3.2 Application domain 2: Supersonic cruise missile inlet domain

3.2.1 Domain description

Our second domain concerns the design of inlets for supersonic and hypersonic missiles. We summarize it briefly here; it is described in more detail in [Zha *et al.* 1996].

The missile inlet designed is an axisymmetric mixed compression inlet that cruises at Mach 4 at 60,000 feet altitude. Minimum manufacture cost for this inlet is critical, and therefore, techniques such as boundary layer bleed and variable geometry are not used — the performance of the inlet thus relies solely on the aerodynamic design of the rigid geometry, such as the extent of external and internal compression, contraction ratio, inlet start throat area, throat location, shock train length, and divergence of subsonic diffuser.

Figure 5 shows the model of the missile geometry which is composed of six fixed parameters and eight design parameters given in Table 2 and Table 3, respectively. The missile inlet is axisymmetric and the coordinates are given in terms of axial (x) and radial (r) positions.

The simulator used in this domain is a program

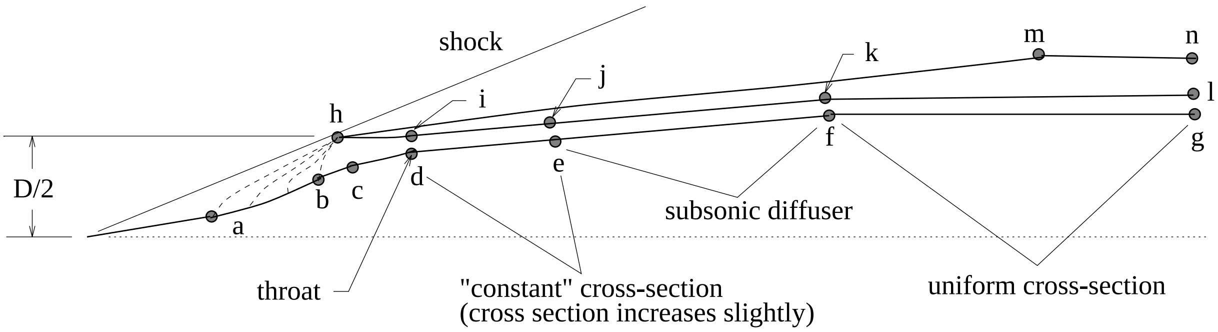

Figure 5: Supersonic missile inlet geometry model

Table 2: Inlet Fixed Parameters

No.	*Parameter*	*Definition*
1	D	cowl diameter
2	r_f	centerbody radius of constant cross section region
3	x_g	length of inlet for computation
4	x_l	length of inlet for computation ($= x_g$)
5	x_n	length of inlet for computation ($= x_g$)
6	r_m	external diameter

Table 3: Inlet Parameters to Optimize

No.	*Parameter*	*Definition*
1	θ_1	initial cone angle
2	θ_2	final cone angle
3	x_d	axial location of throat
4	r_d	radial location of throat
5	x_e	axial location of end of “constant” cross section
6	θ_3	internal cowl lip angle
7	H_{ej}	height at end of constant cross section
8	H_{fk}	height at beginning of constant internal cross section

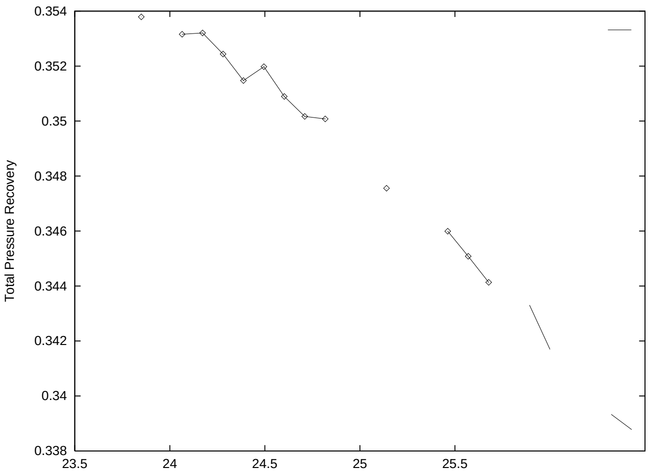

Figure 6: A cross section of the search space defined by NIDA.

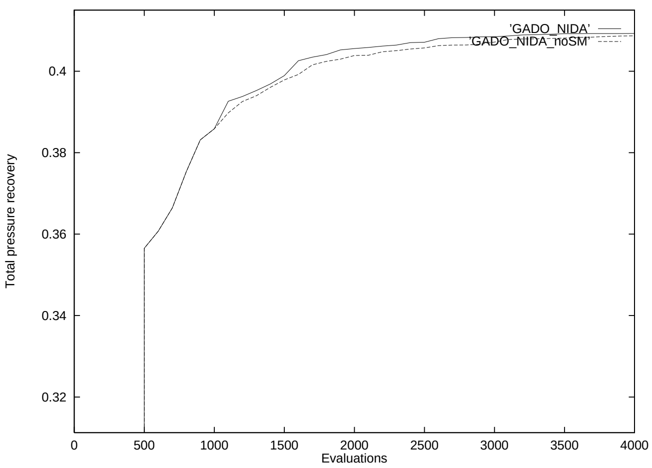

Figure 7: Effect of screening on average performance in application domain 2 (missile design)

called "NIDA" which was developed at United Technology Research Center (UTRC) as an inlet analysis/design tool [Haas *et al.* 1992]. It uses a 1D aerodynamic model with the method of characteristics for the supersonic part upstream of the throat, and empirical correlations based on experimental data downstream of the throat for the region of the terminal shock wave/turbulent boundary layer interaction and sub-sonic diffuser. A complete NIDA run requires about 6 CPU seconds on a DEC Alpha 250 4/266 desktop workstation.

Unfortunately, NIDA suffers from a number of serious shortcomings. There are numerous small discontinuities in the function it computes and in its first derivative, and there are numerous unevaluable points that cause NIDA to crash or print an error message. These discontinuities are sometimes in the middle of regions of good designs. Figure 6 shows a cross section of the search space defined by NIDA, which illustrates these problems. Each point along this curve is for a different value of the axial location of the throat, x_d. The blank regions of the curve are unevaluable points. The figure shows how difficult it can be to do optimizations in this domain.

The eight design parameters (all continuous valued) are given in Table 3, with coordinates given in terms of axial (x) and radial (r) positions. The goal of the optimization is to maximize the total pressure recovery, a quantity that is commonly used to measure the performance of inlets.

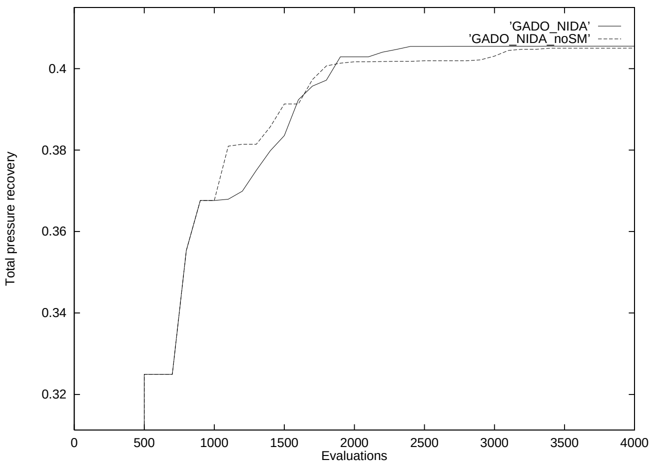

Figure 8: Effect of screening on worst-case performance in application domain 2 (missile design)

In summary, the problem has eight parameters and 20 inequality constraints. 3% of the search space is evaluable and 0.147% is feasible.

3.2.2 Experiments and results

Figure 7 demonstrates the utility of the screening module in application domain 2 (missile design). The figure shows the average of 5 runs of GADO both with (labeled "GADO_NIDA) and without (labeled "GADO_NIDA_noSM") the screening module. All other parameters were kept at their default values[2]. This time it is a maximization problem so higher points on the curve represent better designs. The screening module had very little effect on performance in this domain, neither helping nor harming GADO's behavior. Figure 8 shows a similar lack of effect on worst-case performance. Although the screening module did not help optimization performance in this domain, neither did its use harm the overall optimization effort.

3.3 Benchmark engineering design domains

In order to further define the scope of applicability of the screening module, we examined its performance in a large group of benchmark engineering design domains.

In 1977, Eric Sandgren published his Ph.D. thesis by the title "The utility of nonlinear programming algorithms" [Sandgren 1977]. He applied 35 nonlinear optimization algorithms to 30 engineering design optimization problems and compared their performance.[3] Sandgren's general conclusion was that no single optimization technique among the ones he tested performed reasonably well in all the test cases. He noted, however, that a group of methods (which he called the generalized reduced gradient methods) were reliable across a large number of test cases. The 30 problems he used were also used by other researchers before and after Sandgren's work [Powell and Skolnick 1993].

Those problems have now become used in engineering design optimization domains as benchmarks. One of the most recent experiments involving these domains was reported in [Powell and Skolnick 1993], in which a GA package called OOGA and a numerical optimization package called NumOpt were compared to each other in 10 of Sandgren's domains. The 10 domains were a representative sample of the original 30. We ran experiments in eight of these 10 domains.[4] All eight were minimization problems.

Table 4: Results of optimization in benchmark domains

Domain *No.*	Sandgren *No.*	*Dim.*	**Constraints** *inequ.*	*equ.*	best f
1	13	5	4	0	26.78
2	2	3	2	0	−3.3
3	3	5	6	0	−3.06
4	8	3	2	0	−5.68
5	6	6	0	4	8.92
6	15	16	0	8	244.8
7	21	13	13	0	97.5
8	22	16	19	0	174.7

Some properties of the benchmarks used are summarized in Table 4. The second column of the table shows the problem numbers as they appeared in Sandgren's thesis. The third column shows the problem dimensions (i.e., the number of design variables in each problem). The fourth and fifth columns show the number of inequality and equality constraints respectively. The sixth column shows the best known optima of the problems. A detailed description of these domains is given in [Sandgren 1977]. A detailed description of the experiments comparing GADO to other optimizers in these domains can be found in [Rasheed 1998b]. In this paper we only focus on studying the effect of the screening module on performance.

For each problem GADO was run 5 times using different random starting populations and the default values for its parameters. The limit on the number of function evaluations was set according to the complexity of each problem. The experiments were repeated with the screening module turned off and all other parameters kept the same. We compared the average and worst-case final performance. We found that the screening module never degraded the final average or the worst-case performance in any of these domains. We also found that the effect of the screening module on average performance was moderate in the first four benchmark domains, as can be seen in Figure 9 thru Figure 16, probably because the optimizations were fairly easy. The

[2]The population size was set to 40 rather than its default value of 80 (10 times the dimension) in this domain only, to speedup the experiments.

[3]His study did not include stochastic optimization methods such as genetic algorithms or simulated annealing.

[4]We were unable to do any comparison in 2 of the 10 domains because they had unbounded variables.

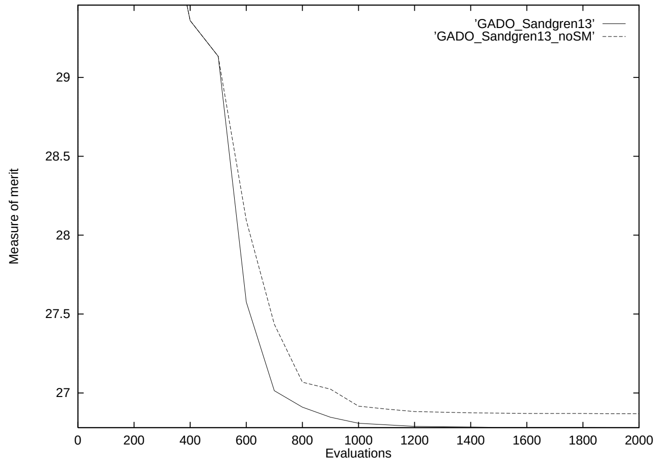

Figure 9: Effect of screening on average performance in benchmark domain 1 (Sandgren's problem 13)

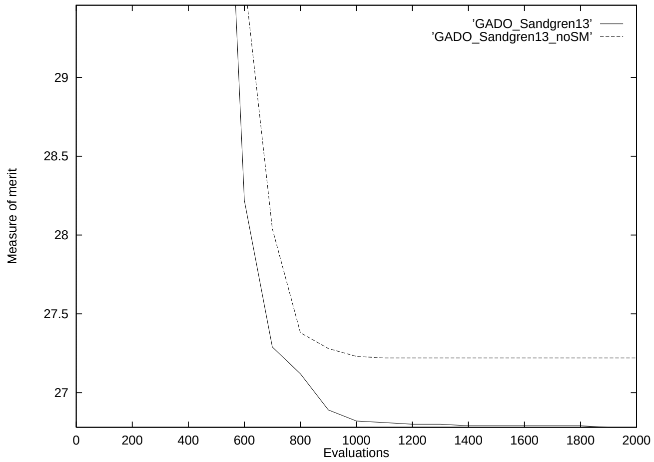

Figure 10: Effect of screening on worst-case performance in benchmark domain 1 (Sandgren's problem 13)

effect of the screening module on worst-case performance was more significant though. A noticeable worst-case performance improvement can be seen in benchmark domains 1, 2, and 4. In benchmark domain 5, the feasible region was found much earlier with the screening module (see Figure 17 and Figure 18), which is expected in a problem with equality constraints – being focused in this case is very important as the feasible region is extremely small.[5] In benchmark domain 6, some GADO optimizations did not find any feasible points in more than 50000 iterations (with or without the screening module) making it difficult to make any qualitative performance comparisons. The last two benchmark domains showed very significant performance improvement due to screening in both average and worst-case performance.

Figure 19 demonstrates the utility of the screening module in benchmark domain 7 (Sandgren's problem 21). The figure shows the average of 5 runs of GADO both with and without the screening module. All other parameters were kept at their default values. The figure shows that the screening module improved the performance in all stages of the search. The feasible region was reached faster with the screening module. The final performance at the end of the search was significantly better with the

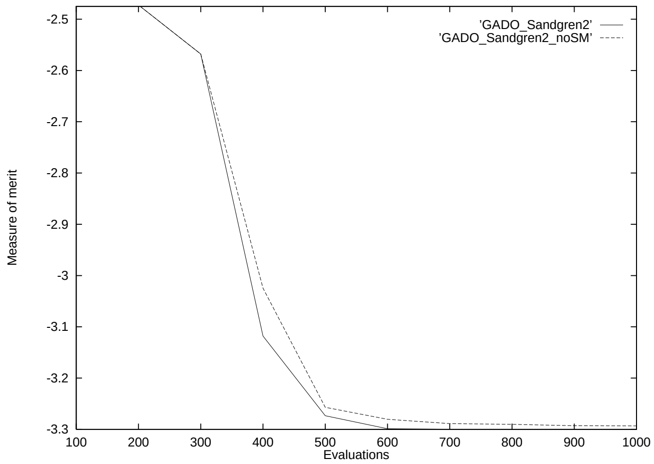

Figure 11: Effect of screening on average performance in benchmark domain 2 (Sandgren's problem 2)

[5] In this research,equality constraints were converted to inequalities by introducing a numerical threshold. Each equality of the form $h_j(\bar{x}) = 0$ was replaced by $|h_j(\bar{x})| \le \epsilon$ where ϵ is a very small constant.

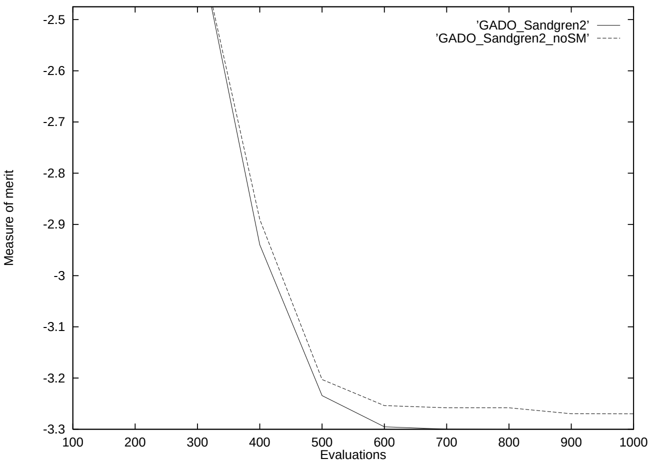

Figure 12: Effect of screening on worst-case performance in benchmark domain 2 (Sandgren's problem 2)

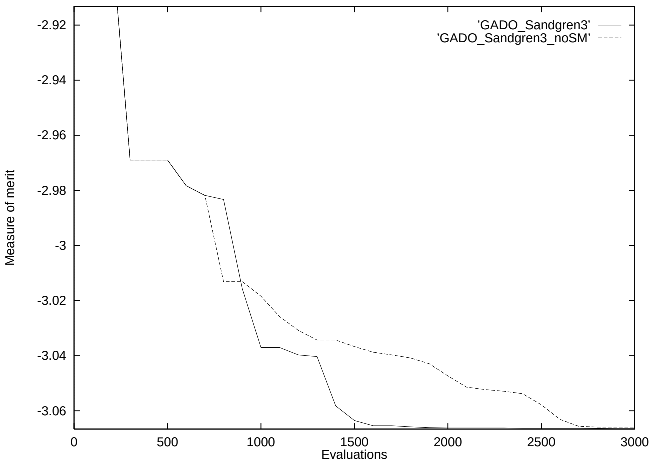

Figure 14: Effect of screening on worst-case performance in benchmark domain 3 (Sandgren's problem 3)

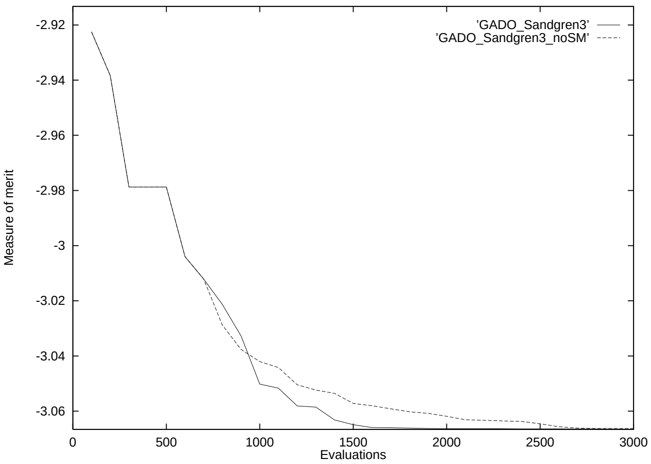

Figure 13: Effect of screening on average performance in benchmark domain 3 (Sandgren's problem 3)

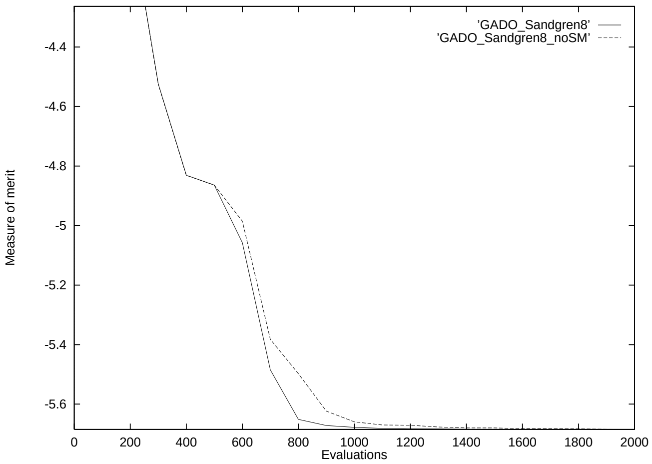

Figure 15: Effect of screening on average performance in benchmark domain 4 (Sandgren's problem 8)

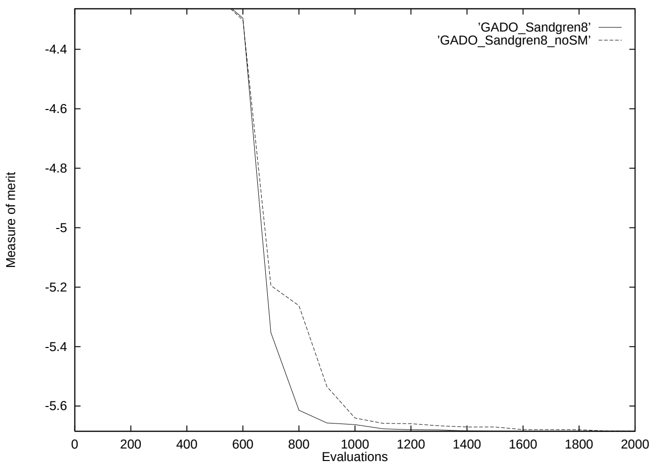

Figure 16: Effect of screening on worst-case performance in benchmark domain 4 (Sandgren's problem 8)

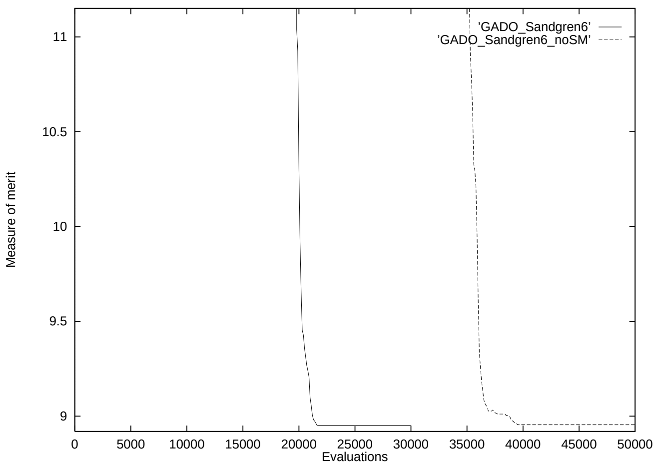

Figure 17: Effect of screening on average performance in benchmark domain 5 (Sandgren's problem 6)

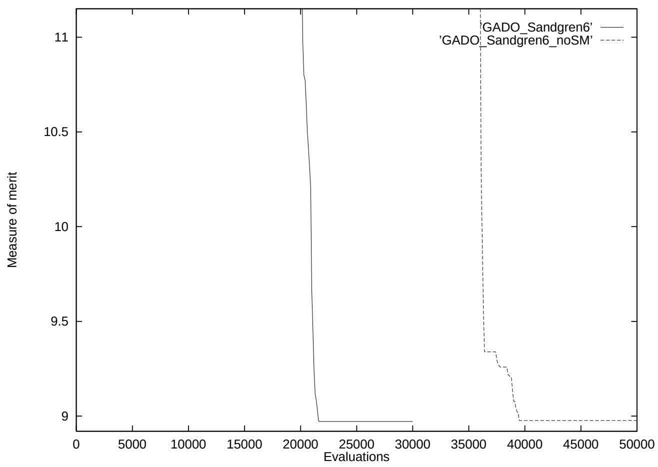

Figure 18: Effect of screening on worst-case performance in benchmark domain 5 (Sandgren's problem 6)

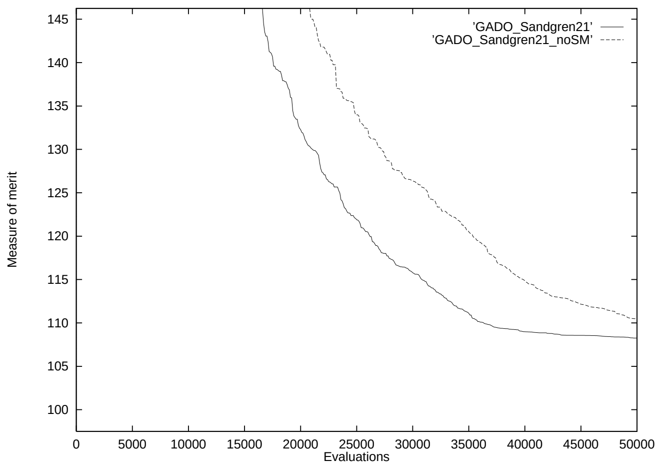

Figure 19: Effect of screening on average performance in benchmark domain 7 (Sandgren's problem 21)

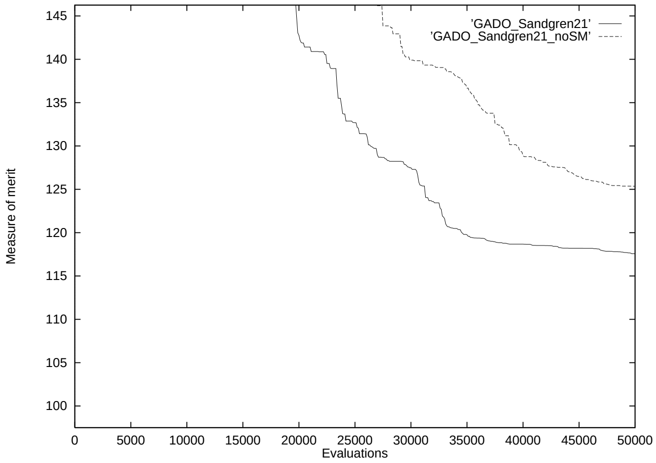

Figure 20: Effect of screening on worst-case performance in benchmark domain 7 (Sandgren's problem 21)

screening module. Figure 20 shows the effect of the screening module on worst-case performance. The figure is qualitatively the same as Figure 19.

Figure 21 demonstrates the utility of the screening module in benchmark domain 8 (Sandgren's problem 22). The figure shows the average of 5 runs of GADO both with and without the screening module. All other parameters were kept at their default values. The figure shows that the screening module improved the performance in all stages of the search. The feasible region was reached faster with the screening module. The final performance at the end of the search was significantly better with the screening module. Figure 22 shows the effect of the screening module on worst-case performance. The figure is qualitatively the same as Figure 21.

4 Discussion

4.1 Pathological search spaces

It is possible to imagine spaces in which the screening module may actually make the global optimum harder to find. Figure 23 illustrates one such situation. The figure shows a one-parameter search space with 2 local optima (assuming it is a maximization problem). Optimum A has a wide basin of attraction while optimum B, the global optimum, has a very

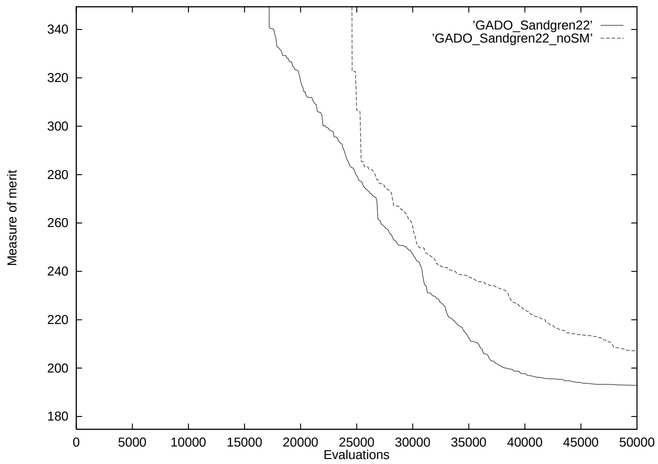

Figure 21: Effect of screening on average performance in benchmark domain 8 (Sandgren's problem 22)

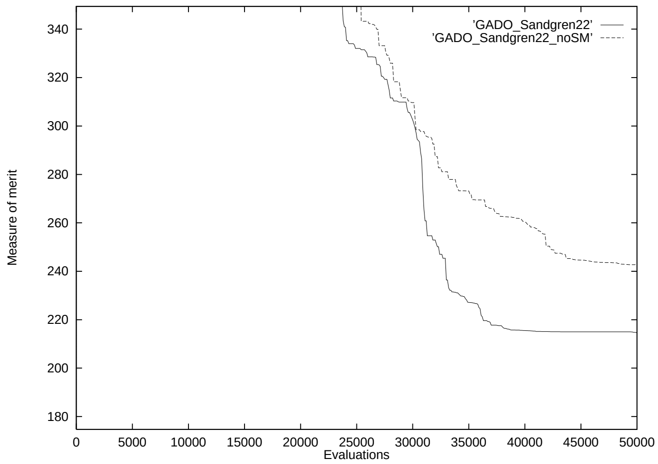

Figure 22: Effect of screening on worst-case performance in benchmark domain 8 (Sandgren's problem 22)

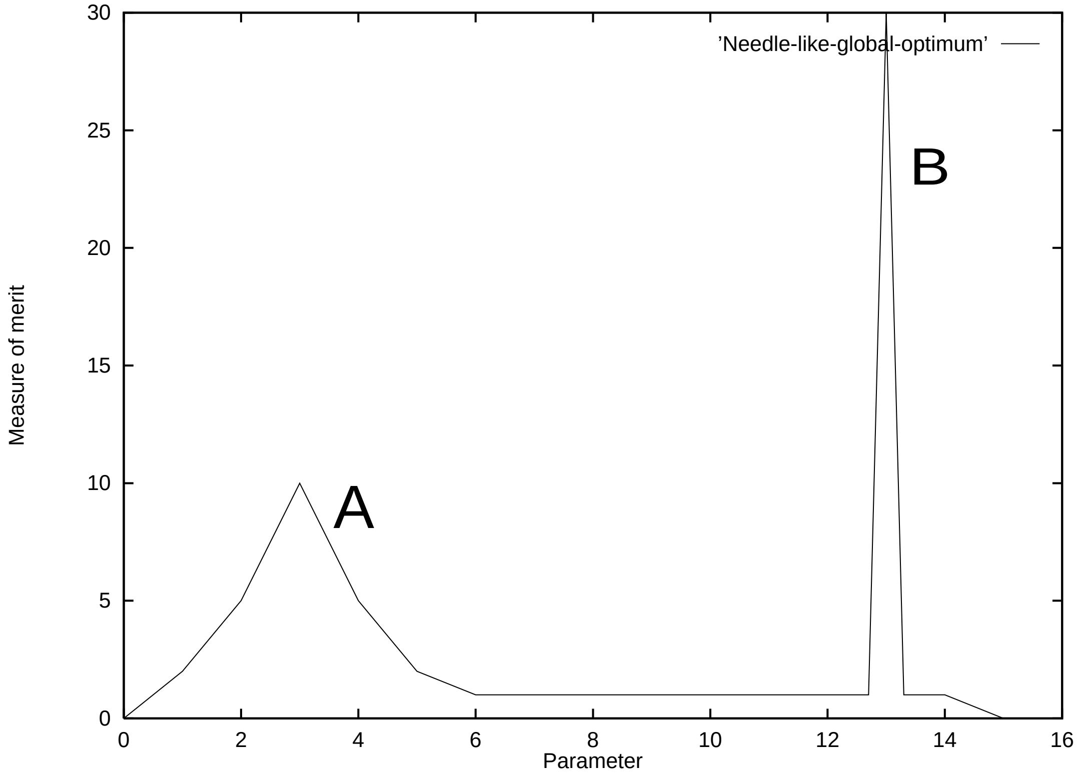

Figure 23: Possibility of screening module failure

narrow basin of attraction.[6] It is therefore possible that all the explorations done before the screening action starts will not encounter a single point in the basin of attraction of optimum B. Thus, the sample used for screening will have only bad points in the vicinity of optimum B and will then prevent the optimizer from evaluating more points in that vicinity. This may result in the optimization finally converging to optimum A. However, in an engineering design optimization problem, optima like optimum B correspond to an unstable design (i.e., a design that will deteriorate considerably due to a slight change in its parameters). In many domains, a design like the one corresponding to optimum A may therefore be considered much better than that corresponding to optimum B from a practical point of view. Finally, we note that if optimum A did not exist (i.e., the curve was flat everywhere except for the basin of attraction of optimum B), the screening module will have no effect on the search behavior until the basin of attraction of optimum B is discovered (if ever). If the basin is discovered, the screening module will then help the search converge faster to the optimum location.

[6]The basin of attraction of an optimum is defined here to be all the points in the search space where if a hill climber is started it will converge to this optimum.

4.2 Effect of the K-nearest neighbor parameter K

We examined the effect of the K-nearest neighbor parameter K in application domain 1 (aircraft design). We compared the average performance of 15 runs of GADO in application domain 1 (aircraft design) with four different values of the parameters K: one, two (the default), three and five. The results are shown in Figure 24. The figure generally demonstrates that the sensitivity to change in this parameter is minimal in this domain. For K=5 (loose screening), we start to see an increase in the time it takes to find the feasible region. Increasing K beyond 5 is likely to be similar to turning off the screening module.

We repeated the experiment in application domain 2 (missile design). The results are shown in Fig-

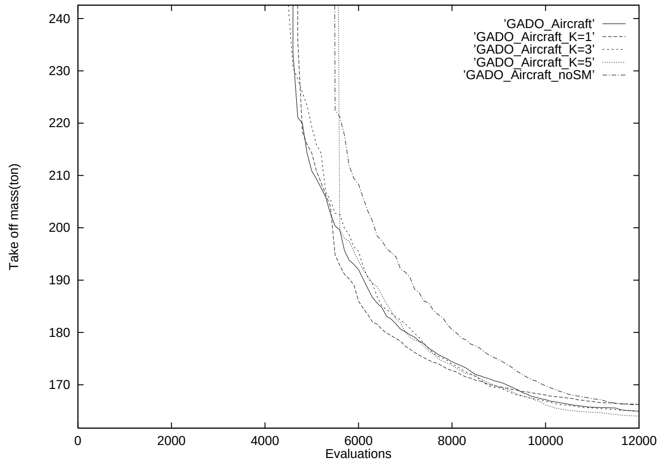

Figure 24: Effect of the K-nearest neighbor parameter K in application domain 1 (aircraft design)

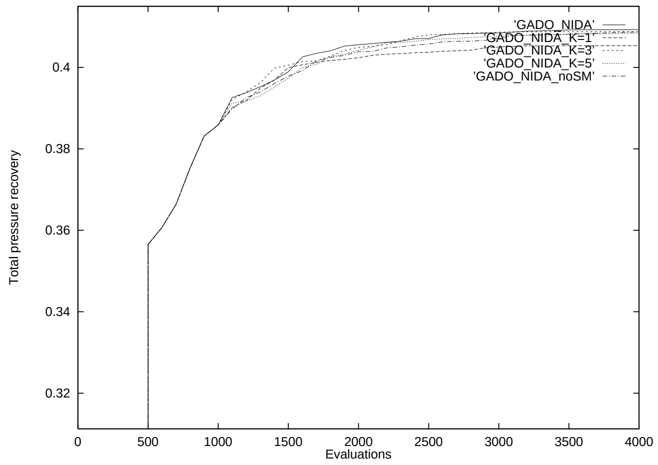

Figure 25: Effect of the K-nearest neighbor parameter K in application domain 2 (missile design)

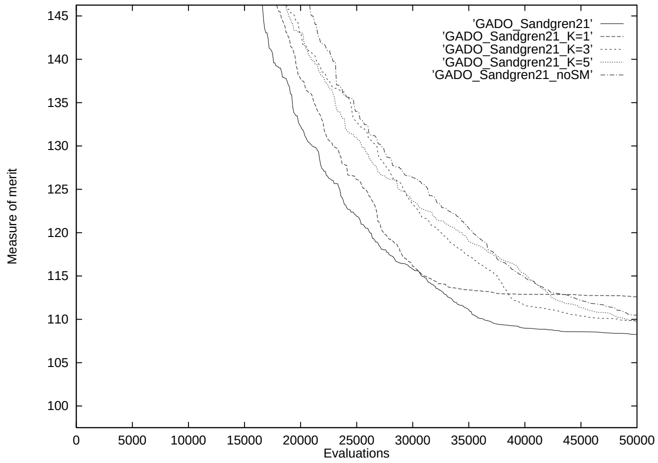

Figure 26: Effect of the K-nearest neighbor parameter K in benchmark domain 7 (Sandgren's problem 21)

ure 25. The figure shows that the sensitivity to the parameter K is minimal in this domain also, except that for K=1, where the final performance was a little worse than with all other values.

We also repeated the experiment in benchmark domain 7 (Sandgren's problem 21). The results are shown in Figure 26. The figure demonstrates a moderate effect of the parameter K on performance. The best final performance was achieved with the default value of 2, the default value set prior to the start of all experiments.

5 Related work

The idea of using the GA optimization history to guide further explorations was studied in [Ravise and Sebag 1996] where inductive learning was used to learn rules describing bad points in a multi-dimension Boolean space. The use of rules was appropriate in that research because the evaluation functions were not expensive enough to warrant the use of case-based learning. The main drawback in that research was that they had to do only negative screening. A point considered bad at any stage of the search will always be considered bad, whereas a point considered good in the beginning of a search may be bad later in the search, as the average quality of the population increases. Using K-nearest neighbor methods implicitly allows the GA to vary the threshold of acceptability over the course of the search, and makes a much weaker assumption about the shape of the boundary between good and bad regions than is the case for other learning methods. Also in the field of GA design optimization [Louis 1997] presented a method for using information about an entire GA optimization to guide *other* GA optimizations in similar domains.

Several research efforts outside the GA field also focused on the idea of using search history to guide future exploration. The examples include:

- Tabu search [Glover 1989, Glover 1990] is a search method similar to simulated annealing in the sense that it is a path following method that may allow movements to an inferior state in an effort to avoid being trapped in local optima. Tabu search uses a list called the "Tabu list" which dynamically changes throughout the course of the search and contains a group of the most recently visited points of the search space. The search avoids going back to these states so as not to keep cycling.

- Dependency-directed backtracking [Rich and Knight 1991] is a search method similar to depth-first search which uses search history to decide which state to backtrack to in case a search path proved fruitless.

- Explanation-based learning (EBL) [Mitchell 1997] takes the outcome of a training process (such as a rule, a proof or a decision tree) and transforms it to a more compact (and often more general) form. [Prieditis and Mostow 1987] proposed an adaptive Prolog interpreter called PROLEARN which reduces the time of executing Prolog queries by using EBL to form generalizations of past proofs that are cached away and used in future problem-solving episodes. Soar [Laird *et al.* 1986] is a general problem-solving architecture that uses EBL to form rules that summarize the problem-solving conducted in a problem space so that the same results can be reproduced in a single step in similar situations, both within a single problem-solving task, as well as on new tasks using that problem space. PRODIGY [Minton 1988] is a domain independent planning system that uses EBL to form search-control rules that help the PRODIGY planner make correct control decisions in situations that are similar to past decisions, both within a single overall task as well as across tasks.

6 Final Remarks

This paper has presented a modification of the GA for continuous design search spaces that is based on maintaining a large sample of previously evaluated points and screening generated points so that only those that are near previously seen good points are actually evaluated. Experimental results demonstrated the merit of the proposed modification in the domains of aircraft design optimization and missile inlet design as well as several benchmark engineering design domains.

The screening module proved to be one of the most important components of GADO. In future work we plan on investigating the possibility of using domain knowledge to dynamically configure the operation of the screening module. For example, the size of the sample used for screening could be set at run time based on the average cost of evaluating each point (i.e., more expensive evaluation functions warrant the use of a larger sample size). The parameter K can also be set in an intelligent way, based on the search space structure that gets discovered at run time. For example, more neighbors can be used to judge the potential of a point in regions where the objective function is observed to change abruptly (such as near the borders of the feasible region).

At a higher level, it is not clear that the K-nearest neighbor approach is the best approach for screening. We plan on investigating the use of a classifier system [Goldberg 1989] which screens potential points by classifying them as promising or unpromising. We also plan to explore the use of more sophisticated machine-learning techniques to extrapolate from past evaluations as part of the screening module. The success of the screening module is dependent on the choice of the distance function for finding neighbors. The Euclidean distance was successful in this research, when the parameters were normalized to have equal ranges. In other domains it may become necessary to use a more complicated distance function.

Acknowledgments

We thank our aircraft design expert, Gene Bouchard of Lockheed, for his invaluable assistance in this research. We thank Marty Haas from United Technologies for his assistance in the missile design part of 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 Defense Advanced Research Projects Agency of the Department of Defense through contract DARPA-DABT 63-93-C-0064.

References

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

[Glover 1989] F. Glover. Tabu Search-Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[Glover 1990] F. Glover. Tabu Search-Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

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

[Haas *et al.* 1992] M. Haas, R. Elmquist, and D. Sobel. NAWC Inlet Design and Analysis (NIDA) Code, Final Report. UTRC Report R92-970037-1, 1992.

[Laird *et al.* 1986] J. E. Laird, P. S. Rosenbloom, and A. Newell. Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, 1(11), 1986.

[Louis 1997] Sushil J. Louis. Working from blueprints: Evolutionary learning for design. *Artificial Intelligence in Engineering*, 11(3):335–341, 1997.

[Michalewicz 1996] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, New York, 1996.

[Minton 1988] S. Minton. *Learning search control knowledge: An explanation-based approach*. Kluwer Academic Publishers, Boston, MA, 1988.

[Mitchell 1997] Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[Powell and Skolnick 1993] 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. Morgan Kaufmann, July 1993.

[Prieditis and Mostow 1987] A. E. Prieditis and J. Mostow. Prolearn: Towards a Prolog interpreter that learns. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, WA, 1987. Morgan Kaufmann.

[Rasheed *et al.* 1997] Khaled Rasheed, Haym Hirsh, and Andrew Gelsey. A genetic algorithm for continuous design space search. *Artificial Intelligence in Engineering*, 11(3):295–305, 1997. Elsevier Science Ltd.

[Rasheed 1998a] Khaled Rasheed. An adaptive penalty approach for constrained genetic-algorithm optimization. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 584–590, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[Rasheed 1998b] Khaled Rasheed. GADO: A genetic algorithm for continuous design optimization. Technical Report DCS-TR-352, Department of Computer Science, Rutgers University, New Brunswick, NJ, January 1998. Ph.D. Thesis, http://www.cs.rutgers.edu/~krasheed/thesis.ps.

[Ravise and Sebag 1996] Caroline Ravise and Michele Sebag. An advanced evolution should not repeat its past errors. In *Thirteenth International Conference on Machine Learning*, 1996.

[Rich and Knight 1991] Elaine Rich and Kevin Knight. *Artificial Intelligence.* McGraw-Hill, 1991.

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

[Wright 1990] Alden Wright. Genetic algorithms for real parameter optimization. In *The First workshop on the Foundations of Genetic Algorithms and Classifier Systems*, pages 205–218, Indiana University, Bloomington, July 1990. Morgan Kaufmann.

[Zha *et al.* 1996] G.-C. Zha, Don Smith, Mark Schwabacher, Khaled Rasheed, Andrew Gelsey, and Doyle Knight. High performance supersonic missile inlet design using automated optimization. In *AIAA Symposium on Multidisciplinary Analysis and Optimization '96*, 1996.