GADO: A GENETIC ALGORITHM FOR CONTINUOUS DESIGN OPTIMIZATION

BY KHALED MOHAMED RASHEED

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Haym Hirsh

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

January, 1998

ABSTRACT OF THE DISSERTATION

GADO: A Genetic Algorithm for Continuous Design Optimization

by Khaled Mohamed Rasheed

Dissertation Director: Haym Hirsh

Genetic algorithms (GAs) have been extensively used as a means for performing global optimization in a simple yet reliable manner. However, in some realistic engineering design optimization domains a general purpose GA is often inefficient and unable to reach the global optimum. In this thesis we describe a GA for continuous design-space optimization that uses new GA operators and strategies tailored to the structure and properties of engineering design domains. Empirical results in several realistic engineering design domains as well as benchmark design domains demonstrate that using our system can greatly decrease the cost of design space search, and can also improve the quality of the resulting designs.

Acknowledgements

I thank all members of the Rutgers HPCD (Hypercomputing and Design) project. In particular, I thank my advisors, Haym Hirsh and Andrew Gelsey, for their continued support and guidance over the years. I thank Donald Smith for being on my thesis committee, and for his thoughtful comments and critiques. I thank Doyle Knight for being on my thesis committee, and for his help with the inlet domain. I thank Don Smith and Keith Miyake for their help with the aircraft simulator. I thank Saul Amarel and Lou Steinberg for their leadership of the HPCD project. I would also like to thank Gene Bouchard of Lockheed Martin for his valuable help.

I would like to thank my parents and my wife Luma for their invaluable support and help, even though I could never thank them enough.

This research was partially supported by the Advanced Research Projects Agency of the Department of Defense (ARPA) and the National Aeronautics and Space Administration (NASA) under the following grants: NASA grant NAG2-817 and ARPA contract DABT 63-93-C-0064.

Publication notes. Description of some of the early stages of the research presented in this thesis has appeared elsewhere with various co-authors, all of whom I thank. Some of the work in Chapter 3 was described in the paper "A Genetic Algorithm for Continuous Design Space Search" which appeared in *Artificial Intelligence in Engineering*, with Haym Hirsh and Andrew Gelsey as co-authors [Rasheed *et al.* 1997]. Some of the work in Chapter 4 was described in the paper "Using Case-Based Learning to Improve Genetic-Algorithm-Based Design Optimization" which can be found in the proceedings of the *Seventh International Conference on Genetic Algorithms*, with Haym Hirsh as co-author [Rasheed and Hirsh 1997b]. Finally, the work in Appendix C is further described in the paper "Automated Optimal Design of Two dimensional High Speed Missile Inlets" which will appear in the *36th AIAA Aerospace Sciences Meeting and Exhibit*, with Michael Blaize and Doyle Knight as co-authors [Blaize *et al.* 1998].

Table of Contents

Chapter 1

Introduction

1.1 The engineering design optimization problem

In a very general form, the engineering design optimization problem can be stated as follows: given a computer tool that can evaluate a design, the goal is to use this tool to come up with the best design according to some measure of merit and subject to some constraints, on condition that this is done within time limits.

The tool is usually a simulator or a piece of analysis code. The measure of merit may be a function of manufacturing cost, quality, stability or any combination of these and/or similar properties. The "thing" to be designed may be a machine or a process. The constraints may be performance related or modeling related.

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". The time limits for the design optimization may be a few hours (as in a quick feasibility study) or a few months (in the case of final product design).

In its most general form, the design problem is extremely complicated. The word "design" may be taken to mean any kind of decision making regarding the shape or composition of some artifact. More formally, it could mean structural design or parametric design or both. Structural design involves making high level decisions about the overall shape of the artifact (for example how many wings the aircraft should have). Most variables involved in structural design are discrete. Parametric design involves making more detailed decisions about numerical aspects of the design (for example what

the length of the aircraft should be). Most variables involved in parametric design are continuous.

In this thesis we limit our view of the design process to the parametric design phase. In this phase the structure of the thing to be designed is already well known, subject to assigning values to a finite set of parameters. We also limit our attention to the case of continuous parameters because this is the type of parameters that usually remain to be set once the structure is defined.

Once we make these assumptions, the problem becomes a general constrained non-linear programming problem in which the objective function and (often) the constraints are the outcome of a program. In other words, the problem may be stated as:

$minimize \quad f(\bar{x})$

$subject \ \ to:$

$g_i(\bar{x}) \leq 0 \quad i = 1, ..., l$

$h_j(\bar{x}) = 0 \quad j = 1, ..., m$

$\bar{x}^{(L)} \leq \bar{x} \leq \bar{x}^{(U)} \quad j = 1, ..., m$

where

- $\bar{x}$ is a vector of real numbers representing the parametric description of the object being designed. The vectors $\bar{x}^{(L)}$ and $\bar{x}^{(U)}$ represent the lower and upper bounds of the design parameters respectively.

- $f(\bar{x})$ is called the objective function. It represents a numerical property of the object being designed, which needs to be minimized (such as production cost).

- $g_i(\bar{x})$ and $h_j(\bar{x})$ are called the inequality constraints and the equality constraints respectively.[1] These constraints are a means of quality control. Some of these constraints ensure that the design is physically realizable, others ensure adequate performance and the rest ensure that the design is within the limits of the model being used.

[1] In practice, equality constraints are usually converted to inequalities by introducing a numerical threshold. For example, $h_j(\bar{x}) = 0$ may be replaced by $|h_j(\bar{x})| \leq \epsilon$ where ϵ is a small constant

Constraints are usually handled in one of two ways:

- *Direct methods* use the gradients (usually their numerical approximations) of the constraints to compute a direction that satisfies all constraints. Once a feasible point is found (i.e. one that satisfies all the constraints), direct methods usually use the gradient of the objective function and the active constraints to compute a direction that improves the objective function while not violating any constraints.

- *Penalty methods* do not work directly on the constraints but instead they add a penalty term to the objective function to account for constraint violations if any and then unconstrained optimization is performed using the augmented objective function. The penalty term is usually the product of a (large) positive penalty coefficient times the sum of the constraint violations.

The problem of proving that a point is a local optimum of a general nonlinear programming problem is undecidable [Schwabacher 1996]. Therefore there is no analytical solution to the above problem. On the other hand there is a large amount of literature on how to solve the nonlinear programming problem in practice. Most such techniques do not take into consideration the special properties of engineering design optimization spaces. As a consequence, their performance is not acceptable in such domains, as will be demonstrated in the remainder of this thesis.

1.2 Difficulty of optimization in engineering design domains

Some of the problems faced in the application of optimization techniques to continuous parameter engineering design optimization problems are:

- **Unevaluable and infeasible points/regions:** 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 designs. We have seen domains in which more than 99% of the space is like this.

- **Expensive evaluations:** The simulator will often take a non-negligible amount of time to evaluate a point. The simulation time ranges from a fraction of a second to, in some cases, many days.

- **Badly structured spaces:** The structure of the space 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, it may become very difficult to do optimization.

- **Multiple local optima:** The search space may well have a large number of local optima. These local optima will certainly trap gradient based local optimizers and may also trap global optimizers such as GAs. The multiple local optima may be:

  True local optima: There can be multiple physical local optima.

  Apparent local optima: These are caused by numerical noise. Noise can have many reasons, such as roundoff errors, approximations in the model and gridding.[2] It can also result from using inexact equation solvers and numerical integration.

- **Non-smoothness:** This often results from table look-up in empirical tables or numerical problems. The non-smoothness can take the form of discontinuity in objective function or in first derivative.

1.3 Current methods

In this section we present a brief survey of the existing design optimization methods. Our goal is to demonstrate the inadequacy of these methods and the need for better methods.

[2]Gridding is the process of partitioning a surface or a volume into small regions in order to apply some methods of analysis (such as finite element analysis).

1.3.1 Manual design

In manual design the design engineer uses his intuition and experience, as well as some computer tools to do the design. The computer tools used are mostly computer aided design (CAD) tools and numerical simulators. The CAD tools are used to visualize and modify the design. The numerical simulators are used to evaluate the design, providing information about its merit and limitations. The process usually works as follows:

1. **Initial design:** The designer obtains an initial design using intuition and experience or by referring to similar previous designs by himself or his company.

2. **Design-evaluate-redesign loop:** The designer then repeatedly does the following:

   - Modify the design to make it more suitable for the problem at hand. The CAD tools are usually used at this stage.
   - Evaluate the design using the numerical simulator.
   - Decide on how to modify the design to further achieve the design goals through the interpretation of the information returned by the simulator. This is the most critical step in this loop. Success in this step depends on the experience, intelligence and creativity of the designer.

   This Design-evaluate-redesign loop is usually repeated until either:

   - An adequate design is found.
   - The designer concludes that the specifications and/or goals could never be satisfied. In this case the specifications may need to be changed and the whole design process is repeated.
   - The designer runs out of time.

The manual design process is slow, mainly because a human designer is in the loop. Humans tend to use satisficing rather than optimization in design. They stop when they find an adequate design (for example, a design that beats the competition) rather

than an optimal design. The use of computer resources is usually inefficient. The quality of the final design as well as the time it takes to find it depend heavily on the experience of the human designer. This dependency is detrimental to innovation. The designer usually has some intuition and qualitative rules that bias his decisions. He will probably try to imitate the good designs that worked in the past. It is possible that this will lead him to explore good regions of the design space that are not necessarily optimal. It is also unlikely for one designer to be able to make decisions about all the design aspects. Therefore, the design problem is usually decomposed to multiple sub-problems, possibly handled by different designers or even different groups. If this decomposition does not correspond to a natural decomposition in the design space, the overall design may not be optimal.

Despite these deficiencies, it is unfortunate that most engineering design is still being done manually. The design-evaluate-redesign loop described above seems like it could be automated and can benefit from optimization. Some engineers have tried to use automated optimization, but were unhappy with the results; others have heard about these failures, and have therefore decided not to use automated optimization [Knight 1994]. We believe that the inadequacy of most existing "off-the-shelf" optimization techniques (as will be demonstrated in the rest of this thesis) is one of the main reasons for this.

1.3.2 Existing optimization methods

Conventional numerical optimization methods

Conventional numerical optimization methods (mostly gradient-based) have been applied with limited success to some engineering design optimization problems. They usually work well in problems with low dimension and smooth search spaces. In order for these methods to succeed, a relatively good starting point must be used. The improvement over the starting point is usually minimal as the optimizer is likely to move to the nearest local optimum or stop if it encounters a numerical hurdle.

Sandgren [Sandgren 1977] applied 35 nonlinear optimization algorithms to 30 engineering design optimization problems and compared the performance. 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 the generalized reduced gradient methods were reliable across a large number of test cases (we will discuss his work further in Chapter 4). Several research efforts used conventional numerical optimization methods for aircraft design [Vanderplaats 1984, Sobieszczanski-Sobieski *et al.* 1985, Bramlette *et al.* 1990, Kroo *et al.* 1994, Sobieszczanski-Sobieski and Haftka 1996]. [Bramlette *et al.* 1989] compared the application of different methods including gradient based numerical optimization to the design and manufacture of aeronautical systems. All of these works describe the application of "off-the-shelf" techniques to design optimization problems with no serious effort to adapt them to the particular properties of these problem domains.

Stochastic optimization methods

Genetic algorithms

Genetic Algorithms (GAs) [Goldberg 1989] are search algorithms that simulate the process of natural selection and survival of the fittest [Holland 1975]. GAs attempt to find a good solution to some problem (e.g., finding the maximum of a function) by randomly generating a collection of potential solutions to the problem and then manipulating those solutions using what is called genetic operators. Genetic operators use existing solutions to produce new solutions. Each solution is assigned a fitness value which is a numerical assessment of how well it solves the problem. The key idea is to select for reproduction the solutions with higher fitness and apply the genetic operators to these to generate new individuals. Through mutation and re-combination 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).

The success of a GA depends on its ability to perform a balanced amount of exploration of the space as well as exploitation of the promising regions. A good GA usually favors exploration in the beginning of the search and gradually shifts attention to exploitation. A more detailed description of GAs is provided in Chapter 2.

Several research efforts have applied traditional genetic algorithms to engineering design optimization problems in a variety of domains, including control system design [Kundu and Kawata 1996], architectural and civil engineering design [Gero *et al.* 1997, Rosenman 1997], VLSI design [Lienig and Thulasiraman 1993], mechanical design [Chapman and Jakiela 1996] and aircraft design [Obayashi *et al.* 1997].

Deb [Deb and Goyal 1997, Deb 1997] developed a GA called GeneAS for engineering design optimization with mixed variables (both discrete and continuous). He demonstrated the merit of his GA in the domain of mechanical component design. Deb also used a binary-coded GA to design car suspension for comfort [Deb and Saxena 1997]. Deb's research represents a significant effort to adapt the GA to the engineering design optimization problem. Deb did not, however, address the problem of unevaluable points.

Simulated annealing

Simulated Annealing methods (SAs) [Kirkpatrick *et al.* 1983] are based on an analogy with thermodynamics, specifically the way in which metals cool and anneal. SAs attempt to solve a problem by starting from an initial point in the solution space and then moving from one point to another until they hopefully reach an optimum. At every iteration the SA generates a potential new point (using the current point). The SA then moves to the new point (making it the current point) with probability P where:

- $P = 1$ if the new point is better than the current point.
- $0 \leq P \leq 1$ if the new point is not better than the current point.

The probability P with which the SA moves to a worse point is large in the initial stages of the search and decreases as the search progresses. SAs are similar to GAs in

their tendency to avoid being trapped in local optima. The main differences between SAs and GAs are:

- SAs are path following methods (i.e. they move from one point to another in the search space over the course of the optimization). At any iteration, the history of the entire optimization is summarized in a single point (the current point). GAs, on the other hand, maintain a much more thorough representation of the search space through a whole population of points that collaborate with each other to find better regions.

- SAs have the ability to quickly switch their attention from one region of the search space to another. GAs on the other hand have a high inertia and take longer to switch attention.

Simulated Annealing has been used in several engineering design domains including communication network design [Chardaire and Lutton 1993, Andersen *et al.* 1993] and mechanical design [Jain and Agogino 1988, Malhotra *et al.* 1991, Jain *et al.* 1992, Cagan and Kurfess 1992].

There has been a long philosophical argument about which is better: SAs or GAs. We believe that this is problem dependent. In the case of engineering design optimization, however, GAs are more likely to win because of the fact that they maintain a whole population of points rather than one. The collaboration between the many points of the GA population makes it easier to find small evaluable/feasible regions. The GA also takes advantage of the decomposability of many engineering design domains through the crossover operation (as will be discussed further in the following chapter).

Hybrid methods

Powell [Powell 1990, Tong *et al.* 1992] has built a module called Inter-GEN, part of the ENGINEOUS system [Tong 1988]. It contains a genetic algorithm and a numerical optimizer, and uses a rule-based expert system to decide when to switch between the two. Powell tested his system on a realistic design task (jet engine design). He does not, however, address the issue of unevaluable points. He also assumed that injecting

human knowledge in the form of rules improves performance, ignoring the potentially detrimental effect this has on innovation. Combining GAs and knowledge-based systems was also done in [Rogers *et al.* 1996].

Gage [Gage 1994, Gage *et al.* 1995] has also combined genetic algorithms with gradient-based optimization. He combined GA's with SQP[3] in two domains. The first domain was aircraft wing design. He used a GA to search a space of wing configurations that is described using a grammar, and then used SQP to optimize the size of the wings. The second domain was truss design. He used the GA to search a space of truss configurations that is described using a grammar, while using SQP at each iteration of the GA to optimize the size of the members. Using the GA to search a configuration space before using SQP to optimize the sizes in a particular configuration seems to be a plausible way to save time (as opposed to direct optimization of the combined space). However, this approach is more vulnerable to being trapped in local sub-optima than a global method which optimizes configuration and parameters simultaneously. Like Powell, Gage does not directly address the issue of unevaluable points. Combining GAs and local search methods was also done in [Land *et al.* 1997].

Schwabacher [Schwabacher 1996] proposed various methods to improve numerical optimization in engineering design through the use of artificial intelligence and machine learning techniques. He used a state of the art gradient-based SQP method called CFSQP[4] and augmented it with AI techniques to over come some of its limitations when used in engineering design. He successfully applied his methods to several realistic engineering design domains, including aircraft design and missile inlet design, the two principal domains used in this thesis. He addressed several of the problems specific to engineering design domains described in Section 1.2. For example, he used rules to compute gradients numerically in the presence of unevaluable points. His success was limited, however, because his base method (CFSQP) was very sensitive to problems like non-smoothness and discontinuities in the design space. Despite Schwabacher's efforts,

[3]SQP is an acronym for Sequential Quadratic Programming, a quasi-Newton method that solves a nonlinear constrained optimization problem by fitting a sequence of quadratic programs to it.

[4]CFSQP will be described in detail in Chapter 3.

his method performed poorly in some of the domains he explored (the missile inlet design domain for example). We used his work, among other techniques, for comparison with our new method in Chapter 3. Work on augmenting numerical optimization techniques with AI methods also includes the use of expert systems [Bouchard *et al.* 1988, Orelup *et al.* 1988] and machine learning [Hoeltzel and Chieng 1987].

Finally, Ogot [Ogot and Alag 1995] combined Simulated Annealing and heuristic search to create an algorithm called MAH.[5] He demonstrated the effectiveness of MAH via three problems in kinematic synthesis.

Except for Schwabacher's work, none of these hybrid methods is focused directly on the problems of the search spaces addressed in this thesis.

1.4 The proposed approach

We present GADO,[6] a GA specifically designed to be used in design optimization domains. It combines existing techniques from the GA literature (and the optimization literature in general) that are proper for these domains with novel ideas. It is a highly adaptive GA that focuses on exploration in the beginning of the search and shifts attention to exploitation towards the end.

GADO is described in detail in Chapter 2. It includes many novel ideas, some of which are:

- **The Screening Module:** The screening module was motivated by the fact that the evaluation of the objective function entails a simulation run that takes a relatively long time. The idea of the screening module is to predict the merit of a proposed design *before* the simulator is used to actually evaluate it. To do this the screening module accumulates a large sample of the designs evaluated so far and uses this sample to predict the merit of a proposed design. The nearest neighbors of the proposed design are found and if all of them are bad the proposed design is rejected. Otherwise, the proposed design is fully evaluated using the simulator.

[5] MAH is an acronym for Mixed Annealing/Heuristic algorithm.

[6] GADO is an acronym for Genetic Algorithm for Design Optimization.

- **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.

- **Dynamic Penalty:** GADO uses a penalty approach for handling constraints. It uses a novel technique 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 highest 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 may lie there.

We claim that engineering design optimization in domains that have the features described in Section 1.2 can be done more reliably and efficiently using GADO than using other optimization techniques. By more reliably we mean GADO is most likely to produce acceptable solutions than other techniques. By efficiently we mean:

- Given a fixed amount of time GADO is most likely to obtain a better design than the other methods
- GADO is most likely to need less time than the other methods to obtain a design with a specific quality.

We intend to support these claims with empirical results in the remainder of this thesis.

1.5 Thesis organization

This thesis focuses on describing GADO and presenting evidence for its strength as a tool for design optimization in engineering domains. Chapter 2 describes in detail the architecture and operation of GADO. Chapter 3 presents an experimental comparison between the performance of GADO and several "off-the-shelf" state-of-the-art techniques for optimization, in two different realistic engineering design domains. The results demonstrate the superiority of GADO to these other techniques. Chapter 4 presents more analysis of the scope of applicability of GADO by applying it to several benchmark design optimization problems. It also presents an analysis of the contribution of some of the novel components of GADO, and an analysis of the performance sensitivity to parameter variation and problem structure. Chapter 5 concludes with a summary of the thesis. It also reviews the major contributions and discusses the limitations and future work. Appendix A lists all the external parameters that control the operation of GADO and their default values. Appendix B describes some of the benchmark design domains used in this thesis. Finally, Appendix C includes a case study in which GADO was used by a design engineer to design a missile inlet.

Chapter 2

Architecture of GADO

This chapter provides a detailed description of GADO, a genetic algorithm specifically designed and tailored to be used for global optimization in engineering design spaces with the characteristics described in Chapter 1. At a very high level of description, GADO is a highly adaptive system in the sense that it gradually changes its focus from pure exploration in the beginning of the search to exploitation towards the end. GADO was made by combining existing techniques from literature with new ideas inspired by the characteristics of the domains for which it is designed.

2.1 Overview of genetic algorithms

Genetic Algorithms (GAs) [Goldberg 1989] are search algorithms that simulate the process of natural selection and survival of the fittest. GAs attempt to find a good solution to some problem (e.g., finding the maximum of a function) by randomly generating a collection of potential solutions to the problem and then manipulating those solutions using genetic operators. In GA terminology, we say that we generate a population of solutions and refer to each solution as an individual. Each solution is assigned a scalar fitness value which is a numerical assessment of how well it solves the problem. The key idea is to select for reproduction the solutions with higher fitness and apply the genetic operators to these to generate new solutions. Again, in GA terminology these new solutions are called newborn individuals. Through mutation and re-combination (crossover) operations, better newborn solutions are hopefully generated out of the current set of potential solutions. This process continues until some termination condition is met.

In the case of numerical optimization, each solution is a vector of numbers (real

or integer). The GA attempts to find the vector with maximum fitness in a specified hypercube.[1] In the remainder of this section we describe the main components of a genetic algorithm in the context of numerical optimization.[2]

2.1.1 Representation (genotype)

In order to use GAs it is necessary to map the solutions of the problem to fixed length strings of some alphabet. The resulting strings are called the representation (genotype in GA terminology). The most common representations are **binary** and **floating point**. In a binary representation, each component of a solution vector is converted to a binary encoding and then these encodings are concatenated (in order) to form a binary string which becomes the genotype of the solution. In the case of integer components, the conversion to binary is straightforward while in the case of real vectors the components should be converted to integers first. The most common method of converting real components to integers is to use their range information. For example, if we have a real component whose value is x with lower bound l and upper bound u and we want to convert it to an integer i between 0 and $2^n - 1$ then $i = \lfloor\ 2^n * \frac{x-l}{u-l}\ \rfloor$. In floating point representation the component vector is represented with a one-dimensional array (i.e. a vector) of floating point numbers.

2.1.2 The initialization strategy

In order to start the GA evolution process, an initial population of solution vectors must be generated. The most common method of initialization in GAs is **random initialization** in which the initial population consists of random vectors uniformly distributed in the search space hypercube. Most GAs generate one random vector to initialize each population member. A few others [Bramlette *et al.* 1990] generate more than one vector (usually 2 or 3), compute their fitnesses, then use the fittest of them to initialize one population member.

---

[1] The user must specify the lower and upper bound of each vector component. This is a limitation in GAs.

[2] The use of GAs is far from being limited to optimization. The reader is referred to [Back *et al.* 1997] for a recent general survey.

2.1.3 The selection strategy

The selection strategy decides how to select individuals to be parents for newborns. Usually the selection applies some selection pressure by favoring the individuals with better fitness. The most common selection methods are:

- **Fitness proportional (roulette wheel) selection:** Each individual's probability of being selected is proportional to its fitness value.
- **Rank-based selection:** Each individual's probability of being selected depends on its fitness rank in the population rather than the actual fitness value. The most common rank-based selection methods are:
  - **Tournament selection:** To select an individual for reproduction, multiple candidates (usually two) are selected with uniform probability. The fittest of these candidates (the winner of this virtual tournament) is then selected for reproduction.
  - **Weight series selection:** In this method, each individual is assigned a weight that depends on its fitness rank in the population. Proportional selection is then done using the weights rather than the actual fitnesses. The weights are usually taken to be a decreasing arithmetic or geometric series.

Rank-based selection methods are more appropriate for use in domains where the fitness range is extremely wide. In these domains it is feared that the first individuals with high fitness values to be discovered will dominate the population and prevent other good individuals in other regions of the search space from being found.

2.1.4 Genetic operators

These operators use the parent(s) to create newborn(s). These operators are usually either:

- **Binary (crossover) operators,** which take two parents and produce a newborn that resembles both, or

- **Unary (mutation) operators,** which take one individual and produce a perturbed version of it.

Crossover operators

The basic operation in a genetic algorithm is crossover. It combines the merits of individuals to hopefully produce better ones. Some of the common crossover operators are:

- **Point crossover:** The point crossover operator aligns the genotypes of the parents. A crossover position is then randomly selected with uniform probability and the part of the first parent's genotype before the crossover position is copied to the corresponding part of the newborn. The rest of the newborn comes from its corresponding place in the second parent's genotype.

  For example, if a floating point representation is used, the space is 5 dimensional, the first parent's genotype is $(x1, x2, x3, x4, x5)$ and the second parent's genotype is $(y1, y2, y3, y4, y5)$ and the crossover point was selected to be 3, the newborn will be $(x1, x2, y3, y4, y5)$ or $(x1, x2, x3, y4, y5)$.

- **Arithmetic crossover:** The arithmetic crossover operator is specific to floating point representation. It produces a newborn whose components are the averages of the corresponding components in its parents.

  For example, if the space is 5 dimensional (like above), the first parent's genotype is $(x1, x2, x3, x4, x5)$ and the second parent's genotype is $(y1, y2, y3, y4, y5)$, the newborn will be $(\frac{x1+y1}{2}, \frac{x2+y2}{2}, \frac{x3+y3}{2}, \frac{x4+y4}{2}, \frac{x5+y5}{2})$. In vector form, if the first parent is $\bar{X}$ and the second parent is $\bar{Y}$ then the newborn is $\frac{\bar{X}+\bar{Y}}{2}$.

- **Linear crossover:** The linear crossover operator is specific to floating point representation. It produces a newborn whose components are convex combinations of the corresponding components in its parents.

  For example, if the first parent is $\bar{X}$ and the second parent is $\bar{Y}$ then the newborn is $a \cdot \bar{X} + (1 - a) \cdot \bar{Y}$ where a is a random value selected uniformly in the interval

$[0, 1]$.[3]

- **Heuristic crossover:** The heuristic crossover operator is also specific to floating point representation. It is a greedy operator that attempts to exploit the search space. The heuristic crossover operator works as follows: let the parents be $\bar{X}$ and $\bar{Y}$ such that $\bar{Y}$ is not worse in fitness than $\bar{X}$. The newborn is $\bar{Y} + r \cdot (\bar{Y} - \bar{X})$ where r is a random value selected uniformly in the interval $[0, 1]$.

- **Random crossover:** The random crossover operator can be applied to any type of representation. If applied to floating point representation, each vector component of the newborn is selected randomly (with equal probability) from either parent. In the case of binary representation, each bit in the newborn is selected randomly from the corresponding position in either parent. This operator introduces a lot of variability (diversity).

Mutation operators

Mutation introduces new genetic material to the GA population in order to maintain diversity and explore new regions. Some conservative mutation operators also help in exploiting the good regions of the space. Some of the common mutation operators are:

- **Uniform mutation:** Uniform mutation replaces each component of a solution vector with a random value uniformly selected from the component range.

- **Non-uniform mutation:** Non-uniform mutation takes the stage of optimization into consideration. At the beginning of the optimization it acts just like uniform mutation. It then becomes more and more conservative about the amount of change it makes to a vector component as the optimization progresses.

  Let the component's value be x with lower bound l and upper bound u and assuming the search is at iteration t and the maximum number of iterations is T,

[3] Linear crossover degenerates to arithmetic crossover in the case in which $a = 0.5$.

then the mutant value is

$$x_{new} = \begin{cases} x + (u - x) \cdot r \cdot (1 - \frac{t}{T}) \cdot scale & \text{with probability 0.5} \\ x - (x - l) \cdot r \cdot (1 - \frac{t}{T}) \cdot scale & \text{with probability 0.5} \end{cases}$$

where r is a random value selected uniformly in the interval $[0, 1]$ and *scale* is a number between 0 and 1 that decides how conservative the mutation should be.

2.1.5 The replacement strategy

In a **steady state GA**, the generation of each new individual is accompanied by the death of one individual of the current population to make room for it. Another type of GAs is **generational GAs** in which the entire population (or a large fraction of it) is simultaneously replaced. Steady state GAs are likely to converge faster than generational GAs. The argument is that if each newborn is on average better than the elements of the existing population (this is a necessary condition for convergence), then it should be introduced to the population immediately.

The replacement strategy is unique to steady state GAs. It decides how to make space for the newborn when the population is full. Some of the most common replacement strategies are:

- **Elitist replacement:** selects for replacement a random individual, on condition that the best *elite fraction* of the population is not replaced. The elitist fraction may be constant or dynamic.
- **Crowding replacement:** This is a family of more sophisticated replacement strategies that take into consideration other factors in addition to fitness (such as preserving diversity in the population for example). Each such strategy is called a *crowding heuristic* and the reader is referred to [Mahfoud 1995] for a detailed discussion of these methods.

2.2 System structure

GADO is a novel GA that is tailored for the specific task of optimization in design spaces. It combines existing techniques which are proper for this task with new ideas.

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 (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). The user is required to specify a number of external parameters (for example the maximum number of iterations). Appendix A lists all external parameters and their default values. In the remainder of this section we describe the system in more detail. For ease of description we define the following two terms:

- **goup:** The proportion of iterations that have been done thus far during an optimization. This is initially zero and increases linearly to one at the very end of the optimization.
- **godown:** The proportion of iterations that remain to be done thus far during an optimization. This is initially one and decreases linearly to zero at the very end of the optimization.

Note that the following equation always holds at any stage of the optimization: $goup + godown = 1$

2.2.1 Representation

Each individual in the GA population represents a parametric description of an artifact, such as an aircraft or a missile. All parameters have continuous intervals. GADO uses a floating point representation (genotype), the reason being the demonstrated superiority of the floating point representation to binary representation in continuous variable optimization problems [Janikow and Michalewicz 1991]. 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). The penalty function is the product of an adaptive penalty coefficient and the sum of the constraint violations if any.

2.2.2 Dynamic penalty function

One of the novel and very convenient features of GADO is its ability to adjust the magnitude of the penalty coefficient depending on situation. If this feature did not exist, the user would have to specify a single penalty coefficient to be used over the entire course of optimization and it is difficult to come up with a penalty coefficient that is good for all stages of optimization. It is better to use a modest penalty in the initial stages to insure adequate sampling of the search space and then gradually increase the penalty to force the optimization to converge to a feasible solution. This is exactly what GADO does, by dynamically changing the penalty coefficient on demand. The initial penalty coefficient is one of the external parameters. Its default value should be computed as follows: if the measure of merit is expected to range between V and $10 * V$ where V is a power of ten, then the initial penalty coefficient will be set to $\frac{V}{100}$.[4]

GADO keeps track of two key individuals of the population:

1. The point that has the least sum of constraint violations.

2. The point that has the best fitness value.

If both points are the same then the penalty coefficient is assumed adequate, otherwise the penalty coefficient is increased to make the two points have equal fitness values. This increase is important in order to force the search towards feasibility, or else the search may wander deeply into an infeasible region with tempting objective function values. We impose a multiplicative limit on the allowed increase of the penalty coefficient at each increment to prevent abrupt changes that may be unnecessary. This limit is one of the parameters, its default value is two.

Another optional feature is to let GADO decrease the penalty coefficient if at some stage the population contains no infeasible points. The intuition behind this feature is that sometimes the best solutions are at the border of constraint violations. If in the coarse of pushing the optimization towards feasibility the penalty coefficient gets too

[4]It is assumed that the numerical magnitude of the constraints is comparable to that of the measure of merit. If this is not true then it may be useful to use a different method for computing the initial penalty coefficient to reflect this fact. The goal is to make the product of the initial penalty coefficient and the sum of constraint violations small compared to the measure of merit.

high, these points will be very hard to reach as the search will approach them single sided. If the penalty coefficient is to be decreased, it is divided by the fourth root of the multiplicative limit described above. Thus, if the penalty coefficient gets too small, it can grow back to a reasonable value relatively fast.

2.2.3 Initialization

GADO uses a novel initialization strategy. If the population size is P, GADO will generate a much larger number of random individuals (this number is one of the external parameters; its default value is 10*P). These individuals are then inserted into the population using the replacement strategy described below. This method provides a more representative initial sample of the search space as opposed to just using P random individuals. The presence of many unevaluable and infeasible points in the search space (the majority of points are like this) makes it very important to use a method like this, to ensure having an adequate number of evaluable points in the initial population.

2.2.4 Selection

Here selection was performed by rank 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 search and causing premature convergence. Each individual is assigned a weight depending on its fitness rank in the population. The weights form an arithmetic series as follows: if the population has P individuals, the best individual is given a weight of P, the second best is given a weight of P-1 and so on until the worst individual gets a weight of 1. This approach results in an average amount of selection pressure which promotes convergence without endangering diversity. In the final stages of the optimization (in the last 25% of the iterations) a more greedy form of selection is used. Instead of using weights, the individual selected for reproduction is the one whose fitness rank is $1 + r_1 \cdot r_2 \cdot (population\_size - 1)$ where r_1 and r_2 are two random values selected uniformly in the interval $[0, 1]$. This greedy form is used in order to promote exploitation in the final stages of the search (note that the product of 2 numbers, each of which is less than one, is usually a very small

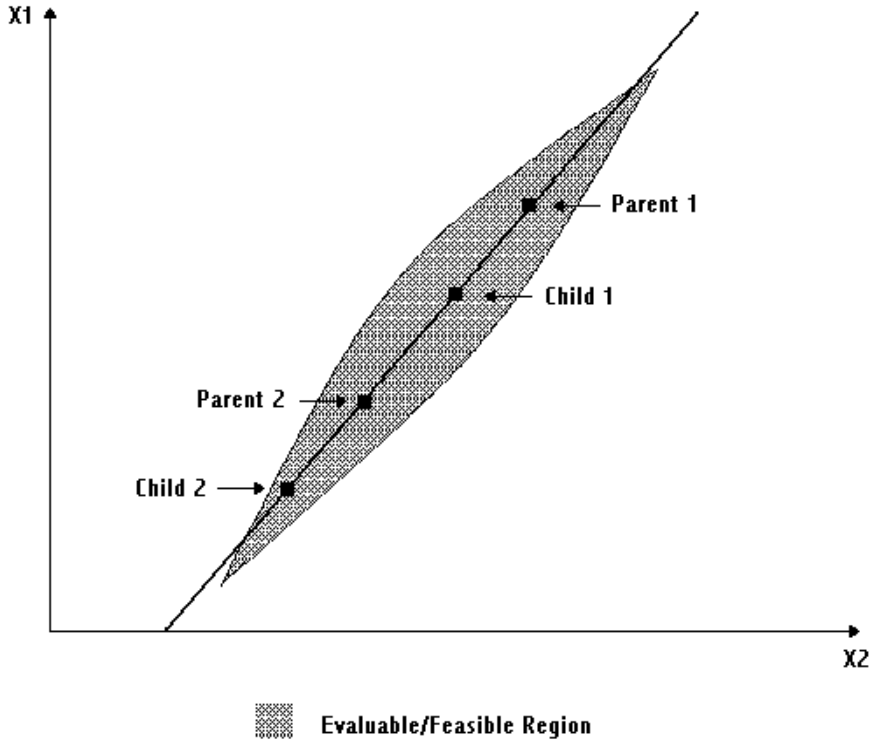

Figure 2.1: Line Crossover behavior

number).

2.2.5 Crossover operators

GADO uses five different crossover operators. Three of these operators are novel (guided crossover, line crossover and double line crossover). The other two are point crossover and random crossover. At every iteration, GADO randomly decides which crossover method to use as follows: with probability *guided_crossover_factor**goup GADO will do guided crossover (as will be described in more detail below). Otherwise, GADO will randomly choose one of the other four crossover methods with equal probability. The quantity *guided_crossover_factor* is one of the external parameters and its default value is 0.25.

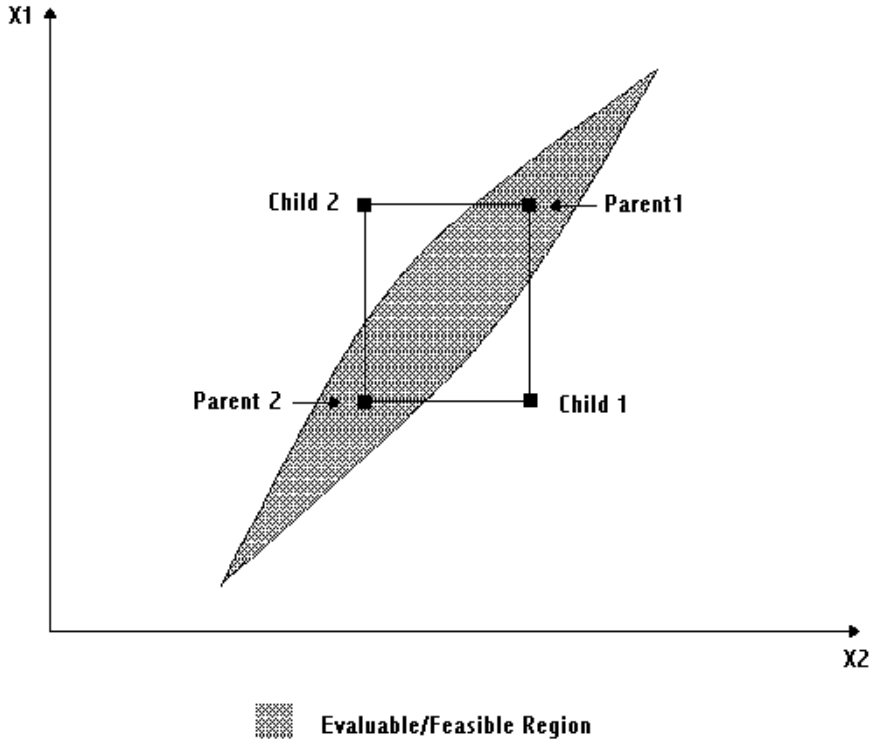

Figure 2.2: Point Crossover behavior

Point crossover

The point crossover operator was described in detail in the previous section. It basically does a cut and paste operation on the genotypes of the parents to produce a newborn. It was also given other names in the literature, such as real crossover [Wright 1990].

In design optimization, point crossover is a valuable tool in decomposable spaces. It can combine the merits of two designs to produce a design that is better than both. The user is thus encouraged to arrange the design parameters in an order that groups the parameters pertaining to the same component/discipline together.[5]

[5]Our experience suggests that in fact most users do this naturally.

Line crossover

Line Crossover (LC) is a new operator introduced in this research. It resembles several genetic operators (such as arithmetic crossover, linear crossover and heuristic crossover [Michalewicz 1996]). Line crossover works by joining a line between the two parent points, extending it from both sides and randomly picking a point on that line or its extensions (with uniform probability) to be the newborn. The amount of extension depends on the side and the stage of the search. Suppose the length of the line segment between the parents is L. Then the amount of extension from the side of the worse parent is set to 2*L regardless of the stage of the search. The amount of extension from the side of the better parent starts at 2*L and increases linearly to 4*L at the end of the search. More formally the amount of extension from the side of the better parent is equal to (2+2*goup)*L.

The use of line crossover is motivated by the fact that many design spaces have good regions which resemble very thin hyper-ellipsoids such as that in Figure 2.1. The figure illustrates the operation of line crossover in a hypothetical two dimensional space. In these spaces, other crossover techniques such as point crossover will have a great deal of difficulty searching the space. Unless both parents are close to each other, point crossover may well start with two good parents and produce a bad newborn. Figure 2.2 illustrates this situation in the same hypothetical two dimensional space.

Double line crossover

This is a novel operator introduced by GADO with the goal of combining the merits of both point and line crossover. The double line crossover does the following:

- Selects a random crossover point with uniform probability (like point crossover).
- Performs 2 line crossover like operations, one on the prefixes of the parents and the other on their suffixes. The only difference between these operations and the line crossover operation is that they join a line between the parent points and extend it by twice its length from both ends (as opposed to extending the line more on the side of the better point, which is what line crossover does).

We decided to include this operator (rather than just relying on the point crossover and line crossover) because sometimes the line crossover forces some of the parameters to move in a direction that degrades performance in order to move others in the right direction. By disengaging the parameters before doing line crossover we have a good chance of avoiding this problem, especially in decomposable spaces.

Guided crossover

This is a novel operator introduced in GADO. Its aim is to improve the final performance (i.e. to make the final solution as close as possible to an optimum).

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 Mutual_fitness(X,$candidate_1$) is computed, where

$$Mutual\_fitness(A,B) = \frac{(fitness(A) - fitness(B))^2}{Euclidean\_distance(A,B)^2}$$

   A choice for X that maximizes Mutual_fitness(X,$candidate_1$) is taken to be $candidate_2$.[6]

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:

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

[6] In the last 5% of the iterations, the point with the best fitness in the current GA population is taken to be $candidate_2$ (unless this is $candidate_1$, in which case the point with the second best fitness is taken to be $candidate_2$). This is an effort to force the best point at the end of the optimization to be locally optimum.

where L is a uniformly distributed random number in the interval [1-0.2*x,1+x] and x is a function of the number of elapsed iterations and the total allowed number of iterations such that:

$$x = 0.75 * godown + 0.25$$

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 decreases 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 is one of the external parameters. Its default value is 0.25.

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. Instead, the quantity Mutual_fitness(A,B) used to rank directions serves as a crude form of gradient calculation and does not entail a single additional evaluation of fitness.

2.2.6 Mutation operators

GADO does a mutation operation following each crossover, before the newborn is used any further. GADO uses three mutation operators, two of which are new (shrinking window mutation and greedy mutation) and the third is non-uniform mutation

(the *scale* of the non-uniform mutation is set according to a formula that will be described below). They are used with the following probabilities: Greedy mutation (0.1), shrinking-window mutation (0.4) and non-uniform mutation (0.5). Mutation is done at the parameter level (i.e. each parameter of the mutated individual can be mutated using a different mutation operator). Each parameter has an equal probability of being mutated. This probability starts at 0.5 and decreases quadratically to 0.1 over the course of the optimization (i.e. the probability is equal to $0.1 + 0.4 * godown^2$ at any iteration).

Shrinking window mutation

Shrinking-window mutation is done by randomly perturbing a parameter of the newborn. The perturbation window shrinks as the optimization progresses. The perturbation window size also depends on the type of crossover method that was used to generate the newborn. The reason for this dependency is that different crossover methods introduce different amounts of diversity. For example, a newborn resulting form random crossover is likely to be quite different from either of its parents and thus it is logical to apply a more conservative perturbation in this case than in the case of point crossover.

If the parameter's value is x with lower bound l and upper bound u, then the mutant value is randomly selected with uniform probability from the interval $[x - godown * (u - l) * \frac{scale}{2}, x + godown * (u - l) * \frac{scale}{2}]$.[7] The *scale* value is set as follows:

$$scale = \begin{cases} 0.15 * mutation\_factor & \text{if point crossover was used} \\ 0.10 * mutation\_factor & \text{if random crossover was used} \\ 0.05 * mutation\_factor & \text{if line crossover or double line crossover was used} \end{cases}$$

where $mutation\_factor$ is one of the external parameters, with a default value of 2.[8]

The major difference between shrinking window mutation and non-uniform mutation

[7] If the mutant value is out of bounds, it is forced to the nearest bound.

[8] The same formula is used to compute *scale* in the case of non-uniform mutation.

is that the latter discriminates against the portions of the space near the borders of the hypercube. These portions are likely to contain optima and should be searched very carefully.

Greedy mutation

Unlike the non-uniform mutation and the shrinking window mutation, this mutation operator does not vary its behavior with the stage of optimization.

If the component's value is x with lower bound l and upper bound u, then the mutant value is

$$x_{new} = \begin{cases} x + (u - x) \cdot r_1 \cdot r_2 & \text{with probability } 0.5 \\ x - (x - l) \cdot r_1 \cdot r_2 & \text{with probability } 0.5 \end{cases}$$

where r_1 and r_2 are two random values selected uniformly in the interval $[0, 1]$.

The purpose of this mutation operator is to maintain reachability of the entire search space in all stages of the optimization. This is done conservatively in order not to waste time (the product of 2 numbers, each of which is less than one, is usually a very small number).

2.2.7 Replacement strategy

The replacement strategy used here is a crowding technique [Mahfoud 1995], which takes into consideration both the fitness and the proximity of the points in the GA population. When a new point is introduced, the closest point to this new point from among a certain "group" is selected for replacement. The "group" includes those points which are:

- worse (in fitness) than the point being introduced, and
- among the worst (in fitness) fraction of the points in the current population

The worst fraction of points among which a point has to be to become a candidate for replacement is called the *crowding factor*. For example, if the crowding factor had a value of 0.25 at some iteration, then the worst quartile of the current population is

eligible for replacement at that iteration. Both the initial and the final value of the crowding factor are external parameters. The crowding factor changes linearly with the number of iterations from its initial to its final value. The default is that the crowding factor's initial value is one (i.e. the entire population is eligible for replacement, this is a pure crowding situation) and drops linearly to its final value of zero (i.e. replacement of the worst) at the end of the optimization. The user can override the default and specify different values for these. In a multi-modal situation in which the user is interested in finding as many solutions as possible rather than only the best solution, both initial and final values of the crowding factor can be set to one. On the other hand, if for lack of time or whatever reason the user wants a local solution as soon as possible, the two values can be set to zero.

2.2.8 Search control

The search control modules are among the completely novel ideas in GADO. The main reason for this is that they entail a great deal of timing overhead. In most 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 in these cases is more often than not an 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 and in fact - as we will demonstrate in subsequent chapters - search control proved to be one of the major contributers to the success of GADO.

Screening module

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 one

of the external parameters 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 one of the external parameters 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 or the evaluation function is too expensive to permit thorough exploration of the space). 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 the candidate point is just discarded.[9] 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. 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.

It is possible to imagine spaces in which the screening module may actually make the global optimum harder to find. Figure 2.3 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 narrow basin of attraction.[10] 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

[9] In the actual implementation we evaluate the point with a very low probability (1%). This avoids the theoretical possibility of deadlock (if the screening module keeps rejecting every point) and gives small regions of good points surrounded by bad points a chance to be discovered.

[10] 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.

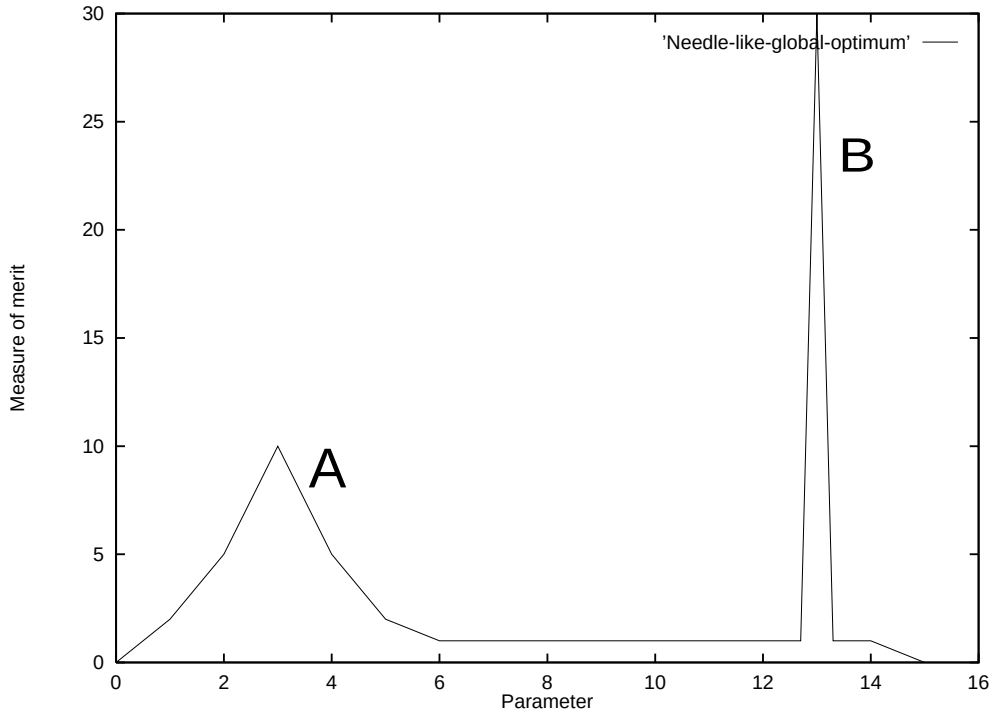

Figure 2.3: Possibility of screening module failure

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.

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. Case-based learning 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 traditional inductive learning methods. Also in the field of GA optimization [Louis 1997] presented a method for using information about an entire GA optimization to guide another GA optimization in a similar domain. He presented his work in a design optimization framework.

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 referring to similar queries executed in the past. PROLEARN uses 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 supports a broad variety of problem-solving strategies. Soar 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. Since such new rules are added immediately to the problem space, they can be used at later points 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 based on a means-ends planner that decomposes problems into subgoals, solves them, then combines their solutions into a solution for the full problem. PRODIGY 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.

Diversity maintenance module

One of the worst things that can happen during a GA optimization is premature convergence to a non-global optimum. In a steady state GA, such as the one used in this research, this happens when the current population loses diversity and all the points become very close to each other. The use of the screening module, unfortunately, does not protect the GA against premature convergence. In fact the SM, as described above, may even promote such undesirable behavior. To protect the GA against premature convergence the diversity maintenance module (DMM) does the following:

- The DMM rejects points that are extremely close to points that have already been evaluated (good or bad). The rationale behind this is that such points carry redundant information and there is no point in introducing them to the current population. The amount of closeness to existing points that warrants rejection — the rejection radius — is set based on the initial average distance between the points in the starting population, the size of the population and the rejection tolerance. The initial value of the rejection radius is equal to $\frac{reject\_tolerance * initial\_average\_distance}{population\_size}$

where $reject\_tolerance$ is one of the external parameters. In addition, the rejection radius decreases quadratically to zero with the number of iterations, so as not to prevent the GA from converging to the global optimum towards the end of the search. Since the SM already determines the nearest neighbor for each new point, this operation does not add any significant extra overhead to the search.

- If severe loss of diversity is detected (i.e. the average distance between the points of the current population becomes less than $reseed\_fraction * godown^2 * initial\_average\_distance$ where $reseed\_fraction$ is one of the external parameters with a default value of 0.25) during the course of the optimization, a re-seeding operation is done. All the points in the current population are discarded except for the best one. The population is then rebuilt using the points accumulated by the SM, with preference going to points that both have good fitness and are far from the retained best point. The way this is done is simply to insert all the points accumulated by the SM into the GA population using the replacement strategy described above, except that the points that are too close to the best point are not inserted. More precisely, we exclude the points that are closer in distance to the best point than $2 * reseed\_fraction * godown^2 * initial\_average\_distance$. This technique restores diversity and gives the GA a second chance to avoid local sub-optima. On the other hand, re-seeding could never take the population away from the global optimum if it is reached, because the best point is retained (in this case the global optimum) and the population will quickly re-converge thereafter. The user must specify the maximum number of times the reseeding may occur. The maximum number of reseeds is one of the external parameters. Its default value is 10. If the specified maximum number of reseeds have already been done, the optimization proceeds without further reseeding until termination.

2.2.9 Stopping criteria

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. The user decides what level of diversity loss warrants stopping through a parameter called *stopping tolerance* in the parameters file. If *stopping tolerance* is set to zero, the GA will only stop when it exhausts the maximum number of evaluations. On the other hand, the user can set GADO to continue beyond the maximum number of evaluations - which he still has to specify as it is needed for some calculations - and stop only by diversity loss. This mode of operation may be useful for, say, an engineer who needs a quick result for proof of concept or feasibility study and then a possibly better result for the practical design. Most GA optimization packages offer only one stopping criterion, and that is number of iterations. Thus, this is a novel and useful feature in GADO.

2.3 The effectiveness of GADO for engineering design optimization

GADO addresses the problems described in Section 1.2 as follows:

- **Unevaluable and infeasible points/regions:** This is targeted by various features of GADO. The screening module avoids spending too much time exploring unevaluable regions. In addition, the adaptive penalty scheme makes efficient use of the infeasible points.

- **Expensive evaluations:** The search control methods (screening module and diversity maintenance module) improve the time efficiency of GADO by avoiding unnecessary evaluations of unpromising points and redundant points.

- **Badly structured spaces:** GADO uses a collection of crossover and mutation operators that are tailored for such cases (for example the line crossover operator).

- **Multiple local optima:** GADO uses a replacement strategy that promotes niching [Mahfoud 1995]. This promotes "species" formation and prevents any single good region from dominating the search. The diversity maintenance modules also helps in this regard. Maintaining diversity gives the algorithm a very good chance of visiting the regions of many optima before converging on (hopefully) the best.

- **Non-smoothness:** GAs in general are less sensitive to this problem than, say, gradient based methods. In addition, the search control methods in GADO help in curing this problem.

2.4 Concluding remarks

This chapter described GADO, a new genetic algorithm specifically designed for use in engineering design optimization. It described the different components and the operation of the system. The chapter also provided an overview of genetic algorithms. The remainder of this thesis is devoted to demonstrating the strength of GADO and analyzing its performance in different engineering design domains. In the next chapter we demonstrate the superiority of GADO to existing optimization methods when used in design optimization. We compare GADO's performance to that of three state of the art optimizers representative of three different paradigms of design optimization.

Chapter 3

Empirical Support

In this chapter we demonstrate the merit and strength of GADO by comparing its performance to that of four different global optimizers in two different realistic engineering design domains.

3.1 Baseline methods

The selected optimizers are state of the art representatives of the different classes of optimizers described in Chapter 1 as well as a very simple optimizer to serve as a baseline. These optimizers are:

- **Random Probes (RP):** RP generates random points in the design space and evaluates them. Each design parameter is assumed to have a fixed range. RP forms design points by selecting a random value of each parameter uniformly distributed over its range. The best point found in this process is the result.

- **Multi-start CFSQP:** CFSQP [Lawrence *et al.* 1995] is a state-of-the-art implementation of the Sequential Quadratic Programming method, a quasi-Newton method that solves a nonlinear constrained optimization problem by fitting a sequence of quadratic programs. CFSQP is a gradient based local optimizer. It is made into a global optimizer by restarting it from different random points in the search space. In the early stages of this research, a large number of conventional nonlinear programming methods were examined. CFSQP proved to be the best in the domains under consideration and thus we chose it to represent conventional nonlinear programming methods in this comparison. The version of CFSQP we used was tailored to design optimization spaces by Mark

Schwabacher [Schwabacher 1996]. It was shown [Schwabacher and Gelsey 1997, Schwabacher 1996] that the modifications to this version of CFSQP made it considerably better in several design optimization spaces.

- **GENOCOPIII:** This is a state of the art "off-the-shelf" genetic-algorithm-based global optimizer for constrained continuous optimization. It is specifically designed to be robust in domains with small, possible disjoint feasible regions. This system has been launched only in 1996 and is described in detail in [Michalewicz 1996]. It uses a repair method for handling constraints. The repair method works as follows: whenever a point generated by the GA is infeasible, it is pushed towards the feasible region by joining a line from it to a feasible point and searching this line until another feasible point is found. The algorithm maintains, in addition to a general purpose population of points (the search population), another population of strictly feasible points (the reference population). Both populations are evolved and the reference population provides points for repair. GENOCOPIII was able to solve very difficult nonlinear optimization problems, finding better optima than the ones that were thought to be global. GENOCOPIII must find a feasible point to seed the reference population. This may be very difficult in domains with extremely small feasible regions. If this is the case, GENOCOPIII will ask the user to supply a feasible point. This behavior may be unacceptable in some domains.

- **ASA:** This is a state of the art "off-the-shelf" implementation of the simulated annealing paradigm. The version we used was launched in 1995. ASA[1] is an implementation of the very fast simulated re-annealing algorithm (VFSR) proposed by Lester Ingber [Ingber 1996]. To the best of our knowledge, this is the only simulated annealing implementation that does not require the user to provide an initial temperature or an annealing schedule. ASA proved to be a very strong global optimizer, especially in highly multi-modal problems (including problems with 10^{12} local optima!). The main disadvantage is that ASA must start from

[1] ASA is an acronym for Adaptive Simulated Annealing.

a feasible point. If the user does not provide a feasible starting point, ASA will do a randomized search for one, effectively defaulting to random probes. In the domains where the feasible region is extremely small, ASA may not be usable.

3.2 Evaluation methodology

We ran each of the above optimizers, as well as GADO, many times in each domain. We used the default values for external parameters (i.e. we did no tuning for each problem) in the case of GADO, ASA and GENOCOPIII.[2] In the case of CFSQP we used the data about runs that were done by other researchers, where the parameters were tuned for each problem. This represents a higher challenge for GADO, especially that the version of CFSQP used was adapted to this class of problems. We compared the average performance, as well as the worst case performance of the methods. The worst case performance is important in this context because the evaluation functions are very expensive. Very often the engineer will only have time to run the optimizer just once, so an optimizer with bad worst case performance but excellent average performance may be undesirable.

3.3 Results

3.3.1 Application domain 1: Supersonic transport aircraft design domain

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.* 1996b]. Figure 3.1 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

[2]The only exception to this rule was that we used a population size of 5 times the dimension (rather than the default of 10 times the dimension) in the missile design domain for GADO and GENOCOPIII. We did this so that the populations will be able to converge in a relatively small number of iterations as the evaluation function in this domain was very expensive (6 CPU seconds).

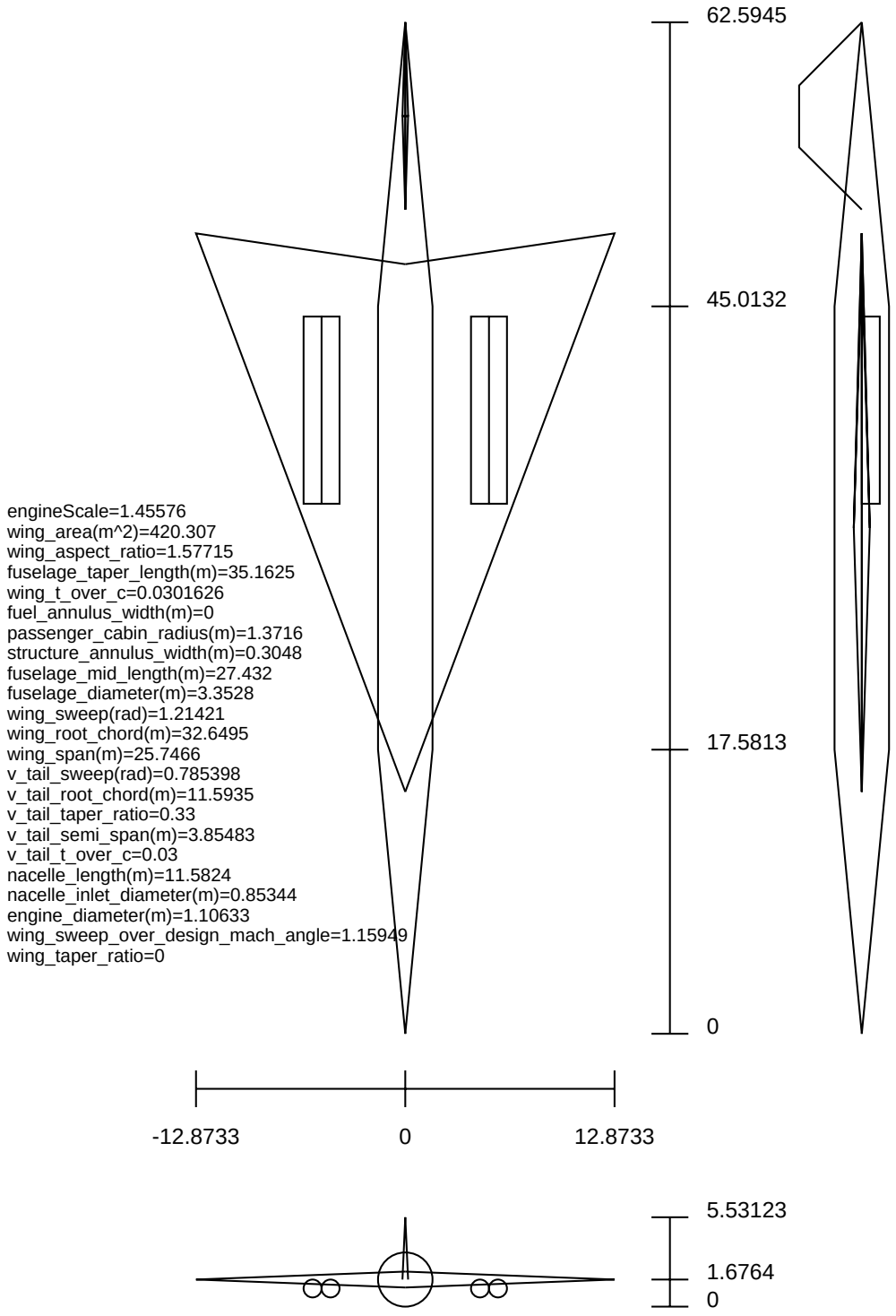

Figure 3.1: Supersonic transport aircraft designed by our system (dimensions in feet)

Table 3.1: 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

twelve of the aircraft conceptual design parameters in Table 3.1 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 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

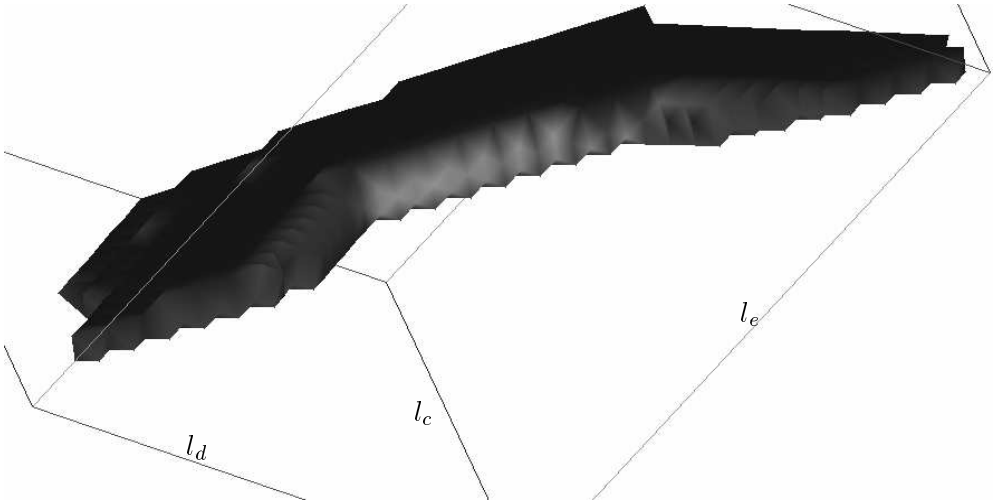

Figure 3.2: The "slab-shaped" evaluable region (We normally display this in color on a workstation screen, with differing colors indicating differing design quality.)

the value of the objective function.

Figure 3.2 shows a three dimensional projection of the search space on the first three parameters (l_c, l_d and l_e) of Table 3.1. The blank regions of the curve are unevaluable points. The figure illustrates the "slab-shaped" evaluable region in this three dimensional subspace. The figure clearly shows how difficult it is to do optimizations in this domain.

In summary, the problem has 12 parameters and 37 inequality constraints. 0.6% of the search space is evaluable.[3]

Experiments and results

GADO vs. RP, GENOCOPIII and ASA

We first attempted to run all optimizers in this domain. It turned out that the feasible region was extremely small. An RP run of 50000 evaluations did not find a single feasible point. Consequently, RP failed completely in this domain. Similarly, GENOCOPIII and ASA were unable to start as they use random search to find the first feasible point.

[3]No statistics exist regarding the fraction of the search space that is feasible because it is extremely small.

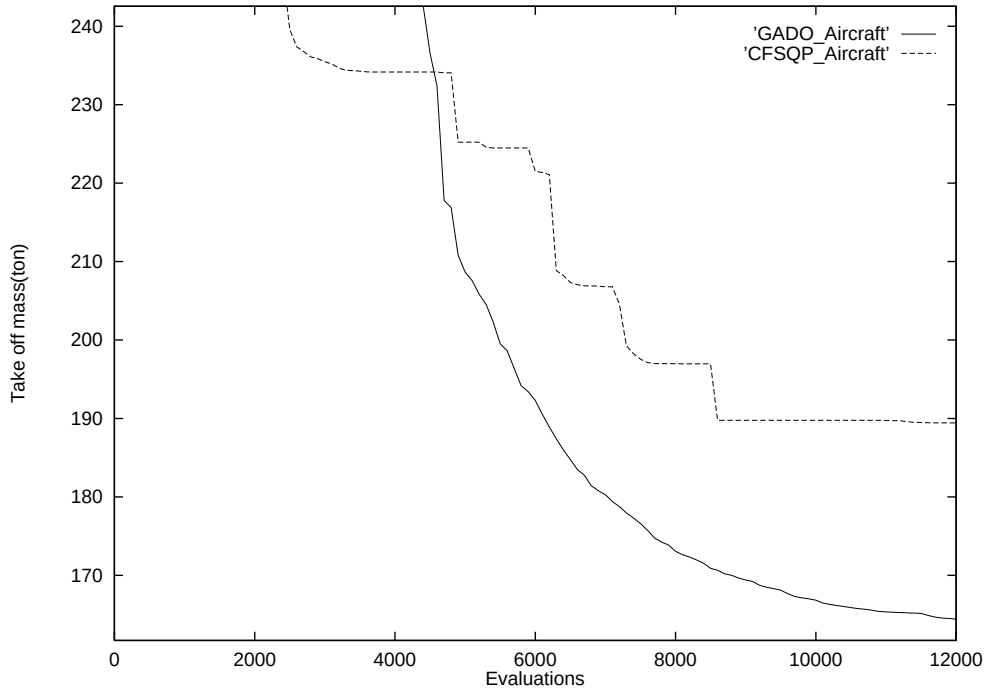

Figure 3.3: Comparison of average performance of GADO and CFSQP in application domain 1 (aircraft design)

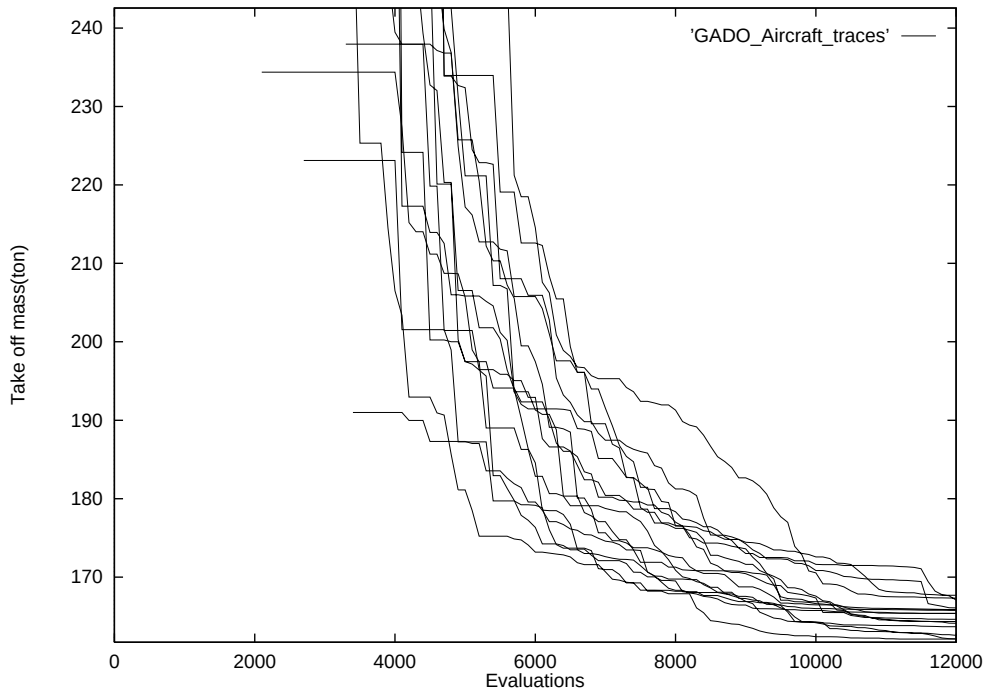

Figure 3.4: Trace of 15 GADO runs in application domain 1 (aircraft design)

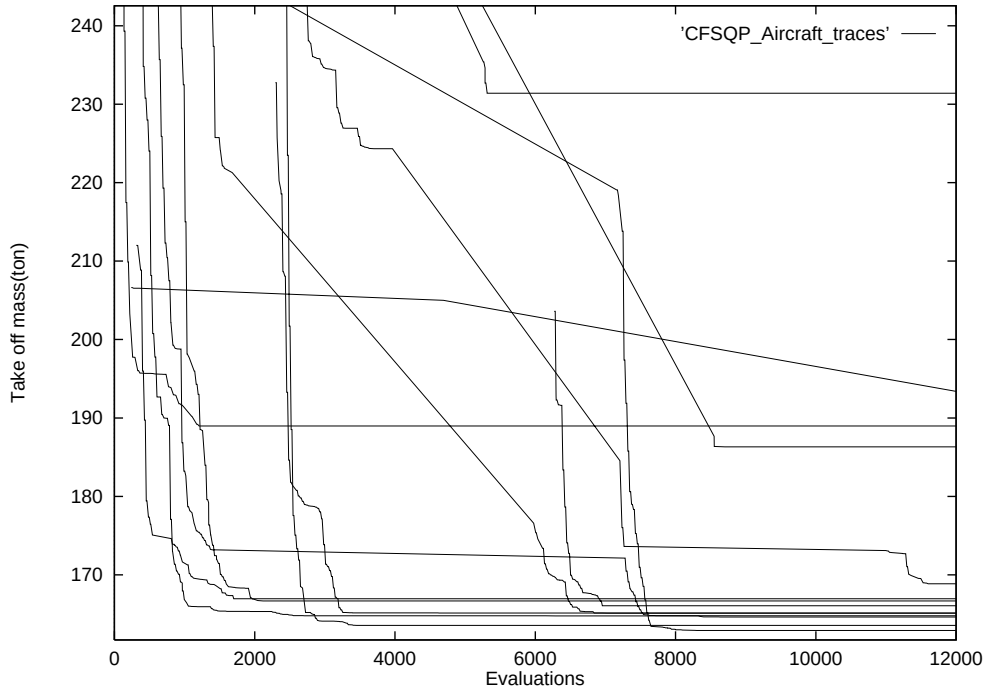

Figure 3.5: Trace of 15 multistart-CFSQP runs in application domain 1 (aircraft design)

GADO vs. CFSQP

The only optimizers that were able to actually optimize without requiring any external help were GADO and multi-start CFSQP. Each was run 15 times with different random seeds. CFSQP does random search until it finds an evaluable (not necessarily feasible) point and then works from there in two phases:

1. It first tries to move to a feasible point by using constraint gradients to minimize violation.

2. If it succeeds in the first phase it then attempts to optimize in the feasible region using constraint projections to avoid stepping into infeasibility.

The average performance of GADO and multi-start CFSQP is illustrated in Figure 3.3.[4] In order to plot meaningful curves for the average performance, in the presence of unevaluable and infeasible points, we substituted a large number (375) for the take-off mass for these points. We chose this number because it was larger than the take

[4]We use the following rule for the legends of curves in all the figures in this thesis: the label of each curve in the legend will have the form <optimizer name>_<domain name>_[comment if any].

off masses of all the feasible designs we have seen. The leading, almost vertical edges of the curves represent the regions where all the runs found the feasible region. One run of CFSQP failed to reach the feasible region, despite restarting from 10 random points. To be able to draw the figure we treated this run differently, substituting 375 for the takeoff mass of all. Despite this, the figure clearly demonstrates the superiority of GADO. On average, the final best design of each GADO run had a takeoff mass of about 164.9 tons. In all the experiments done by us and by other researchers using more than 10 different optimizers and millions of evaluations, the best takeoff mass ever found was 161.7 tons (it was found by GADO).

Figure 3.4 shows all 15 runs of GADO, and it is evident that all the runs are similar to each other except for the very early stages.[5] The final standard deviation was about 1.7 tons. The worst case run gave a final takeoff mass of 167.6 tons which is only 3.6% higher than the best optimum ever found. Compare this to not finding any feasible points, which is the worst case performance of CFSQP.

Figure 3.5 shows all 15 runs of multi-start CFSQP.[6] The figure clearly demonstrates the large variance between runs. Some runs were very lucky and got close to the best optimum in a few thousand iterations, others were very unlucky and never got close to the best optimum.

GADO vs. GENOCOPIII and ASA (revisited)

As a final experiment to calibrate performance, we took the first found feasible point of one GADO run and gave it to GENOCOPIII and to ASA so they could get started. Obviously this favors GENOCOPIII and ASA over GADO. We then compared their progress with that of the same GADO run. The result is shown in Figure 3.6. The figure clearly demonstrates that the feasible starting point requirement was not the only problem GENOCOPIII and ASA had in this domain. Their performance was still poor. Specifically, their performance was worse than the average performance of CFSQP.

---

[5] Each curve starts from the iteration in which the corresponding run first found a feasible point.

[6] Actually there are only 14 curves since, as mentioned earlier, one run never found any feasible points so it had no curve.

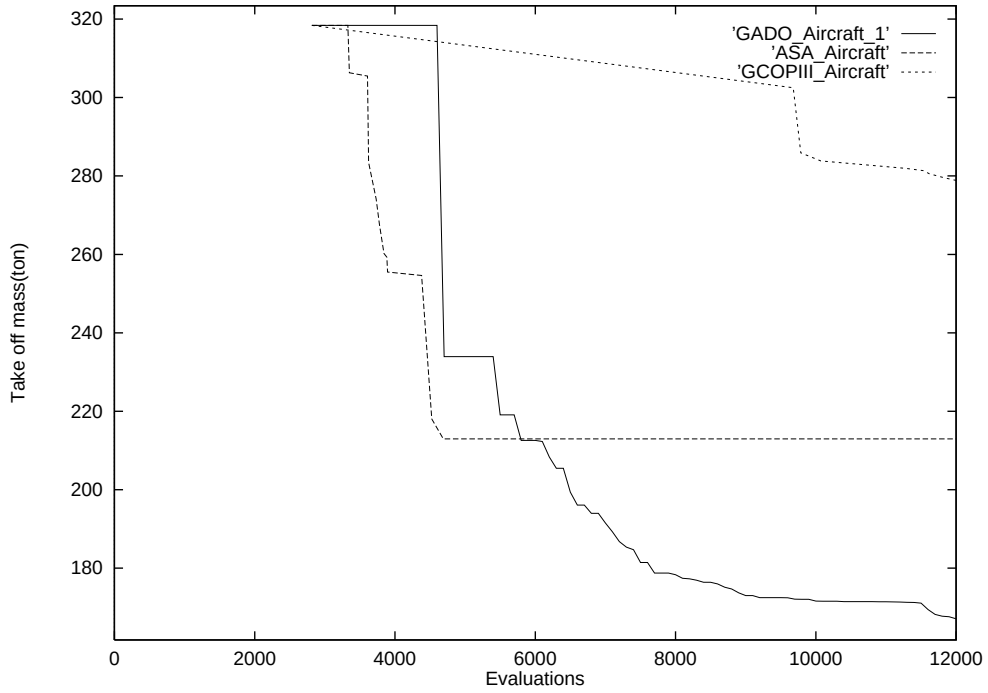

Figure 3.6: Comparison of GADO, GENOCOPIII and ASA in application domain 1 (aircraft design)

3.3.2 Application domain 2: Supersonic cruise missile inlet domain

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 60000 feet altitude. Minimum manufacture cost for this inlet is critical,

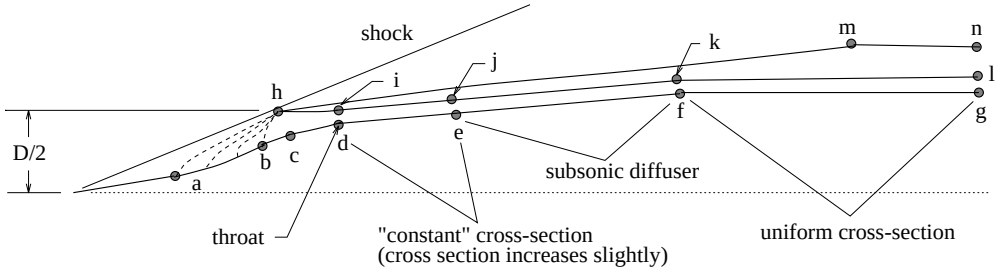

Figure 3.7: Supersonic missile inlet geometry model

Table 3.2: 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 3.3: 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

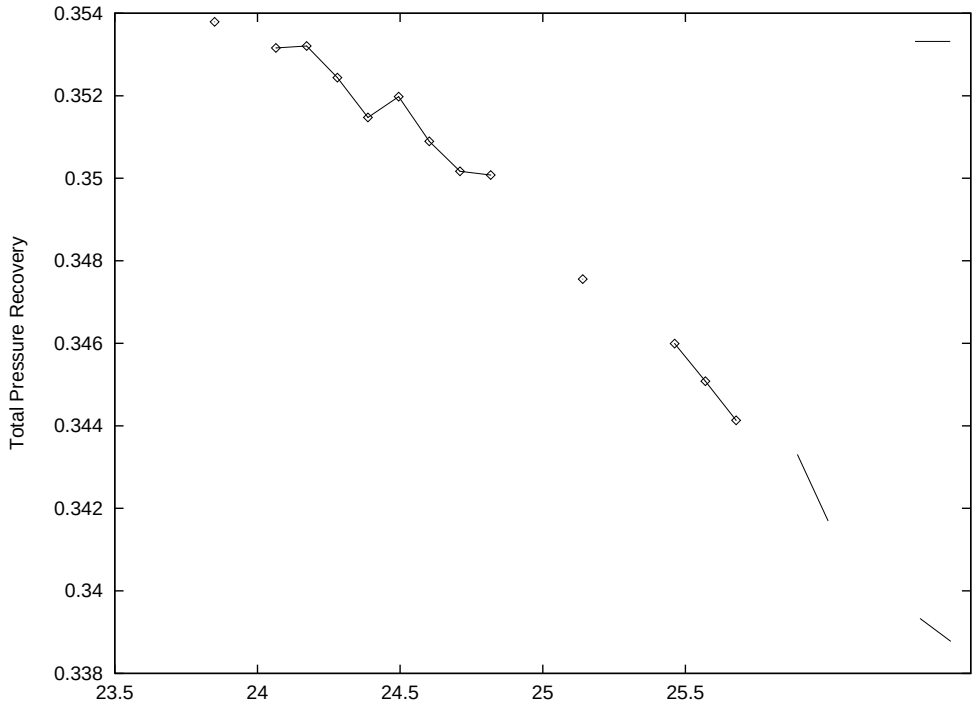

Figure 3.8: A cross section of the search space defined by NIDA.

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 3.7 shows the model of the missile geometry which is composed of six fixed parameters and eight design parameters given in Table 3.2 and Table 3.3, 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 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. Unfortunately, NIDA suffers from a number of serious shortcomings. There are numerous small discontinuities in the

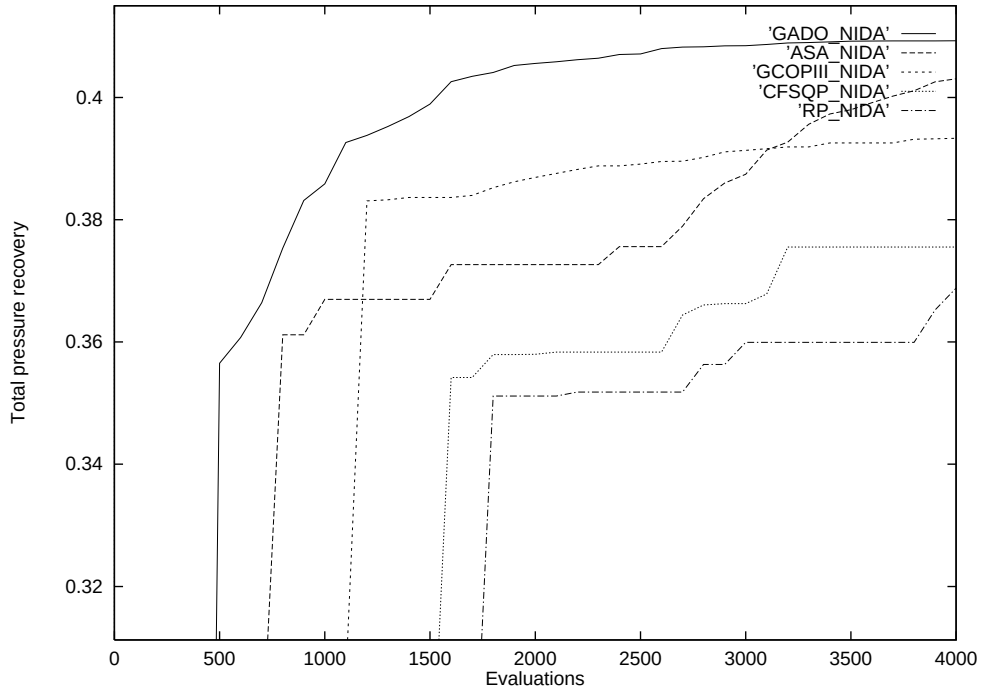

Figure 3.9: Comparison of average performance in application domain 2 (missile design)

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 3.8 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 clearly shows how difficult it is to do optimizations in this domain.

The eight design parameters (all continuous valued) are given in Table 3.3, 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.

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

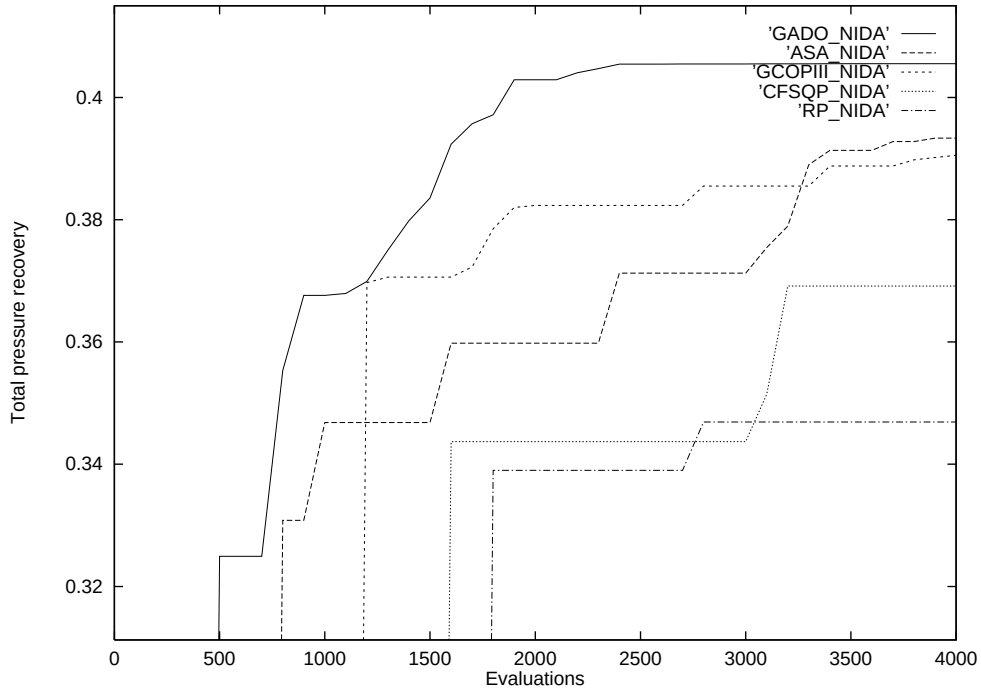

Figure 3.10: Comparison of worst case performance in application domain 2 (missile design)

Experiments and results

All five optimizers were able to run without external help in this domain. The feasible region was much larger than the case of application domain 1 above. Each optimizer was run 5 times. All optimizations found feasible points. Figure 3.9 illustrates the average performance of the different optimizers.[7] In order to plot meaningful curves for the average performance, in the presence of unevaluable and infeasible points, we substituted the number zero for the total pressure recovery of unevaluable and infeasible designs (any feasible design must have a total pressure recovery larger than zero). The best total pressure recovery ever found in this domain was 0.415 (it was found by GADO). The average final total pressure recovery was 0.409 for GADO and 0.403 for ASA. In general, the figure clearly shows that GADO is superior to all the other methods in this domain.

We compared the worst case performance of the different optimizers. The result is

[7]Note that this is a maximization problem so higher curves are better.

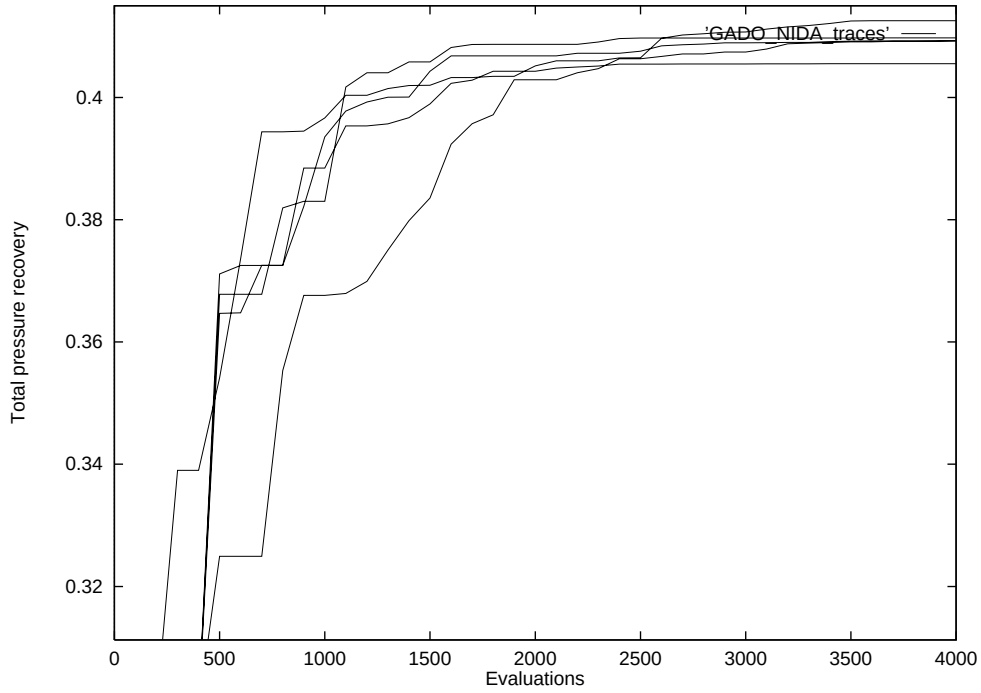

Figure 3.11: Trace of 5 GADO runs in application domain 2 (missile design)

Table 3.4: Comparison of final performance in application domain 2 (missile design)

Method	*Average*	*Worst*	*Best*	*S.D.*
GADO	0.4092	0.4055	0.4125	0.00223
ASA	0.4030	0.3933	0.4123	0.00627
GCOPIII	0.3933	0.3905	0.3997	0.00327
CFSQP	0.3755	0.3691	0.3856	0.00645
RP	0.3688	0.3469	0.3848	0.01382

shown in Figure 3.10. The worst case final total pressure recovery for GADO was 0.405 where as for ASA (the best contender) it was only 0.393.

Figure 3.11-3.15 illustrate the traces of all five runs of the different methods in application domain 2 (missile design). It is clear from these figures that the performance of GADO was more *stable* than the performance of the other methods in the sense that all the GADO runs look the same but the runs of other methods have a large degree of variation. This strongly suggests that GADO is more reliable than the other methods.

Table 3.4 also demonstrates the reliability of GADO. The table presents information about the final performance (the measure of merit at the end of 4000 iterations) of the

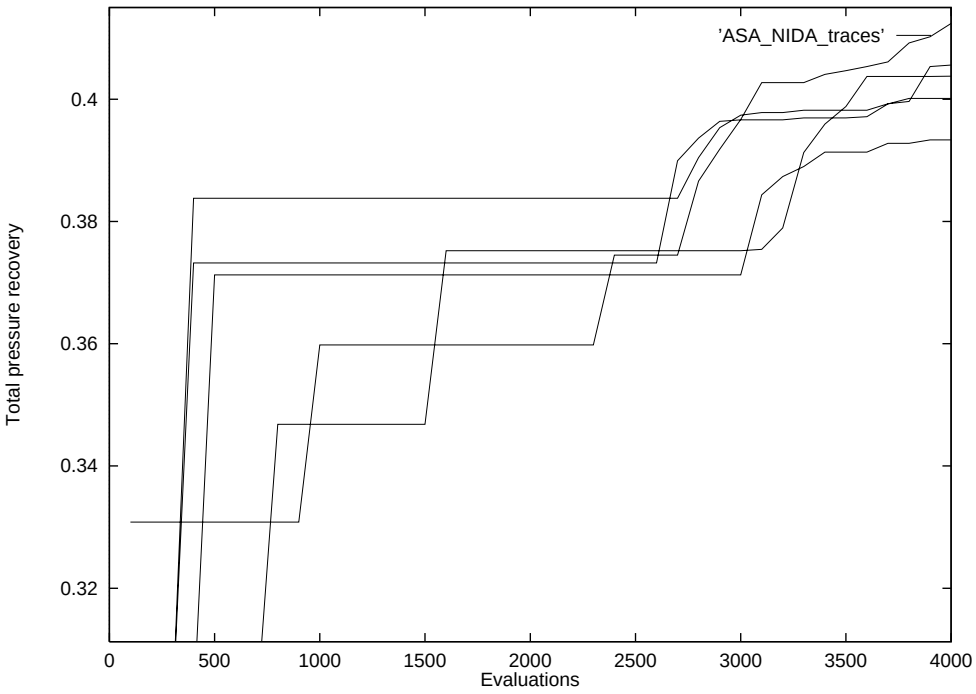

Figure 3.12: Trace of 5 ASA runs in application domain 2 (missile design)

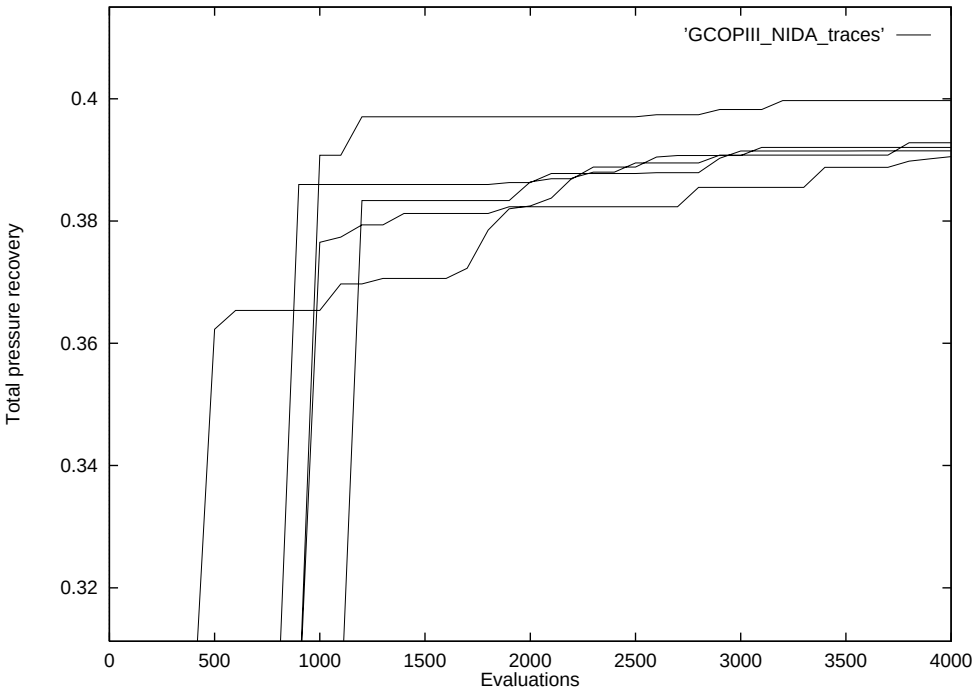

Figure 3.13: Trace of 5 GCOPIII runs in application domain 2 (missile design)

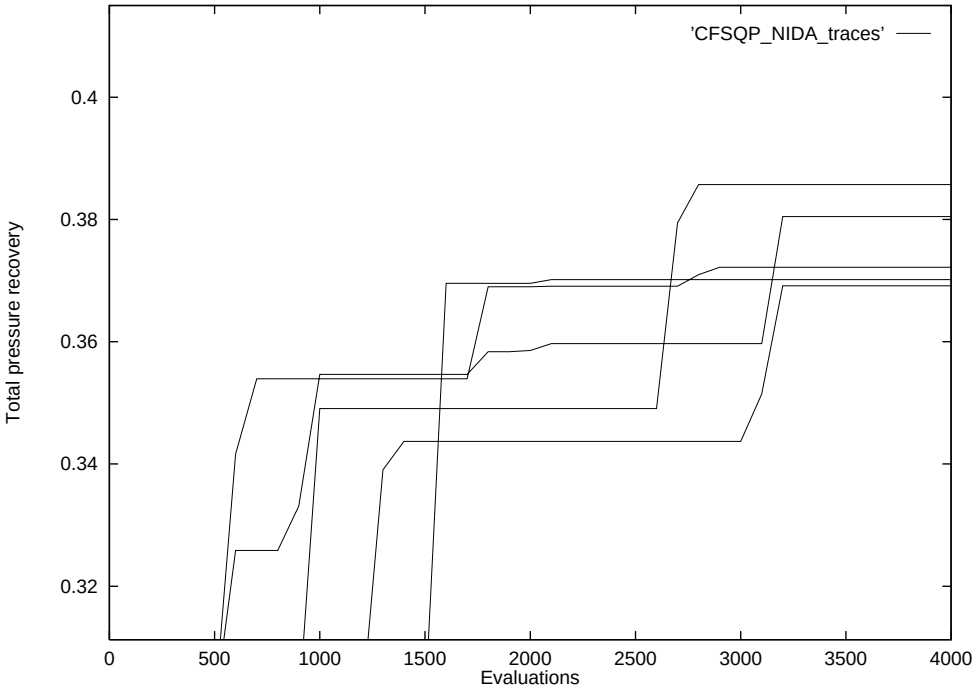

Figure 3.14: Trace of 5 CFSQP runs in application domain 2 (missile design)

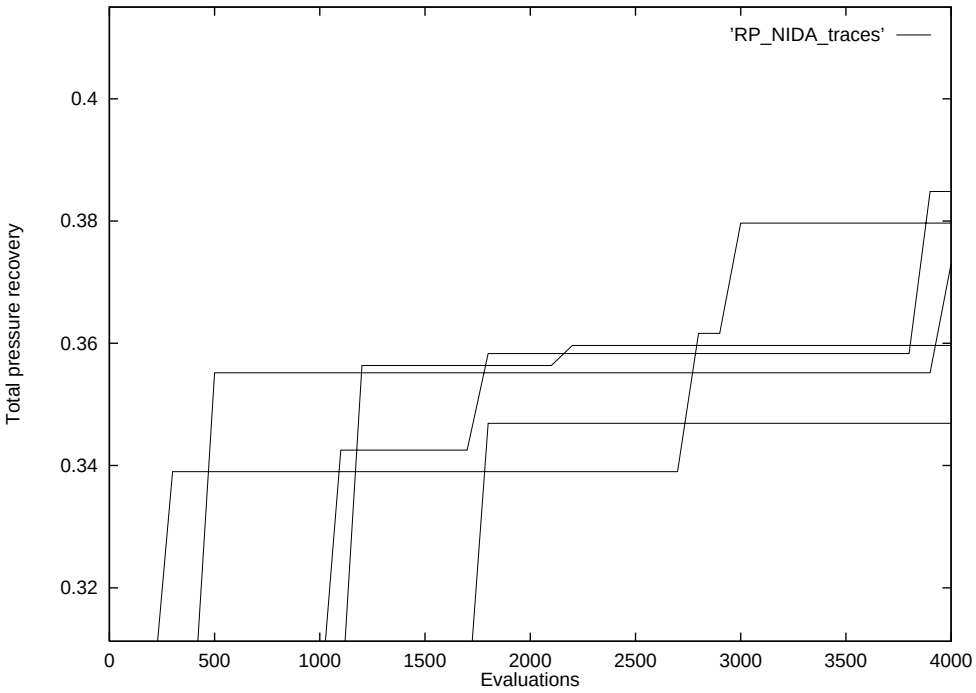

Figure 3.15: Trace of 5 RP runs in application domain 2 (missile design)

5 runs of the different optimization methods in application domain 2 (missile design). Column 2 reports the average final performance, column 3 and 4 report the worst and the best final performance respectively and the last column reports the standard deviation of the final performance. The methods are listed in decreasing value of the average final performance. The table shows that GADO had the best average, best and worst final performance with respect to the other methods. The table also shows that GADO had the lowest standard deviation among the methods.

3.4 Concluding remarks

In this chapter we demonstrated the merit and strength of GADO by comparing its performance to that of four different global optimizers in two different realistic engineering design domains. The optimizers are:

- **Random Probes:** Random Probes generates random points uniformly in the design space and evaluates them.
- **Multi-start CFSQP:** CFSQP is a state-of-the-art implementation of the Sequential Quadratic Programming method, a quasi-Newton method that solves a nonlinear constrained optimization problem by fitting a sequence of quadratic programs. CFSQP is a gradient based local optimizer. It is made into a global optimizer by restarting it from different random points in the search space.
- **GENOCOPIII:** This is a state of the art "off-the-shelf" genetic-algorithm-based global optimizer for constrained continuous optimization.
- **ASA:** This is a state of the art "off-the-shelf" implementation of the simulated annealing paradigm.

We compared the average and worst case performance of these optimizers to those of GADO in the domains of aircraft design and missile inlet design. In the aircraft design domain, the density of feasible points was extremely low. As a result Random Probes, GENOCOPIII and ASA failed to optimize without user intervention. CFSQP

and GADO were able to optimize without user intervention but the performance of GADO was superior to that of CFSQP. In the missile inlet design domain the density of feasible points was very low but not as low as the previous case. As a result all optimizers were able to perform in this domain but again the performance of GADO was clearly superior to that of the other optimizers.

In the next chapter we further investigate the scope of applicability of GADO as well as the sensitivity of its performance to parameter variation and problem structure. We also demonstrate the utility of some of the novel components of GADO in the two domains described in this chapter as well as eight benchmark design optimization domains.

Chapter 4

Scope, Sensitivity and Analysis

In this chapter we analyze the scope of applicability of GADO by testing it in further engineering design domains previously used in the literature. We also investigate the role of some of the key novel components of GADO. We show how these components significantly improved the overall performance in several domains. Moreover, we show that these components never degraded the performance significantly in the domains investigated. Finally, we analyze the effect of parameter variation and search space structure on performance.

4.1 Scope of applicability

In order to further define the scope of applicability of GADO, we examined its performance in a large group of engineering design domains that do not have all the properties targeted by GADO.

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.[1] 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

[1] It should be noted that his study did not include stochastic optimization methods such as genetic algorithms or simulated annealing. This is not surprising because these methods hardly gained any popularity until the late eighties. Genetic algorithms as optimization tools were introduced by Holland in 1975 [Holland 1975] and simulated annealing as a search method was introduced by Kirkpatrick in the early eighties [Kirkpatrick *et al.* 1983].

were also used by other researchers before and after Sandgren's work [Powell and Skolnick 1993]. These problems differ from the problems previously described in Chapter 3 in the following major ways:

- There are no unevaluable points. The objective functions and the constraints are computed via computer programs that will yield a result for any input in the proper range.
- The computation of objective functions and constraints is significantly less expensive than the domains described in Chapter 3. Roughly, they typically take around one third the time the aircraft simulator takes on the same environment. Thus they are about 3 times faster than the aircraft simulator and 100 times faster than the missile design code (NIDA).

Those problems have now become used in engineering design optimization domains as benchmarks. The most recent experiment 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 GADO in eight[2] of these 10 domains and compared its performance to the results reported in [Powell and Skolnick 1993]. All eight were minimization problems. A description of the domains we used is given in Appendix B. Even though these problems are not exactly the kind of engineering design domains that GADO was designed to handle, GADO did extremely well in them nevertheless.

For each problem GADO was run 5 times using different random starting populations and the default values for its parameters (these are the exact values used in the optimizations of the more practical domains described in Chapter 3). The limit on the number of function evaluations was set to 50,000. This setup was the same one used in [Powell and Skolnick 1993] and this allowed comparison of results. The research in

[2]We were unable to do any comparison in 2 of the 10 domains because they had unbounded variables. The researchers did not report the bounds they imposed on the unbounded variables in [Powell and Skolnick 1993] and we were unable to get any clarification from them.

Table 4.1: Results of optimization in benchmark domains

Domain *No.*	Sandgren *No.*	*Dim.*	**Constraints** *inequ.*	*equ.*	best f	OOGA f	NumOpt f	**GADO** f
1	13	5	4	0	26.79	26.79	28.02	26.78
2	2	3	2	0	-3.3	-3.3	-3.3	-3.3
3	3	5	6	0	-3.06	-3.06	-3.06	-3.06
4	8	3	2	0	-5.68	-5.68	-5.68	-5.68
5	6	6	0	4	8.92	(8.93)	9.26	8.97
6	15	16	0	8	244.8	(2412)	244.8	(587.54)
7	21	13	13	0	97.5	134.2	161.1	117.57
8	22	16	19	0	174.7	(537.1)	209.5	214.51

[Powell and Skolnick 1993] only reported the worst case performance (i.e. the worst of the 5 runs) of OOGA so we also report the worst case performance of GADO for fair comparison. The research in [Powell and Skolnick 1993] was not clear about the experiments with NumOpt. It mentioned however that a limit of 50,000 evaluations was imposed on NumOpt and reported one number for each problem as its performance measure so we used it for comparison.

The results are summarized in Table 4.1. 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 (excluding the findings of this thesis). The seventh column shows the performance of OOGA reported in [Powell and Skolnick 1993]. The eighth column shows the performance of NumOpt reported in [Powell and Skolnick 1993]. Finally, the last column shows the worst case final performance of GADO. Numbers in brackets represent infeasible points. The table demonstrates the following facts:

- All three optimizers reached the best known optimum in three of the eight domains (2,3,4).
- GADO outperformed OOGA in domains (1,5,7,8).

Table 4.2: Results of optimization in benchmark domains (without screening)

Domain *No.*	Sandgren *No.*	*Dim.*	**Constraints** *inequ.*	*equ.*	best f	OOGA f	NumOpt f	**GADO** f
1	13	5	4	0	26.79	26.79	28.02	26.78
2	2	3	2	0	-3.3	-3.3	-3.3	-3.3
3	3	5	6	0	-3.06	-3.06	-3.06	-3.06
4	8	3	2	0	-5.68	-5.68	-5.68	-5.68
5	6	6	0	4	8.92	(8.93)	9.26	8.97
6	15	16	0	8	244.8	(2412)	244.8	(802.75)
7	21	13	13	0	97.5	134.2	161.1	125.36
8	22	16	19	0	174.7	(537.1)	209.5	242.7

- GADO outperformed NumOpt in three domains (1,5,7) but NumOpt outperformed GADO in two domains (6,8).

Since the evaluation functions were not expensive enough to justify the use of the screening module (the total time spent on screening in some cases exceeded the total time spent running the evaluation code) we felt obligated to repeat the experiments with the screening module turned off. All the other parameters were kept at their default values. The overhead for the GA bookkeeping in this setup was no more than 10% of the total CPU time and thus we believe the comparison to OOGA and NumOpt should be fair. The results are summarized in Table 4.2. GADO's performance was slightly worse in some cases - as expected in the absence of the screening module - but the results were qualitatively the same as with the screening module.

The reason why GADO did not do as well in benchmark domain 6 (Sandgren's problem 15) as it did in the other domains is the fact that there were eight nonlinear equality constraints. This is a very difficult situation for any algorithm using a penalty approach to constraint satisfaction for the following reasons:

- The penalty coefficient has to increase tremendously to enforce feasibility. Thus, once a feasible (or near feasible) point is found it becomes extremely hard to explore other regions of the space.

- If the constraints are highly nonlinear (in this problem they were products of

cubic functions) it becomes extremely hard to maintain the satisfiability of one constraint while trying to satisfy others. In other words, it is hard for a penalty based method to walk on the nonlinear constraint boundary. Therefore the algorithm must satisfy all constraints at once which is extremely hard.

NumOpt on the other hand uses a direct approach to satisfy and maintain satisfiability of constraints, using constraint gradients.

Nevertheless, by changing the values of four parameters of GADO from their defaults, we were able to tremendously improve the performance. Even though the performance was still inferior to that of NumOpt in this problem, all five runs found feasible points and the worst run had a measure of merit value of 300.

For the remaining experiments in this chapter we decided to limit our attention to the two application domains (the aircraft design and the missile design) and benchmark domain 7 (Sandgren's problem 21). We dropped the other benchmarks from consideration because:

- Benchmarks 1, 2, 3 and 4 turned out to be extremely easy for GADO (the global optimum was reached in a few hundred to at most few thousand iterations!). We doubt that the component contribution or effect of parameter variation will be noticeable in these domains.

- Benchmarks 5 and 6 had all equality and no inequality constraints at all. This is a very special case, considering that in all the modern practical design optimization domains we considered, there were no equality constraints.

- Benchmark 8 is basically a more complex version of benchmark 7. The behavior of these two benchmarks is expected to be qualitatively the same.

4.2 Analysis of component contributions

In this section we demonstrate the utility of some of the key components of GADO. We show how these components improved the performance in several domains. The results also show that these components never degraded the performance significantly

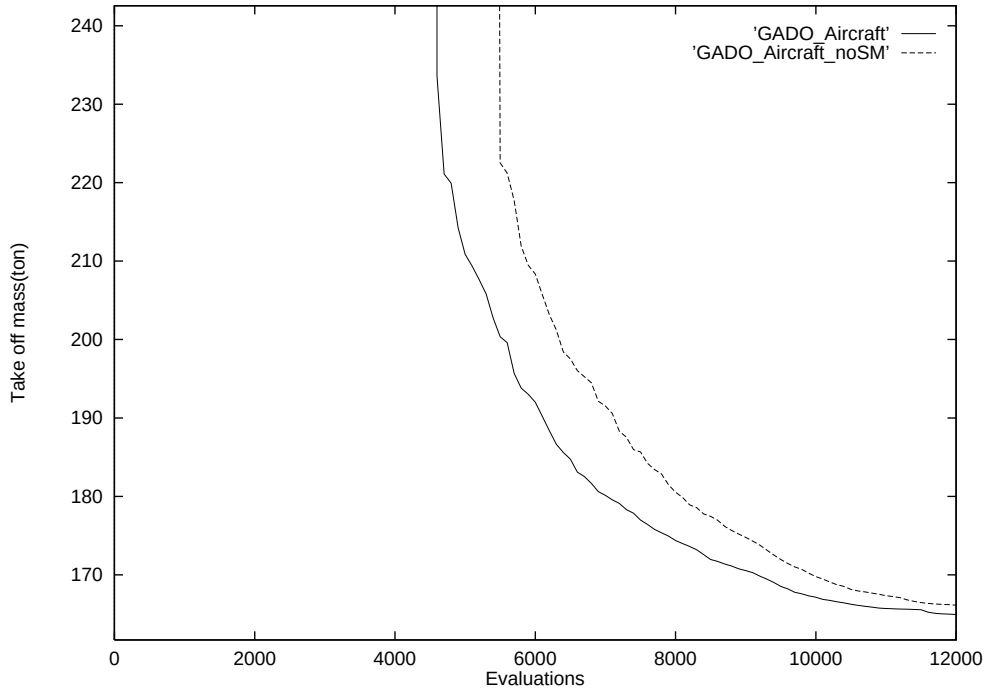

Figure 4.1: Effect of the screening module on average performance in application domain 1 (aircraft design)

in the domains investigated. We conducted the experiments by running GADO without one component at a time and comparing the performance with that of the default parameters (which uses all components).

4.2.1 Utility of the screening module

To demonstrate the utility of the screening module we turned it off and kept all other parameters intact. We compared the performance of GADO with this setup to its performance with the default parameters in several domains.

Figure 4.1 demonstrates the utility of the screening module in domain 1 (aircraft design). The figure shows the average of 15 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 almost vertical leading parts of the curves are where the feasible region was reached). Figure 4.2 shows the effect of the screening module on worst case performance. The figure is qualitatively

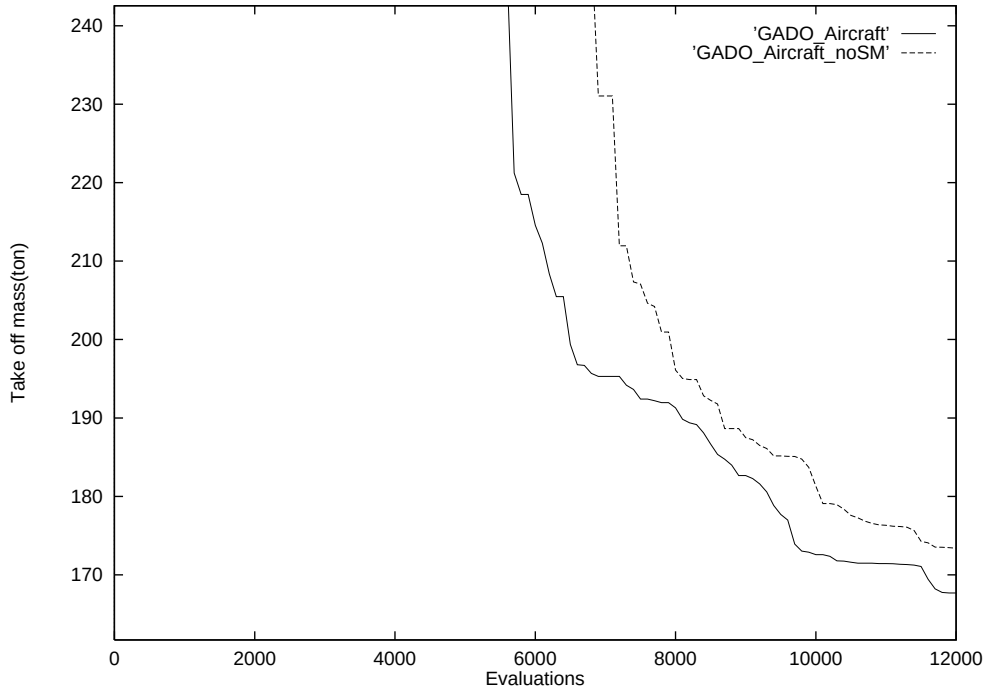

Figure 4.2: Effect of the screening module on worst case performance in application domain 1 (aircraft design)

the same as Figure 4.1.

Figure 4.3 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 and without the screening module. All other parameters were kept at their default values. The screening module had very little effect on performance in this domain. Figure 4.4 shows the effect of the screening module on worst case performance. The figure is qualitatively the same as Figure 4.3.

Figure 4.5 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 screening module. Figure 4.6 shows the effect of the screening module on worst case performance. The figure is qualitatively the same as Figure 4.5.

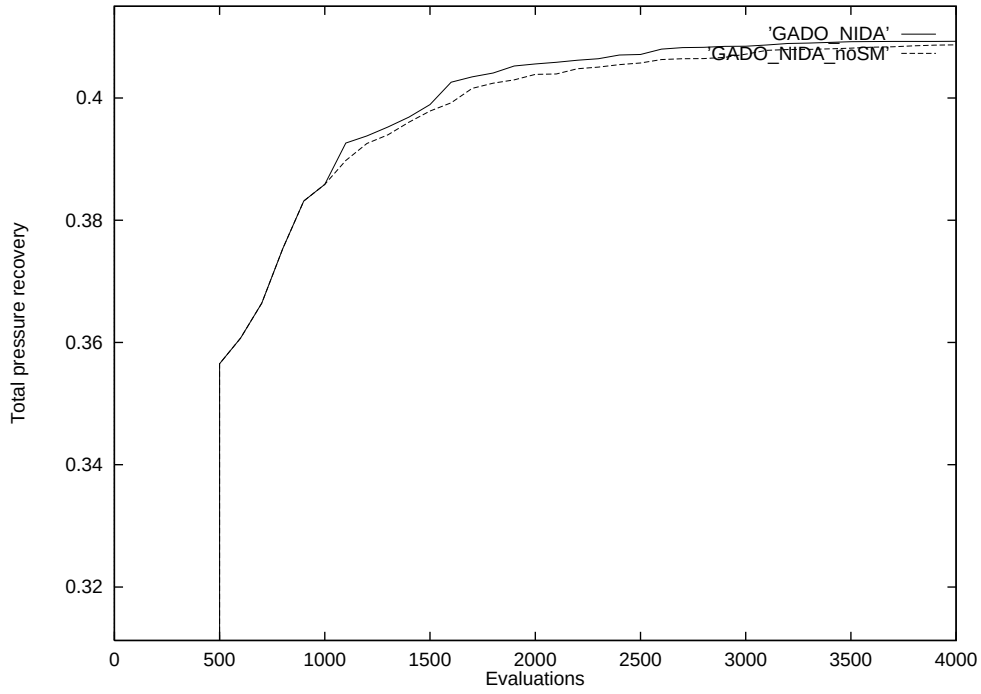

Figure 4.3: Effect of the screening module on average performance in application domain 2 (missile design)

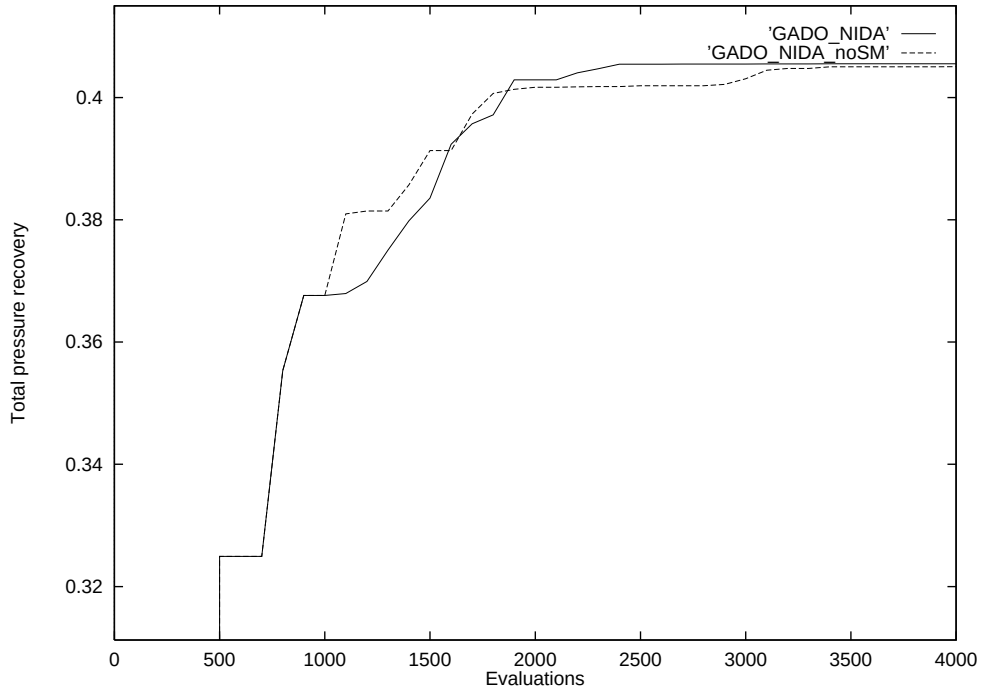

Figure 4.4: Effect of the screening module on worst case performance in application domain 2 (missile design)

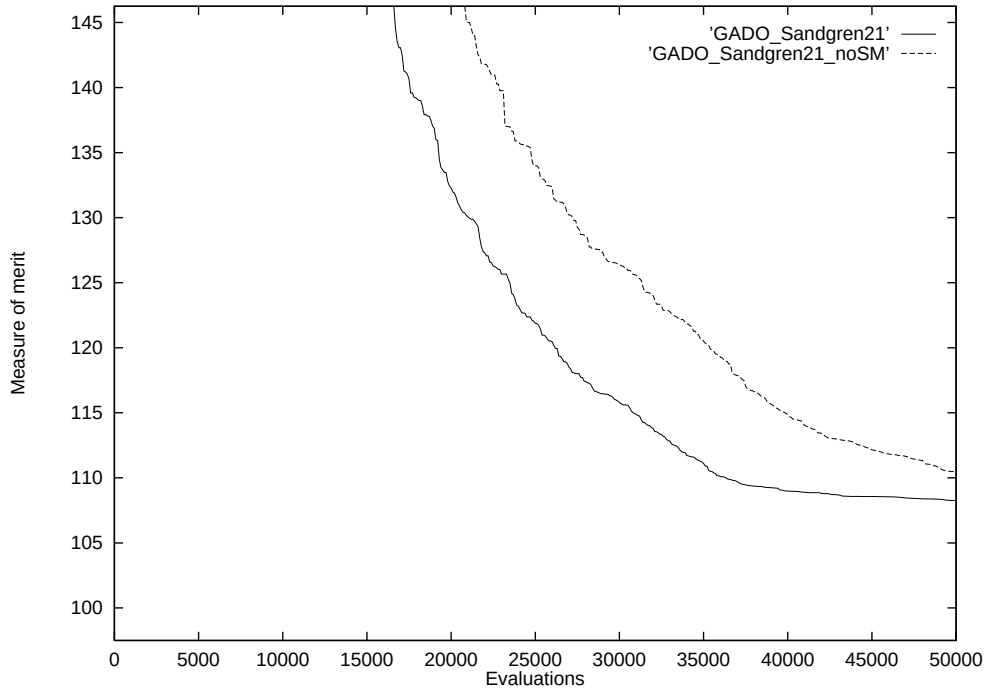

Figure 4.5: Effect of the screening module on average performance in benchmark domain 7 (Sandgren's problem 21)

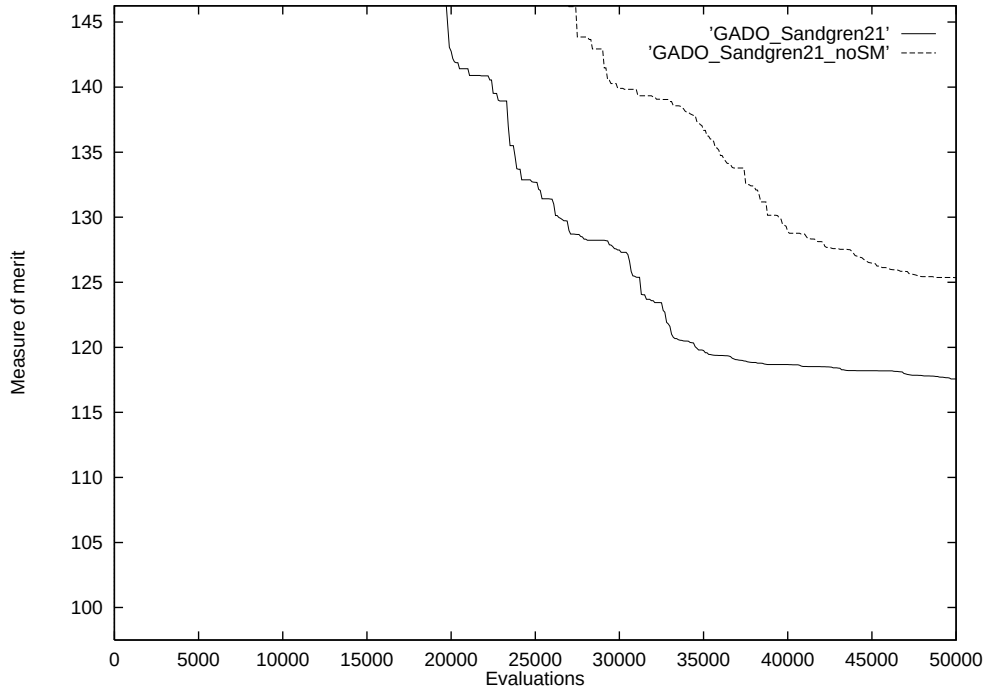

Figure 4.6: Effect of the screening module on worst case performance in benchmark domain 7 (Sandgren's problem 21)

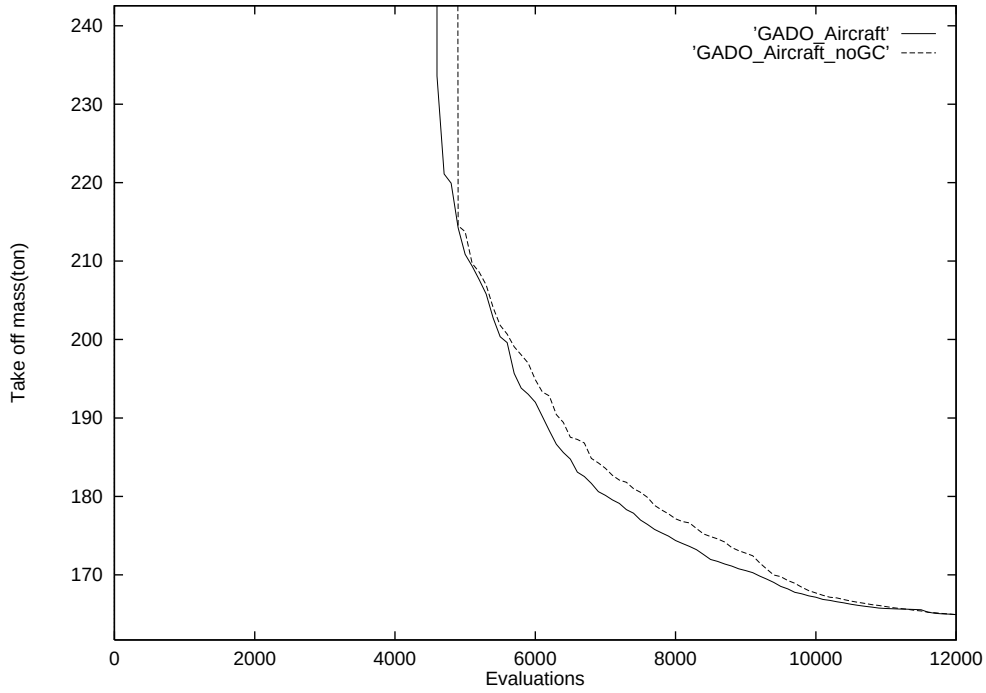

Figure 4.7: Effect of guided crossover on average performance in application domain 1 (aircraft design)

4.2.2 Utility of guided crossover

To demonstrate the utility of guided crossover we changed the parameters so that it is never used and kept all other parameters intact. We compared the performance of GADO with this setup to its performance with the default parameters in several domains.

Figure 4.7 demonstrates the utility of guided crossover in application domain 1 (aircraft design). The figure shows the average of 15 runs of GADO both with and without guided crossover. All other parameters were kept at their default values. The figure shows that guided crossover improved the performance in all stages of the search, although the improvement was not very large. The feasible region was reached faster with guided crossover. It is noted however that the performance at the end of the search was almost identical. Figure 4.8 shows the effect of guided crossover on worst case performance. The figure shows that the effect of guided crossover on worst case performance was minimal.

Figure 4.9 demonstrates the utility of guided crossover in application domain 2

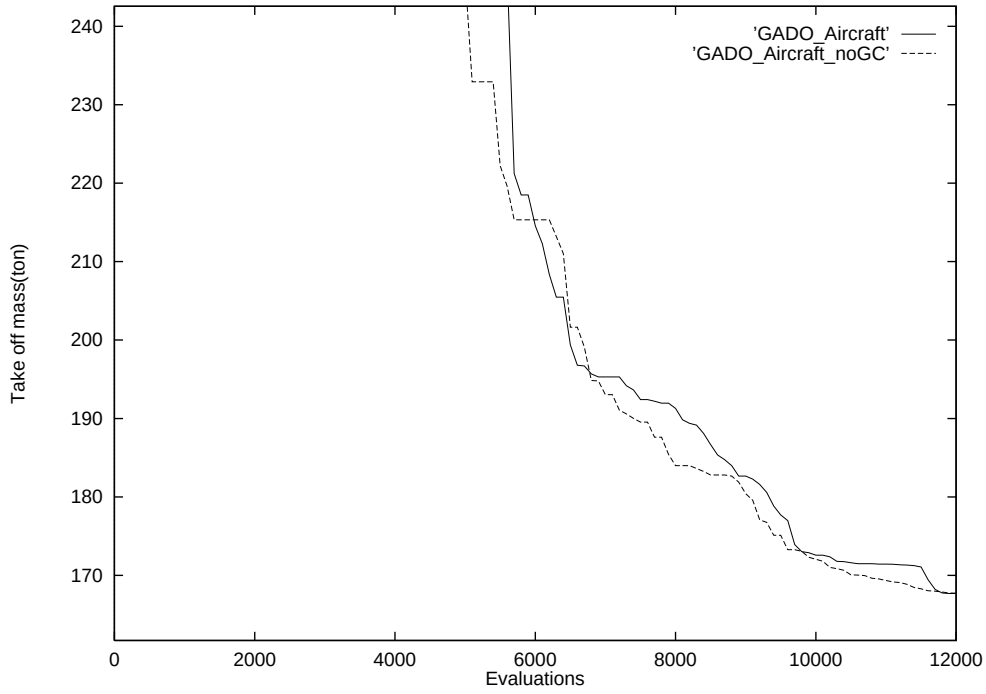

Figure 4.8: Effect of guided crossover on worst case performance in application domain 1 (aircraft design)

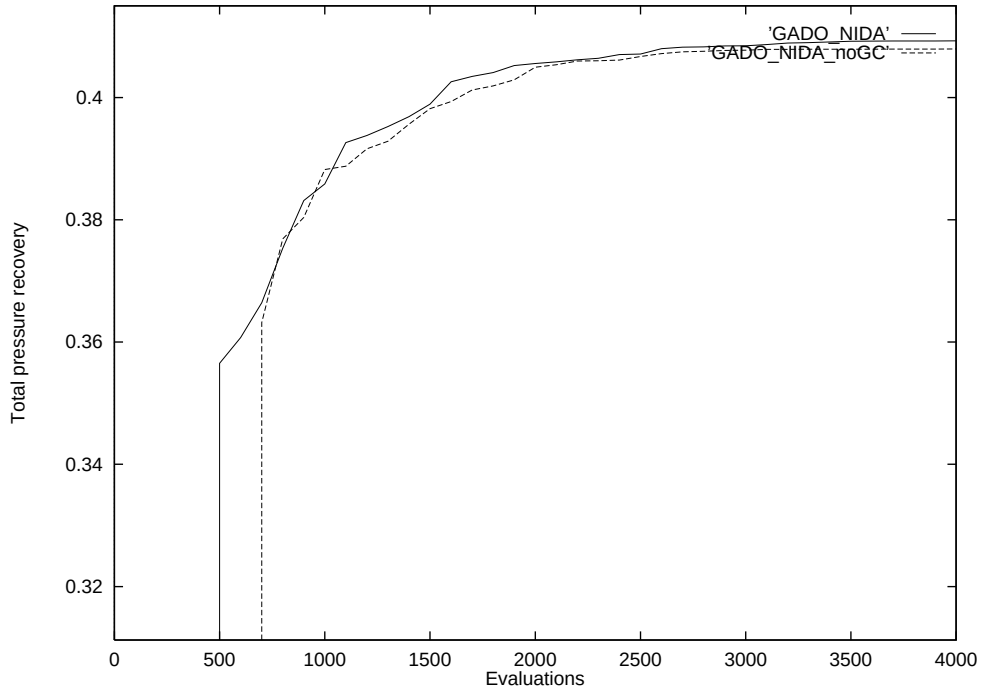

Figure 4.9: Effect of guided crossover on average performance in application domain 2 (missile design)

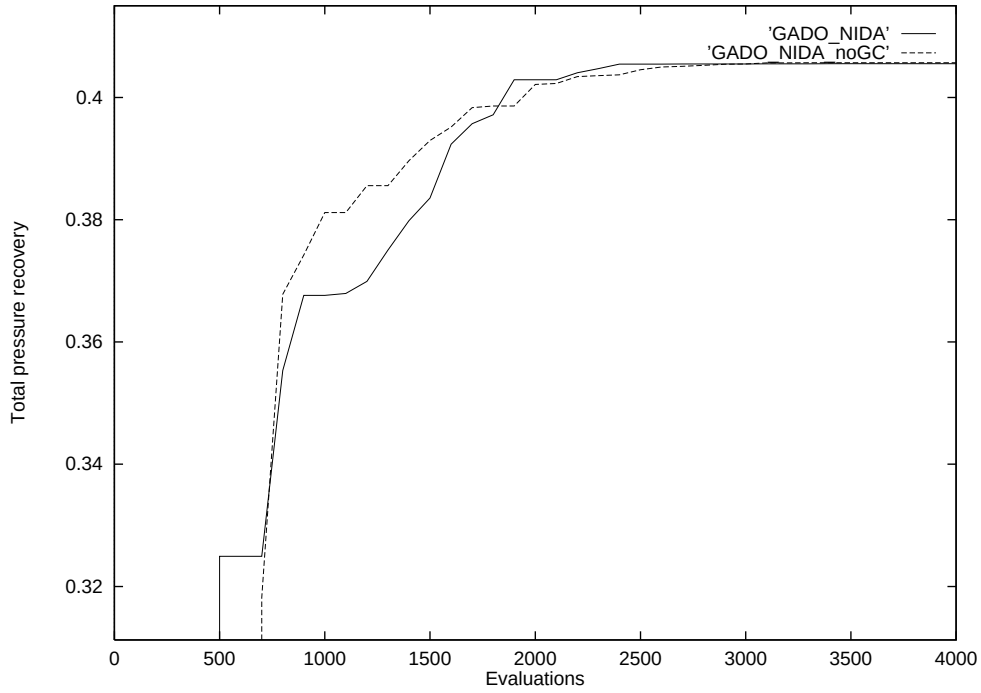

Figure 4.10: Effect of guided crossover on worst case performance in application domain 2 (missile design)

(missile design). The figure shows the average of 5 runs of GADO both with and without guided crossover. All other parameters were kept at their default values. The figure shows that guided crossover improved the performance in most stages of the search, although the improvement was not very large.[3] The feasible region was reached faster with guided crossover. Figure 4.10 shows the effect of guided crossover on worst case performance. The figure shows that the effect of guided crossover on worst case performance was minimal.

Figure 4.11 demonstrates the utility of guided crossover in benchmark domain 7 (Sandgren's problem 21). The figure shows the average of 5 runs of GADO both with and without guided crossover. All other parameters were kept at their default values. The figure shows that guided crossover improved the performance in all stages of the search. The feasible region was reached faster with guided crossover. The final performance at the end of the search was significantly better with guided crossover.

[3]It should be noted that the numerical accuracy of the simulator (NIDA) is about 5%. Thus, from the engineering point of view, the two curves in Figure 4.9 are indistinguishable in the region beyond 700 iterations.

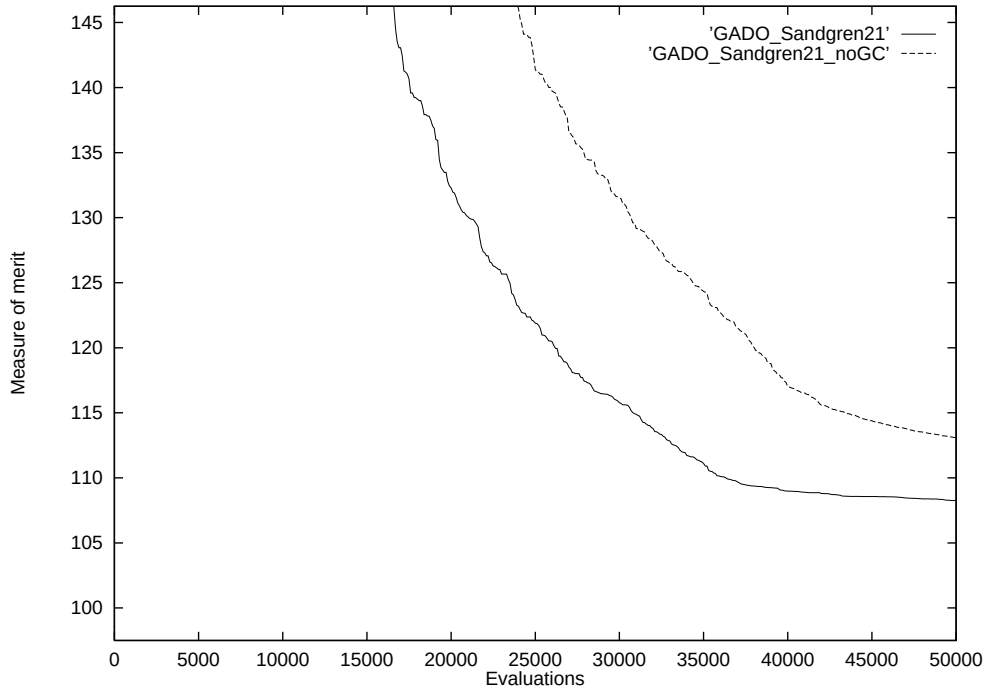

Figure 4.11: Effect of guided crossover on average performance in benchmark domain 7 (Sandgren's problem 21)

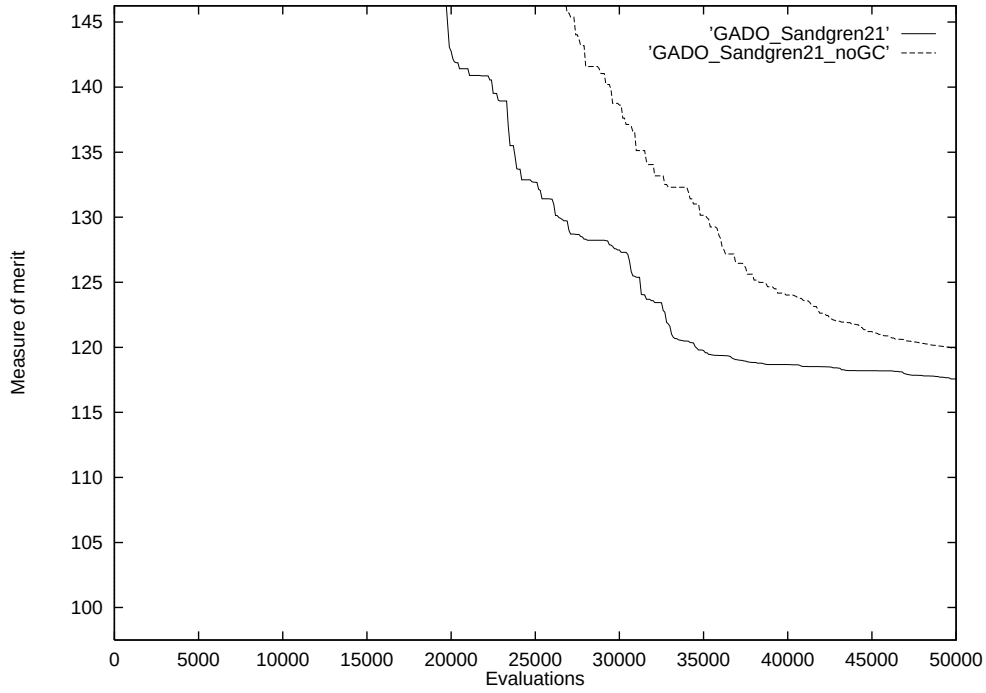

Figure 4.12: Effect of guided crossover on worst case performance in benchmark domain 7 (Sandgren's problem 21)

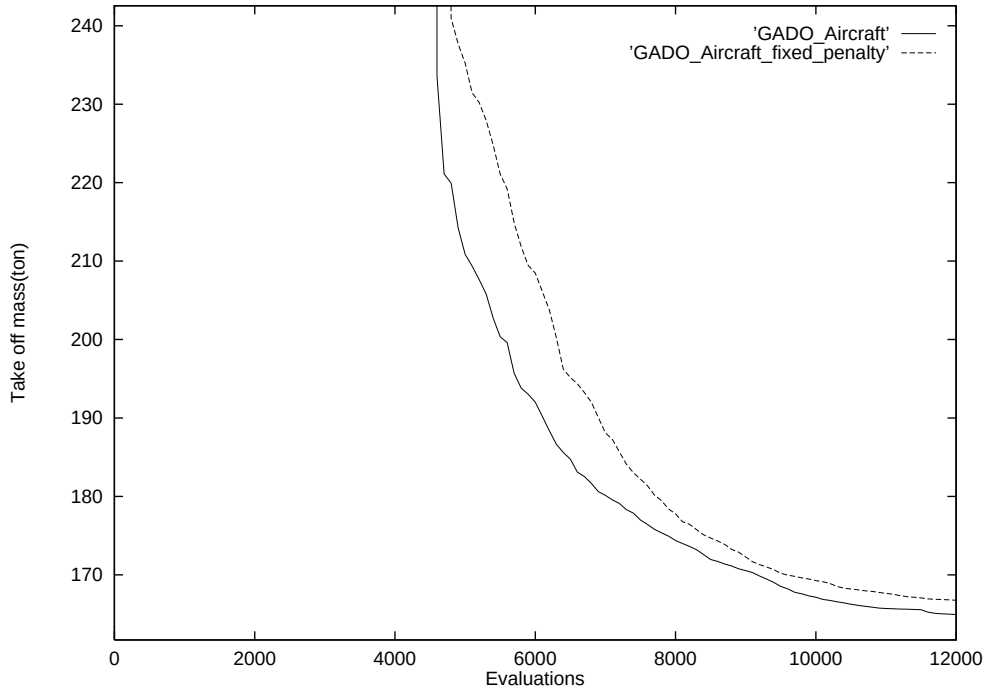

Figure 4.13: Effect of dynamic penalty on average performance in application domain 1 (aircraft design)

Figure 4.12 shows the effect of guided crossover on worst case performance. The figure is qualitatively the same as Figure 4.11.

4.2.3 Utility of the dynamic penalty method

To demonstrate the utility of the dynamic penalty method, we changed the parameters so that GADO uses a fixed penalty coefficient of 10^6 times the default initial penalty coefficient and kept all other parameters intact. The coefficient had to be so large to insure that the feasible region could be reached in all domains. We compared the performance of GADO with this setup to its performance with the default parameters.[4]

Figure 4.13 demonstrates the utility of the dynamic penalty method in application domain 1 (aircraft design). The figure shows the average of 15 runs of GADO once with the default parameters (which implies using the dynamic penalty method) and another time with the large fixed coefficient. All other parameters were kept at their default

[4]We also tried using a small fixed penalty coefficient but this failed in several domains as there were infeasible regions with objective values that are much better than the global optimum.

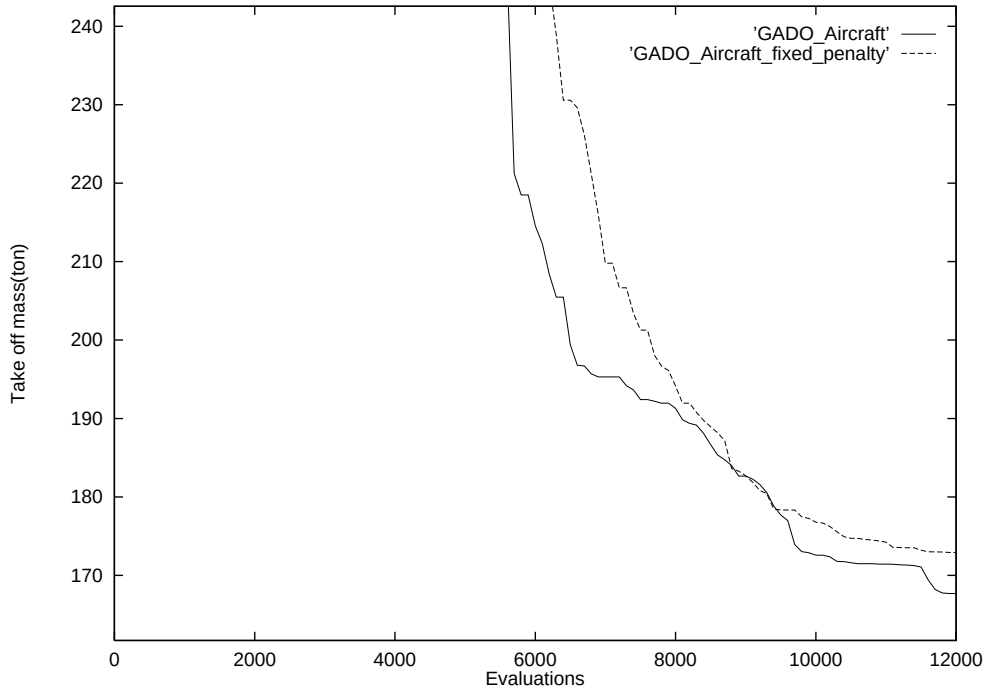

Figure 4.14: Effect of dynamic penalty on worst case performance in application domain 1 (aircraft design)

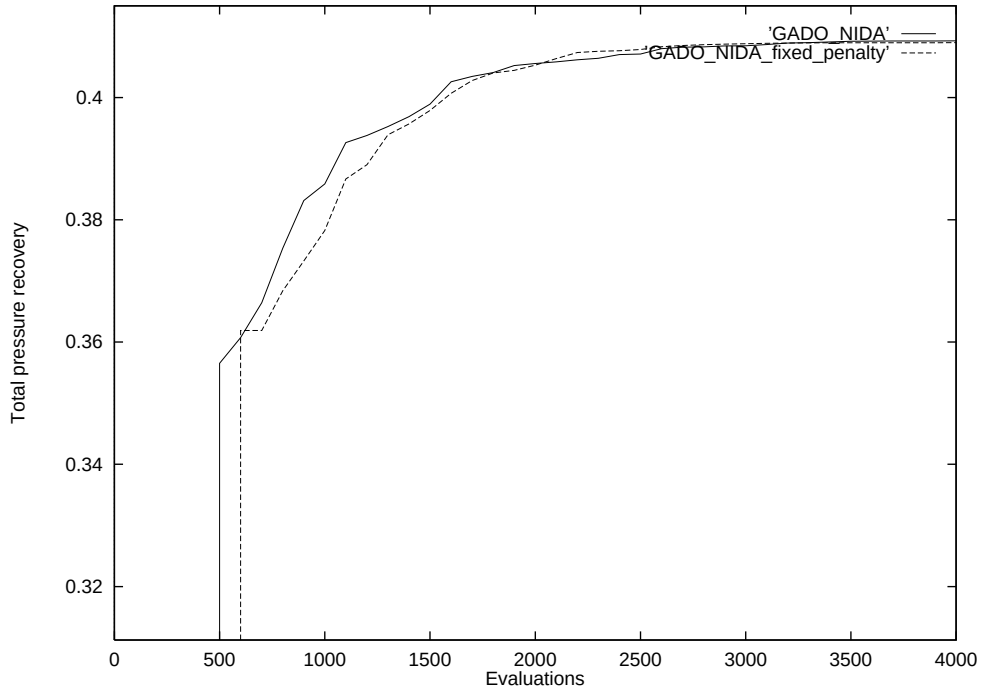

Figure 4.15: Effect of dynamic penalty on average performance in application domain 2 (missile design)

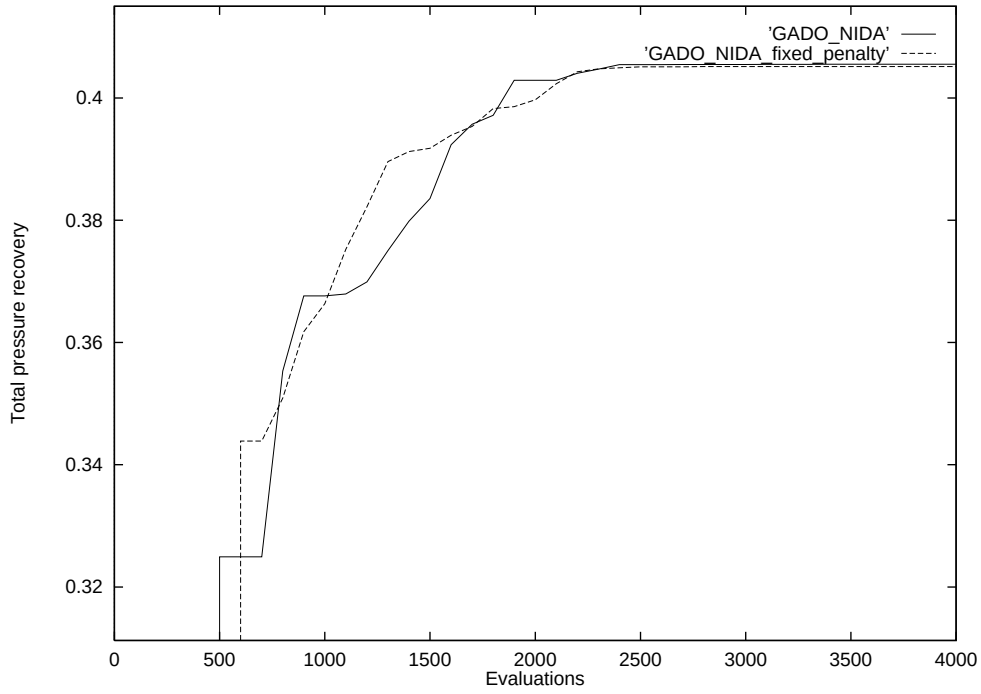

Figure 4.16: Effect of dynamic penalty on worst case performance in application domain 2 (missile design)

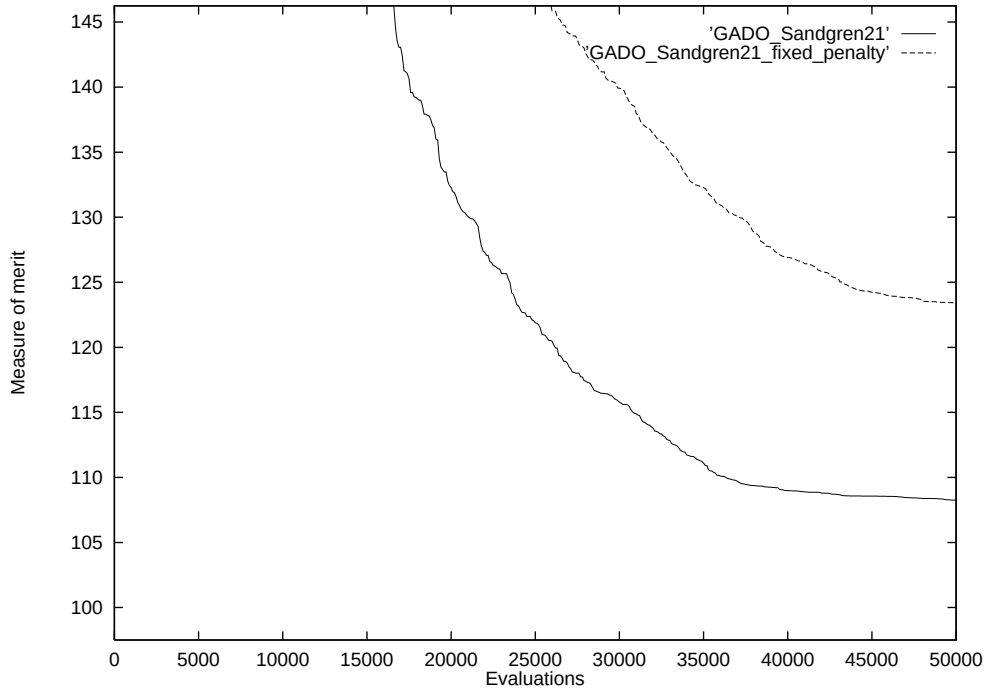

Figure 4.17: Effect of dynamic penalty on average performance in benchmark domain 7 (Sandgren's problem 21)

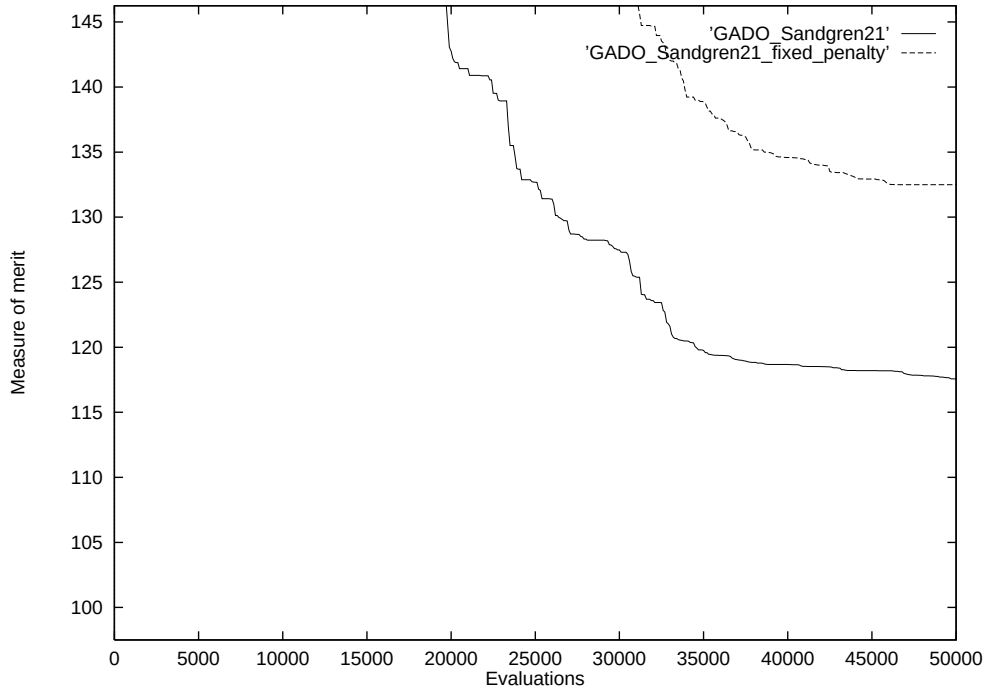

Figure 4.18: Effect of dynamic penalty on worst case performance in benchmark domain 7 (Sandgren's problem 21)

values. The figure shows that the dynamic penalty method improved the performance in all stages of the search. This should be expected since the dynamic penalty approach allows for better exploration in the beginning of the search. The fact that the penalty coefficient is actually allowed to decrease towards the end of the search – to allow for better examination of the constraint borders – also helps. Figure 4.14 shows the effect of the dynamic penalty method on worst case performance. The figure is qualitatively the same as Figure 4.13.

Figure 4.15 demonstrates the effect of using the dynamic penalty method in application domain 2 (missile design). The figure shows the average of 5 runs of GADO once with the default parameters (which implies using the dynamic penalty method) and another time with the large fixed coefficient. All other parameters were kept at their default values. The figure shows that the effect was minimal in this domain. The same can be said about Figure 4.16 which shows the effect of the dynamic penalty method on worst case performance.

Figure 4.17 demonstrates the utility of the dynamic penalty method in benchmark

domain 7 (Sandgren's problem 21). The figure shows the average of 5 runs of GADO once with the default parameters (which implies using the dynamic penalty method) and another time with the large fixed coefficient. All other parameters were kept at their default values. The figure shows that the dynamic penalty method substantially improved the performance in all stages of the search. Figure 4.18 shows the effect of the dynamic penalty method on worst case performance. The figure is qualitatively the same as Figure 4.17.

4.3 Sensitivity analysis

The goal of the experiments reported in this section was to investigate the effect of parameter variation and problem structure on GADO. This study is important for gaining a better understanding of the limitations of GADO as well as its degree of robustness and stability. The experiments demonstrated a great deal of stability.

4.3.1 Effect of parameter variation

In each of these experiments we varied one or two parameters from their default value(s) and compared the average performance. All other parameters were kept at their default values. We concentrated on the average performance and dropped the worst case performance because the previous experiments suggested they usually give the same results. Note that Appendix A lists all the performance parameters and their default values.

Effect of the population size

It is well known that smaller populations converge faster [Goldberg 1989]. However, they have a higher tendency to converge to a local sub-optimum. Larger populations do more exploration and have a better chance of finding the global optimum. However, they need more time to converge. Our default value for the population size is 10 times the dimension of the space. Our intuition is that this is an average value, that is likely to strike a good balance between reliability and efficiency.

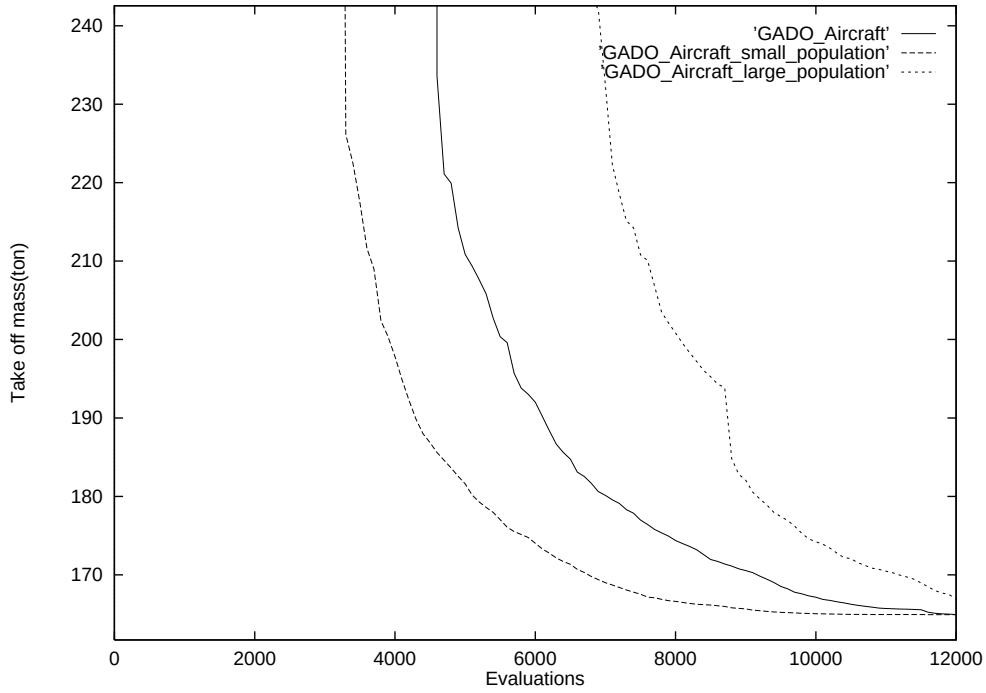

Figure 4.19: Effect of the population size in application domain 1 (aircraft design)

In the first experiment we compared the average performance of 15 runs of GADO in application domain 1 (aircraft design) with three different population sizes:

- Using the default value of 120.
- Using a relatively large population of 240.
- Using a relatively small population of 60.

The results are shown in Figure 4.19. The figure shows that the smaller population was the best in this domain. The final performance, however, was the same in all three settings.

We repeated the experiment in application domain 2 (missile design). The results are shown in Figure 4.20 and are very similar to the results in the previous domain.

We also repeated the experiment in benchmark domain 7 (Sandgren's problem 21). The results are shown in Figure 4.21. In this domain, the smaller population did get a little too greedy, despite its initial advantage.

We conclude that the population size has a considerable effect on performance. Our

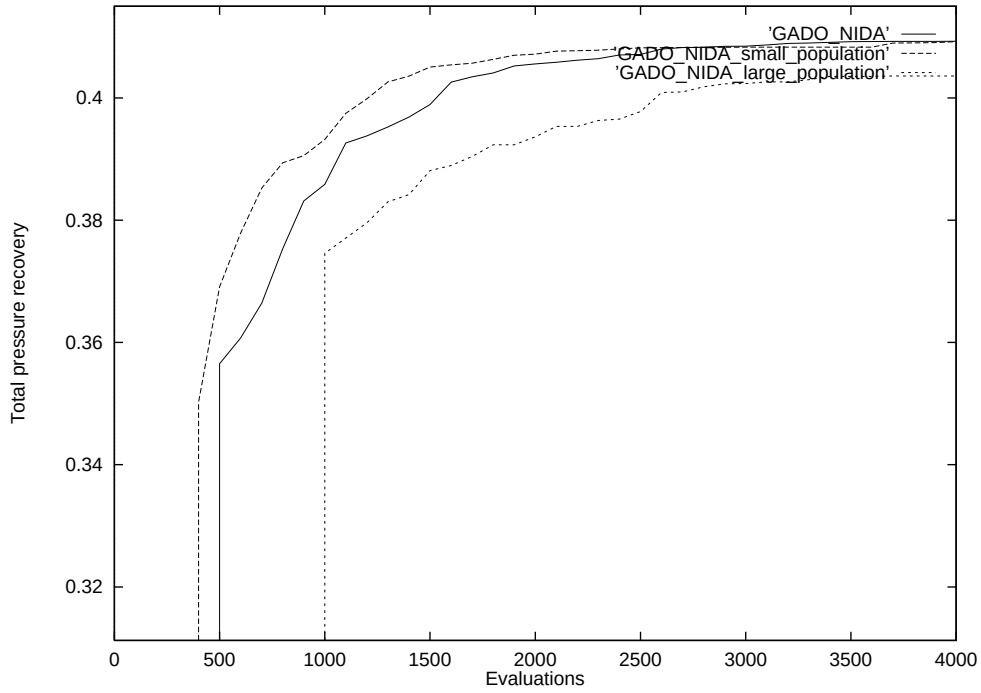

Figure 4.20: Effect of the population size in application domain 2 (missile design)

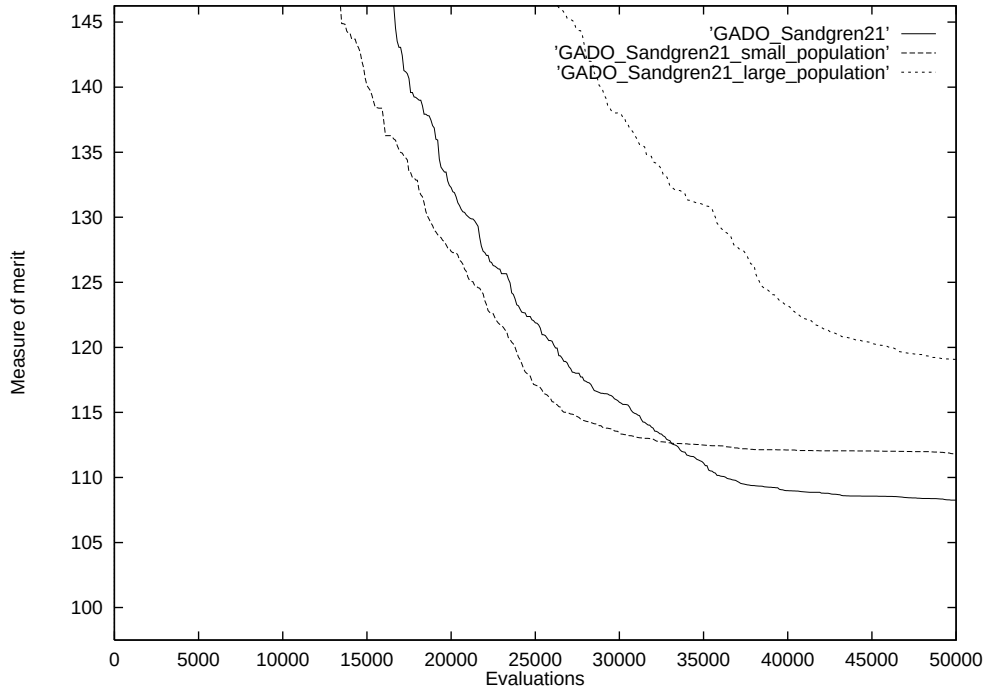

Figure 4.21: Effect of the population size in benchmark domain 7 (Sandgren's problem 21)

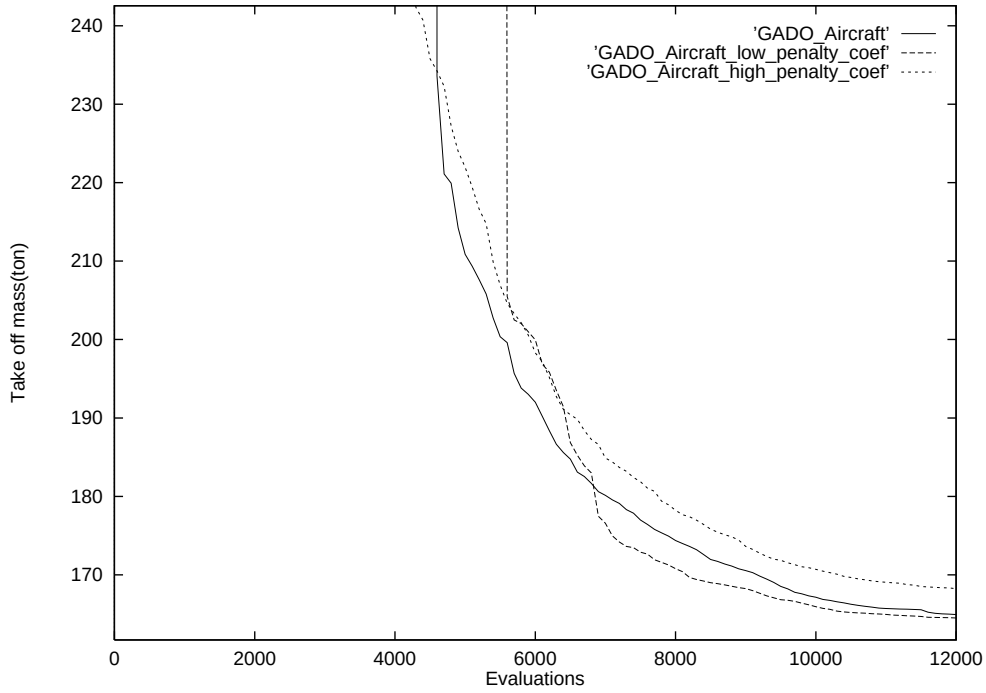

Figure 4.22: Effect of the initial penalty coefficient in application domain 1 (aircraft design)

choice of the default value is reasonable because it gave the best final performance in all three domains.

Effect of the initial penalty coefficient

In the first experiment we compared the average performance of 15 runs of GADO in application domain 1 (aircraft design) with three different values of the initial penalty coefficient:

- Using the default value of 1.
- Using a relatively high value of 1000.
- Using a relatively low value of 0.001.

The results are shown in Figure 4.22. The figure shows that a higher initial penalty coefficient yields an initial advantage followed by a degradation of performance in the late stages of the search. The reason for this is that the high coefficient pushes the

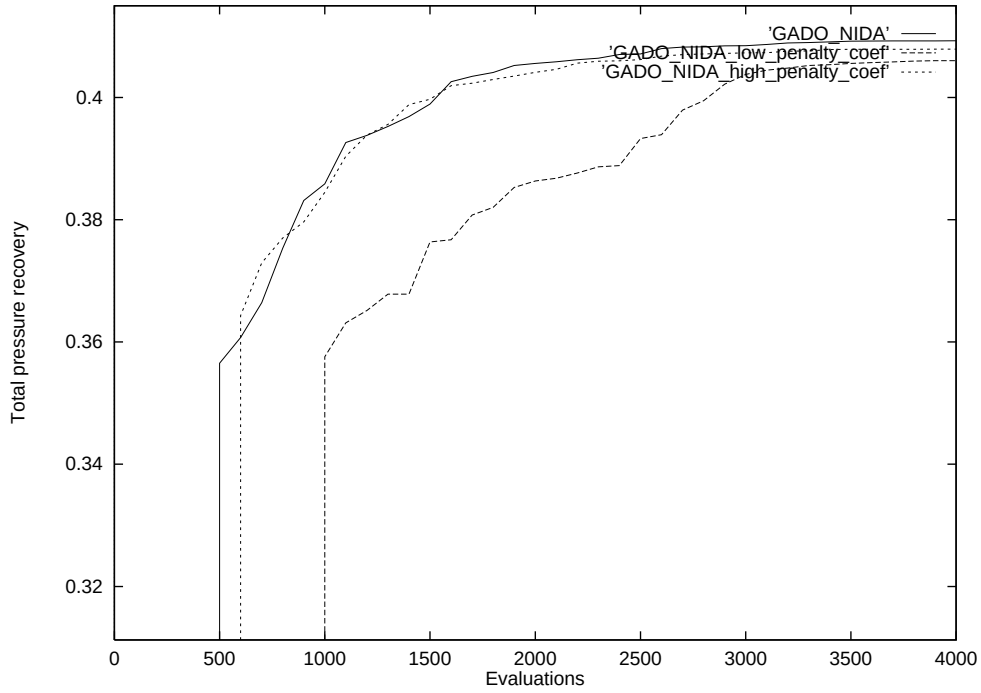

Figure 4.23: Effect of initial penalty coefficient in application domain 2 (missile design)

search strongly towards the feasible region, at the expense of initial sampling of the space. The final performance is likely to suffer as the search may be more difficult if the feasible region was found in a sub-optimal part than if it was found in a region close to the global optimum had the search been less hasty about achieving feasibility. On the other hand, a low initial penalty coefficient will slow down the progress initially to do more sampling of the space but it pays towards the end of the search. The performance in the late stages of the search is better in this case. The figure also shows that the final performance, however, was not significantly different in all three cases.

In the second experiment we repeated the above experiment in application domain 2 (missile design). The results are shown in Figure 4.23. In this case the performance with the default and with the high initial penalty coefficients were very similar. The performance with the low initial penalty coefficient was inferior in all stages of the search. We believe that the low iteration limit used in this domain (4000 iterations) prevented the penalty coefficient from getting high enough to drop the infeasible regions from consideration and concentrate on the feasible regions soon enough. The final performance, however, was not significantly different in all three cases.

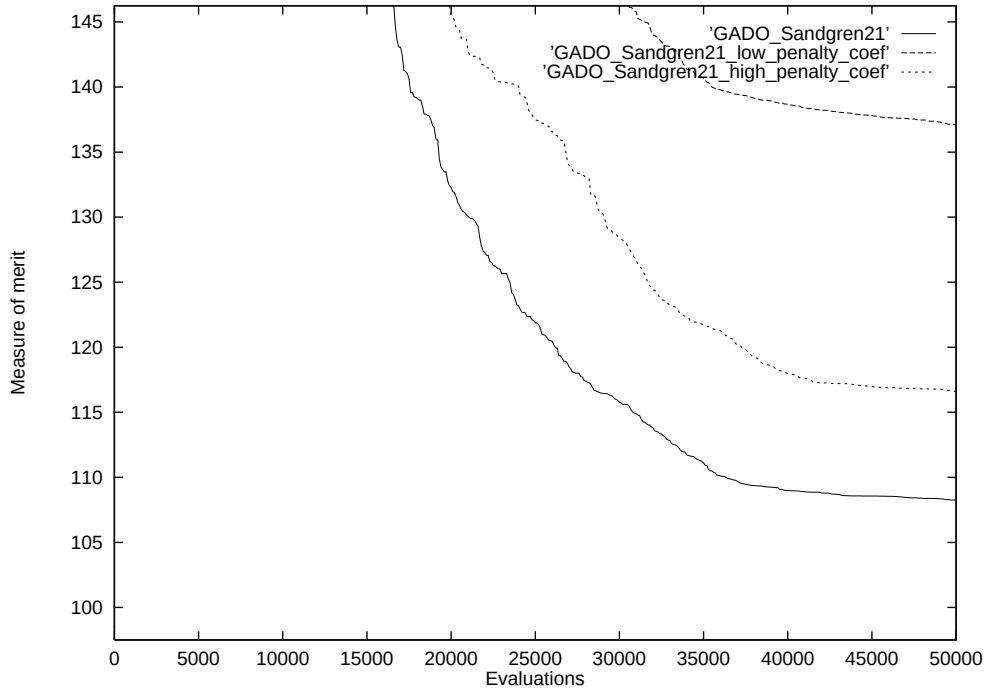

Figure 4.24: Effect of the initial penalty coefficient in benchmark domain 7 (Sandgren's problem 21)

We also repeated the experiment in benchmark domain 7 (Sandgren's problem 21). The results are shown in Figure 4.24. The default value gave the best result, and a considerable degradation of performance happened when it was changed (especially with the low value).

We conclude that our choice of the default value is reasonable.

Effect of the crowding factors

As mentioned in Chapter 2, the crowding factors affect the level of diversity maintained in the GA population. The default it to have an initial crowding factor of one and a final crowding factor of zero. This shifts the replacement strategy from maximal crowding to minimal crowding (replacement of the worst). In the following set of experiments we compared our default strategy with the two pure strategies (maximal and minimal crowding). More precisely, we ran the experiments and compared performance with three setups:

- *Initial crowding factor*=1 and *final crowding factor*=0 (the default).

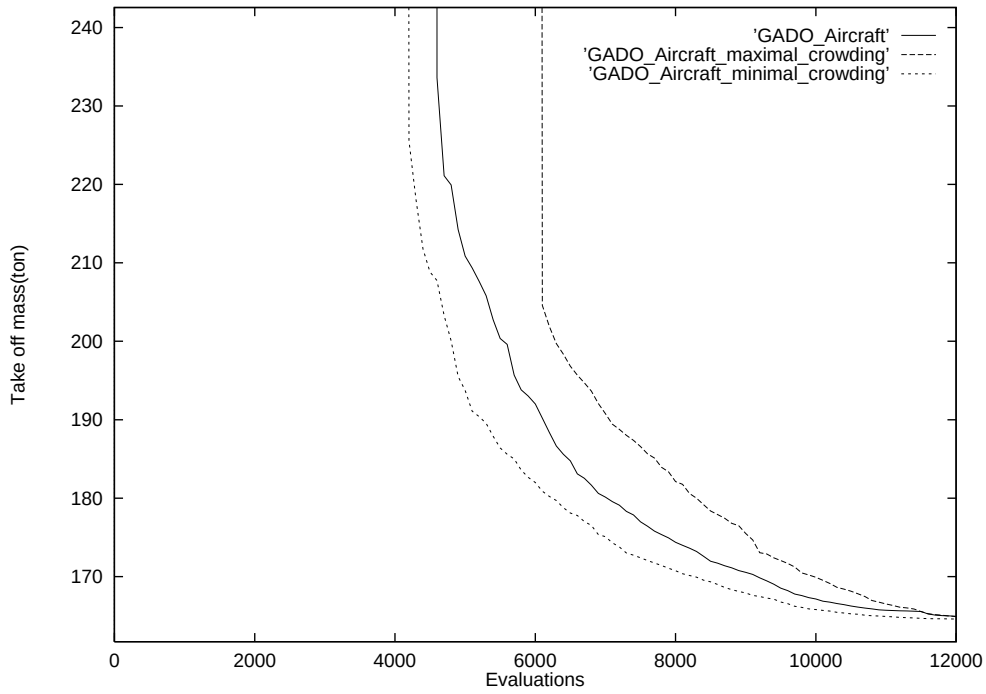

Figure 4.25: Effect of the crowding factors in application domain 1 (aircraft design)

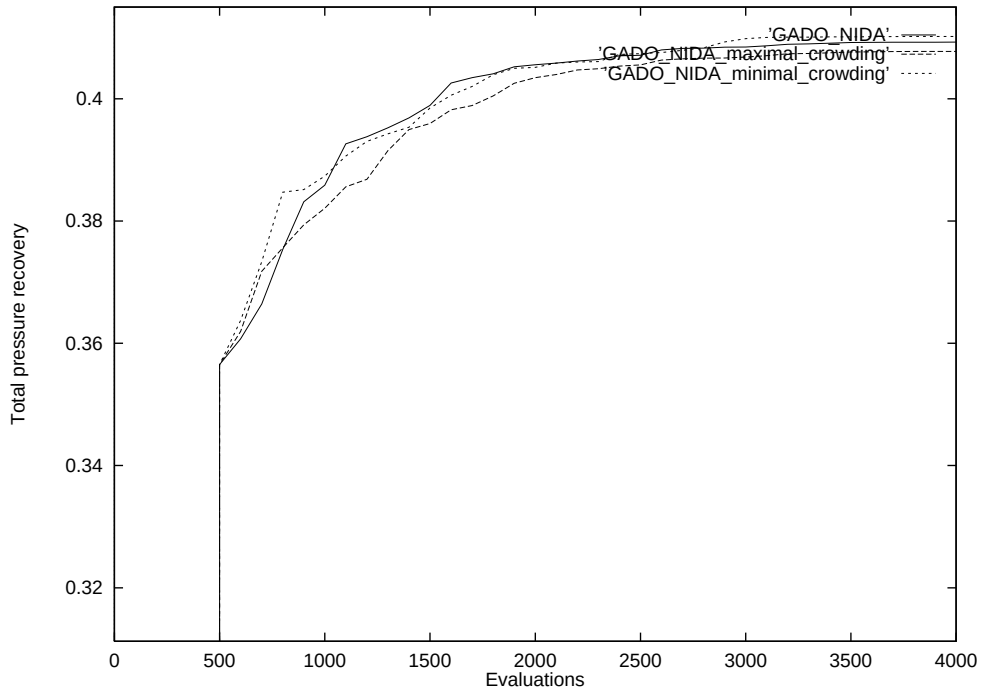

Figure 4.26: Effect of the crowding factors in application domain 2 (missile design)

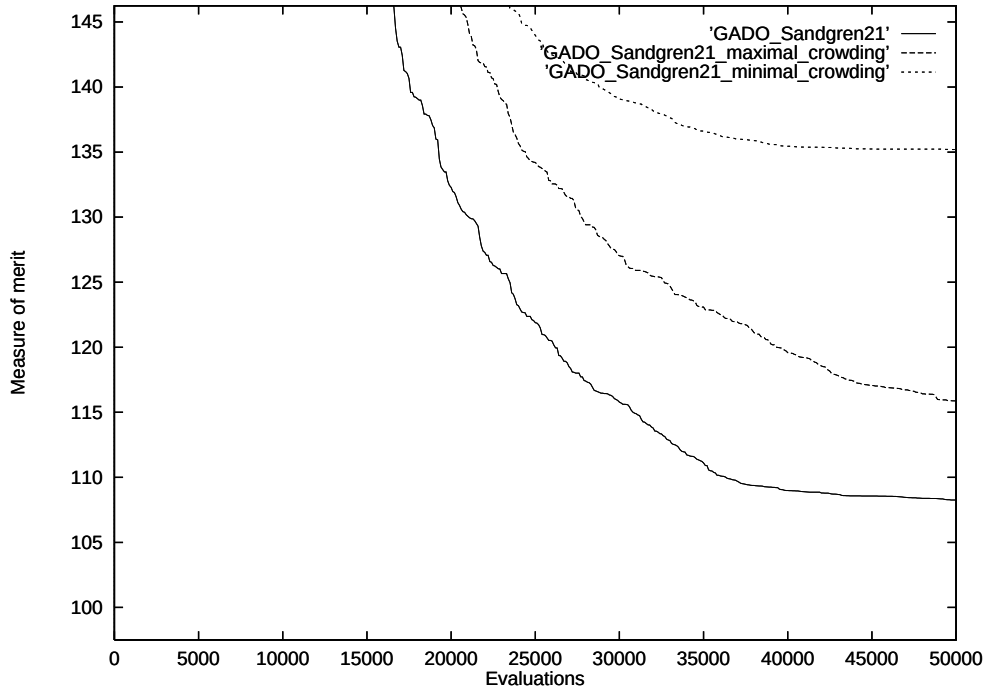

Figure 4.27: Effect of the crowding factors in benchmark domain 7 (Sandgren's problem 21)

- *Initial crowding factor*=1 and *final crowding factor*=1 (maximal crowding).
- *Initial crowding factor*=0 and *final crowding factor*=0 (minimal crowding).

In the first experiment we compared the average performance of 15 runs of GADO in application domain 1 (aircraft design). The results are shown in Figure 4.25. The figure shows that minimal crowding (replacement of the worst) was the best strategy in this domain, which is surprising considering its greedy nature. However, the results are conceivable because GADO uses several other diversity maintenance techniques (reseeding for example). The final performance, however, was the same for all three strategies.

We repeated the experiment in application domain 2 (missile design). The results are shown in Figure 4.26. The effect of the crowding factors was much less noticeable in this domain but still, minimal crowding was a little better than other 2 strategies.

We also repeated the experiment in benchmark domain 7. The results are shown in Figure 4.27. In this domain, minimal crowding was the worst, the default strategy was the best and maximal crowding was average.

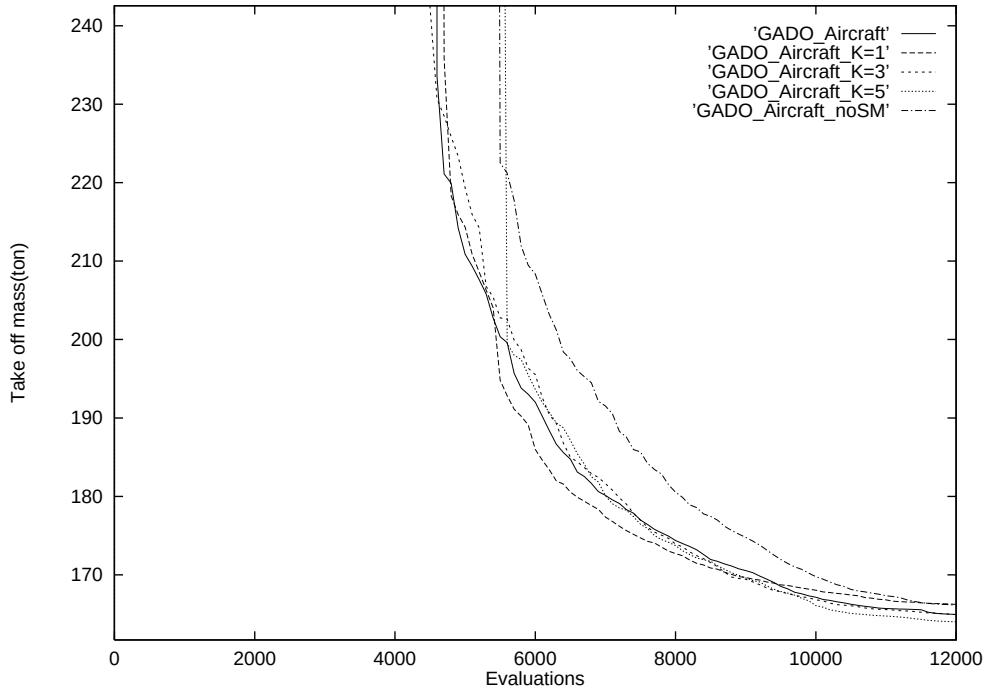

Figure 4.28: Effect of the K-nearest neighbor parameter K in application domain 1 (aircraft design)

Our general conclusion is that the default strategy is the most reliable in general. Minimal crowding could improve efficiency in some cases (such as application domain 1). In other case, however, it can severely degrade the performance (such as benchmark domain 7).

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 4.28. 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

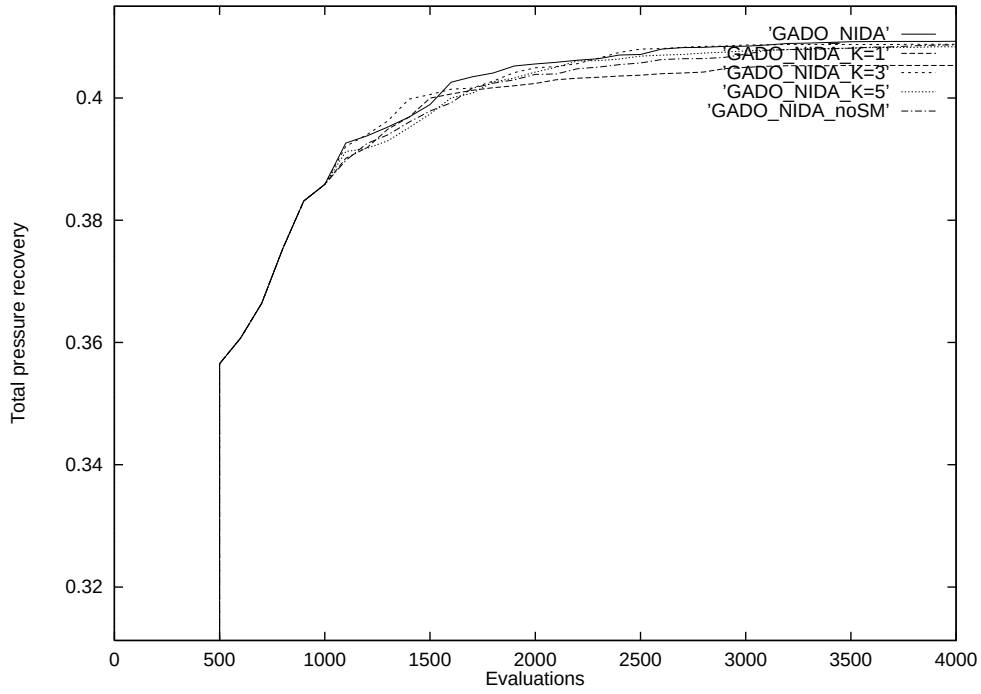

Figure 4.29: Effect of the K-nearest neighbor parameter K in application domain 2 (missile design)

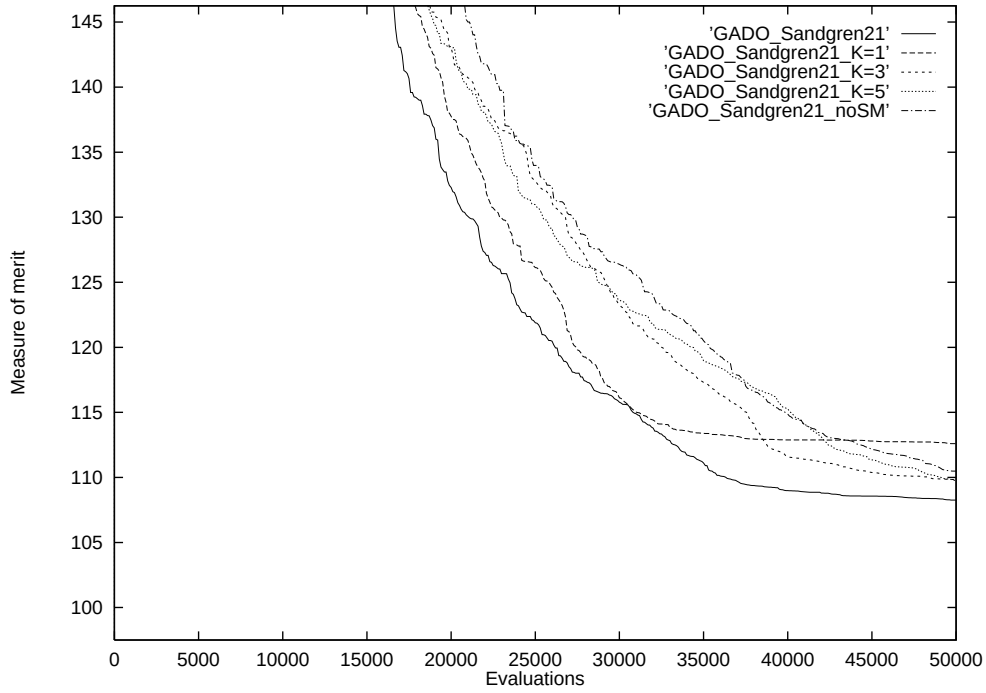

Figure 4.30: Effect of the K-nearest neighbor parameter K in benchmark domain 7 (Sandgren's problem 21)

are shown in Figure 4.29. The figure shows that the sensitivity to the parameter K is minimal in this domain also, except that for K=1 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 4.30. The figure demonstrates a moderate effect of the parameter K on performance. The best final performance was achieved with the default value of 2.

4.3.2 Effect of the search space structure

We investigated the effect of the search space structure on GADO's performance. We pursued this investigation in a synthetic domain rather than one of the realistic design domains we previously used. The reason for this is to have more control over the search space structure. Nevertheless, we made sure that the synthetic domain used suffered from all the structural problems described in Section 1.2. More precisely, we created a search space that had:

- many unevaluable and infeasible points.
- many local optima.
- a discontinuous objective function and/or first derivative.
- a badly structured feasible region.

For ease of visualization, we used a three-dimensional function. The optimization problem can be stated as:

$minimize \quad f(x,y) \quad where:$

$$f(x,y) = \begin{cases} (|x| + |y|) \cdot (1 + |sin(|x| \cdot \pi)| + |sin(|y| \cdot \pi)|) & if \quad -60 \leq x \leq 40 \\ & and \quad -30 \leq y \leq 70 \\ Unevaluable & Otherwise \end{cases}$$

$$subject \quad to:$$

$$-300 \leq x \leq 700$$

$$-400 \leq y \leq 600$$

$$(\frac{x+25}{30})^2 + (\frac{y}{1.5})^2 \leq 1$$

This search space has a very small evaluable region (only 1% of the space is evaluable). Inside this small evaluable region lies a very small elliptic feasible region (only 1.5% of the evaluable region is feasible). The feasible region is a thin ellipse with a very small ratio between its two radii (1:20). The local optima lie at the points which have integral values of both x and y. Therefore, there are 10000 local optima inside the evaluable region, of which 148 lie inside the feasible region. The use of absolute values causes a discontinuity in first derivative at every local optimum. The border between the evaluable and unevaluable region results in a discontinuity of the objective function. The global optimum is at the origin,[5] with an objective value of zero. Figure 4.31 illustrates the evaluable and the feasible regions. The feasible region is the ellipse near the middle of the figure. Figure 4.32 illustrates the the objective function landscape in the bounding rectangle of the feasible region.

In the first experiment we used GADO to minimize the objective function under the given constraints, without any transformations. The limit on the number of evaluations was set at 2000. The result is shown in Figure 4.33 (the solid curve). Each curve in Figure 4.33 represents the average of 10 runs of GADO. We also used GADO to find the maximum feasible value of the objective function (by changing the sign). The maximum feasible value turned out to be 161.64. Thus, the vertical range of Figure 4.33 is only 3% of the total objective range. The figure clearly shows that GADO reached the global optimum very easily in this case.

We then repeated the above experiment three times, each time with one of the following three transformations:

[5]This does not imply symmetry, however, as the parameter/evaluable/feasible ranges are not symmetric around the origin.

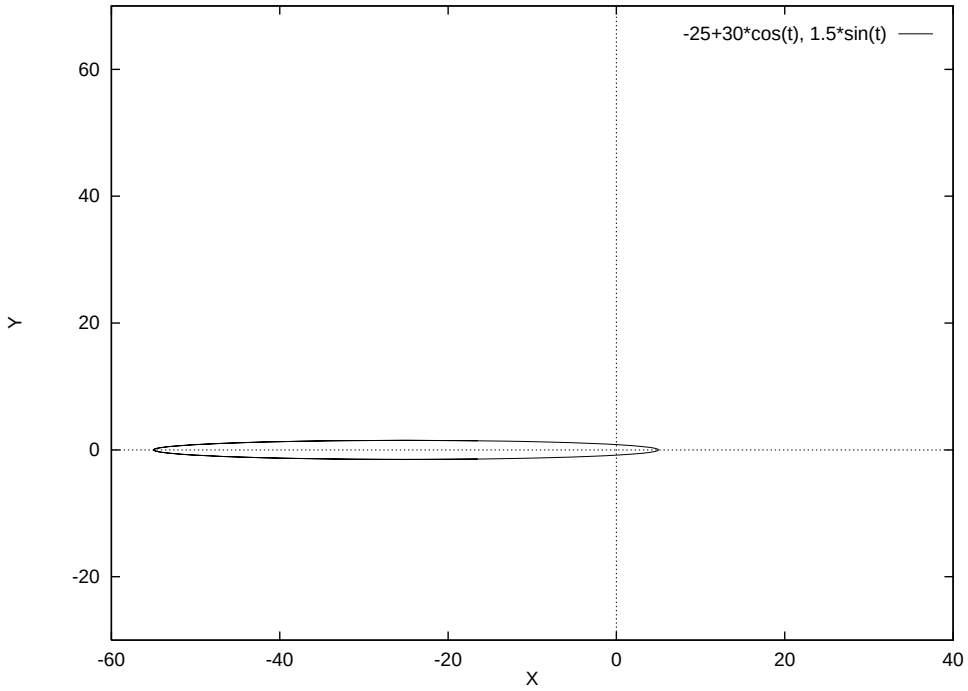

Figure 4.31: The evaluable and the feasible regions for the synthetic domain

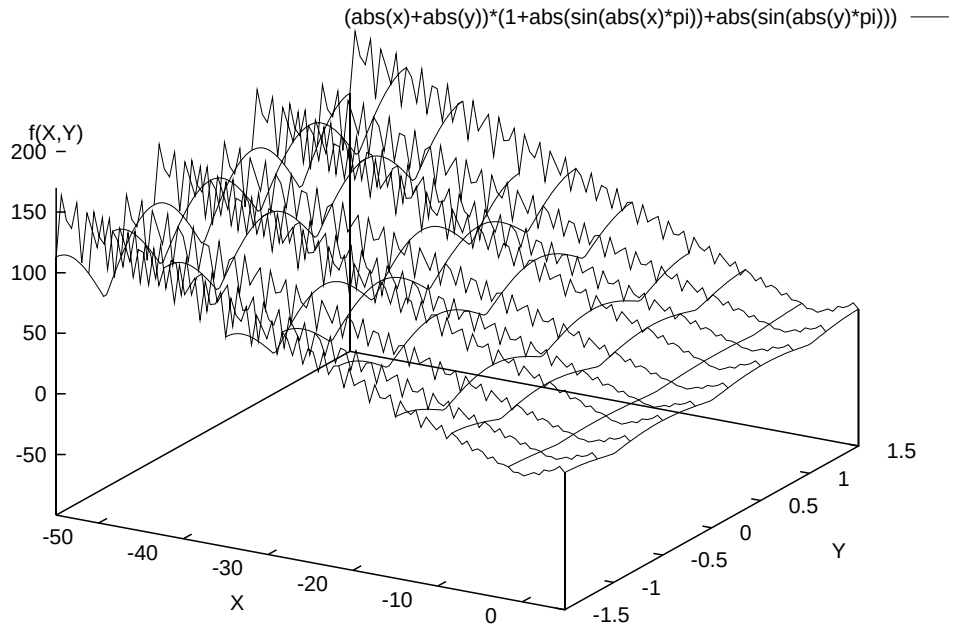

Figure 4.32: Surface plot of the objective function landscape for the synthetic domain

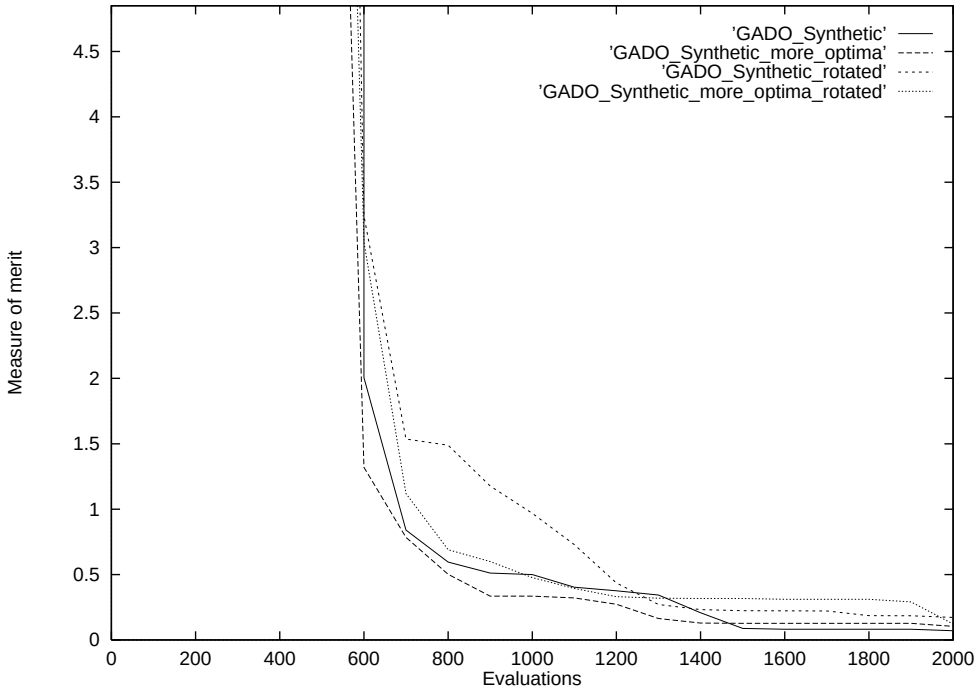

Figure 4.33: Effect of the search space structure on performance

1. The number of local optima was increased approximately 10 times. This was accomplished by changing the function expression in the evaluable region to be:

   $f(x, y) = (|x| + |y|) \cdot (1 + |sin(3 \cdot |x| \cdot \pi)| + |sin(3 \cdot |y| \cdot \pi)|)$

2. The coordinate axis were rotated by 0.5 radian clockwise. This is equivalent to rotating both the evaluable region and the feasible region by 0.5 radian anti-clockwise.

3. The number of local optima was increased and the coordinate axis were rotated (i.e. the combination of 1 and 2 above).

The results are also shown in Figure 4.33 (the dotted curves). The figure shows that the effect of all transformations on performance was minimal. This result suggests that unlike other GAs that degrade severely under coordinate rotation [Salomon 1996], GADO's performance degrades only slightly under these circumstances. The reason for this is that three of the five crossover methods used by GADO are rotation independent. The increase in the number of local optima did not seem to have any significant effect on performance.

4.4 Concluding remarks

In this chapter we analyzed the scope of applicability of GADO by testing it in further engineering design domains previously used in the literature. We also investigated the role of some of the key novel components of GADO. We demonstrated how these components significantly improved the overall performance in several domains. Finally, we analyzed the effect of parameter variation and search space structure on performance. In the next chapter the thesis is concluded with a general summary and a discussion of the future research directions.

Chapter 5

Conclusion

5.1 Summary

GADO is a highly adaptive 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 which typically arise in such applications. GADO has been applied in a variety of optimization tasks which 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 have continuous intervals. 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 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. The most innovative crossover method is guided crossover which emulates gradient based

methods to improve the local convergence of the GA. GADO also uses some search control strategies such as a screening module which saves time by avoiding the full evaluation of points that are unlikely to correspond to good designs.

5.2 Review of contribution

This thesis presents a new method for design optimization in engineering domains with complex, expensive evaluation functions. In Chapter 3, the new method (GADO) was compared to three state of the art optimizers in 2 modern engineering design domains. The superiority of GADO to the other techniques was clearly demonstrated. GADO includes many novel features. In Chapter 4, the utility and significance of some of these was demonstrated. The experiments described in Chapter 4 also show that GADO's performance is fairly stable with respect to parameter variation. Based on all the evidence reported in this thesis, we believe that GADO is a very reliable and efficient optimizer.

GADO includes many new ideas. We list some of them here, in decreasing order of novelty:

1. **The guided crossover operator.** This is an operator which endows the GA with some of the efficiency advantages of gradient-based methods without actually computing any gradients.

2. **The screening module.** This is a search control module that saves time by avoiding the expensive evaluation of points that are unlikely to be good.

3. **The reseeding module.** This is a module that restores diversity to the GA population in an effort to prevent premature convergence to a local optimum.

4. **The dynamic penalty method.** This methods ensures adequate sampling of the search space in the early stages of the optimization with a sufficient focus on feasibility towards the end of the optimization.

5. **The line crossover operator.** This is an operator that is particularly suitable for slab-like evaluable/feasible regions, a type of region that is not uncommon in

design optimization spaces.

6. **The double line crossover operator.** This is an operator that combines the merits of line crossover and the classical point crossover operator.

7. **The initialization strategy.** This strategy ensures that the GA starts from a population that contains an adequate number of evaluable points.

8. **The shrinking-window mutation operator.** This is an operator that promotes exploration in the beginning of the optimization and exploitation towards the end. This operator also pays special attention to the borders of the search space.

5.3 Limitations and future work

5.3.1 Application of GADO to other application domains

We are in the process of applying GADO to design optimization problems in various domains from different engineering disciplines. These include:

- **Design of a nano-powder thermal reactor:** This domain has an extremely expensive evaluation function (about 5 CPU minutes per evaluation on a workstation). Preliminary experiments were done using GADO, CFSQP and Random Probes. The maximum number of iterations each method was allowed was only 100. This is a very unusual setup for using a GA (or any global optimizer, as these methods tend to require a large number of iterations to converge). However, GADO gave the best result.

- **Design of two dimensional high speed inlets:** This is a collaborative effort with Aerospatiale Missiles (France). Several inlets have been designed. See Appendix C for details about the design process of one of these inlets. We are also in the process of using GADO for *mission-based* design of inlets. In mission-based design, the goal is to optimize the performance of the inlet over an entire mission profile, rather than one flying condition, which is the conventional approach.

- **Design of potential energy functions:** This has been a collaborative effort with material scientists. The evaluation functions are quite expensive (about 5 CPU seconds per evaluation on a workstation). Our preliminary results in simplified versions of these problems suggest that GADO could be a very robust optimizer in these domains.

- **Drug design:** This work is still in the planning phase. The first item on the agenda is the design of force fields (potential functions) for the modeling of the interactions of drugs with their biological receptors. This is very similar to the material science collaboration. The most interesting feature about the search spaces expected to arise in this domain is their very high dimensionality (300 or greater). It is expected that many other problems will be investigated.

5.3.2 Improvements in the screening module

The screening module proved to be one of the most important components of GADO (see Chapter 4 for details). 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 (like 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 SM success 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.

5.3.3 Extensions to guided crossover

It was shown that the guided crossover operator contributed a lot to the success of GADO (see Chapter 4 for details). 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.

5.3.4 Acquisition and utilization of domain knowledge

The screening module can be thought of as a very simple approach for acquiring knowledge about the domain and using it to guide further explorations. We plan on investigating the use of more sophisticated methods to acquire knowledge about the domain in the course of the GA optimization, such as neural networks. We can afford to use these relatively expensive methods because the objective functions are even more expensive. We also plan on exploring methods for utilizing the acquired domain knowledge. For example, if we can acquire the knowledge that the objective function is decomposable (for example, it can be expressed as the product of two functions, each of which is a function of a subset of the parameters, with the two subsets being disjoint), we can use this knowledge to solve the problem more efficiently by optimizing each subset separately. Domain knowledge can also be used to tailor GADO for a particular problem. If domain knowledge is available beforehand (for example, if we know the space is unimodal or has a low modality) it is easy to set the performance parameters to take advantage of this knowledge (like using guided crossover more intensely in this example). The more interesting case is when the knowledge becomes available only at run time. In this case the system needs to do some self-adaptation.

Appendix A

GADO parameters

The following is a list of the parameters used by GADO, together with their recommended (default) values:

- **maxpop:** The number of individuals in the GA population. The default is 10 times the dimension of the problem.
- **maxstore:** The number of sample points stored for use in screening and reseeding. The default is 30 times *maxpop*.
- **minevalpop:** The number of random individuals used to seed the initial population. The default 10 times *maxpop*.
- **maxevals:** The maximum number of individual evaluations (actual calls to the simulator) allowed. We recommend no less than 100 times *maxpop* but this is likely to vary significantly from problem to problem, depending on the available time and complexity of the simulator.
- **maxevals increment:** If this parameter is non-zero, then after *maxevals* evaluations have been done, the maximum allowed number of evaluations will be repeatedly incremented by *maxevals increment* until the optimization stops due to another stopping criterion (see for example *stoptolerence* below). The default is zero.
- **penaltycoef:** The initial penalty coefficient. The default is set as follows: if the average measure of merit is expected to have a numerical value between V and $10 * V$ where V is a power of ten, then *penaltycoef* should be set to $\frac{V}{100}$.[1]

[1] Using the value 1 worked very well in several domains, but we decided to make the default relative to the measure of merit rather than an absolute value because some domains may have very small

- **penaltymult:** This is an upper bound on how much the penalty coefficient can be multiplied at one increment. For example, if the penalty coefficient was 7 and the program decided it should increase, the maximum value it can take is 7**penaltymult*. The default is 2.

- **stoptolerance:** If the ratio between the average distance among points in the current population and the average distance among points in the initial population is less than *stoptolerence*, the optimization is terminated. The default value is zero (i.e. termination is based on number of evaluations).

- **rejecttolerance:** If the distance between a proposed point and an existing point in the sample used for screening is less than $\frac{rejecttolerence*godown^2}{maxpop}$ the point is rejected by the diversity maintenance module. The default is zero (i.e. only exact duplicates are rejected).

- **maxreseeds:** The maximum number of reseeding operations allowed. The default is 10.

- **reseedfraction:** If the ratio between the average distance among points in the current population and the average distance among points in the initial population is less than $reseedfraction*godown^2$, reseeding is done (unless *maxreseeds* reseeds have already been done). The default is 0.25.

- **mutation factor:** This controls the amplitude of mutation. A value of 1 is conservative, 2 is average and more than that is large. The default is 2.

- **GC factor:** The final proportion of time guided crossover is done instead of the regular crossover techniques. The default is 0.25.

- **no penalty decrease:** This will prevent the penalty coefficient from ever decreasing, even if all the population became feasible. If it is known that the optimum is not at the border of any constraint violations, then setting this option may save time. The default 0 (meaning decrease is allowed).

---

objective values which makes an initial penalty coefficient of 1 extremely large.

- **no screen:** If this is set to 1, no screening will be done. The default is 0.
- **K for K-nearest neighbor:** The default is 2.
- **screening offset:** If the screening offset is V then the fitness of the $(1+V)^{th}$ worst individual in the GA population is used as the threshold for screening. The default is 1 (i.e. the second worst).
- **initial crowding factor:** The initial value of the crowding factor. The default is one.
- **final crowding factor:** The final value of the crowding factor. The default is zero.

Appendix B

Benchmark domains

The following is a brief description of the benchmark domains used in Chapter 4:

1. Design gear ratios for a five speed automotive transmission in order to accelerate from rest to 100mph in minimum time.

2. Design a rectangular box to maximize volume, subject to post office restrictions on the length plus the girth and individual limits on the length, depth and height of the box.

3. Description not available.

4. Design a solid disk flywheel for maximum energy storage subject to constraints on the weight, diameter, speed of rotation and width.

5. Optimize an electrical network of two nodes.

6. Description not available.

7. Mathematical programming model of a three-stage membrane separation process. 11 out of the 13 inequality constraints are active at the global optimum.

8. Mathematical programming model of a five-stage membrane separation process. 16 out of the 19 inequality constraints are active at the global optimum.

Appendix C

Case study: Redesign of a two dimensional high speed missile inlet

C.1 Domain description

This domain concerns the design of a two dimensional, mixed compression, high speed inlet to maximize total pressure recovery. This inlet was experimentally designed in March 1996 by the Institute of Theoretical and Applied Mechanics (ITAM), in Novosibirsk (Russia). The total pressure recovery of the final design was 0.134. The main goal of this part of the research was to redesign the inlet using GADO and compare the resulting design to the experimental ITAM design in an effort to test and evaluate GADO. All the experiments in this domain were done by Mr. Michael Blaize, a design engineer with Aerospatiale Missiles (France) with only minimal guidance on the use of GADO.

The analysis code used in this research is a flow solver called OCEAS. OCEAS was developed at Aerospatiale Missiles to assist engineers in the aerodynamic design of missile inlets. It is a semi-empirical flow solver which uses simple but accurate physical models that require little CPU time. Each run of OCEAS takes about 2 CPU seconds on a DEC ALPHA 2100 workstation. Evaluation of the objective and penalty values was done differently in this domain than all the other domains described in this thesis. In each evaluation, some preliminary constraint calculations are done (which takes about 0.05 CPU seconds on the same workstation). Based on these calculations, it is decided whether the design is evaluable or unevaluable. OCEAS is run *only* if the design is evaluable, to calculate the total pressure recovery and the remaining constraints.

C.2 Experiments and results

C.2.1 Design optimization using GADO

When GADO was used for the design optimization of the ITAM inlet, a 5000 iteration optimization was done in which OCEAS was called 2246 times (i.e. 2246 feasible designs were examined). The total pressure recovery of the best found inlet was 0.194. The entire optimization took about 1.25 CPU hours.

C.2.2 Design optimization using CFSQP

Further experiments were conducted in which CFSQP was used for the design optimization of the same inlet.[1] GADO did not need a starting point (and none was given to it) but CFSQP required a starting point. Several methods were used to provide the starting point to CFSQP resulting in different experiments as described below.

CFSQP from the experimental ITAM inlet

In this experiment the experimental ITAM inlet design was used as the starting point for CFSQP. CFSQP stopped after a very small number of iterations (less than 20 calls to OCEAS). The best design found by CFSQP in this experiment had a total pressure recovery of 0.160.

CFSQP from the best inlet found by GADO

In this experiment the best inlet found by GADO was used as the starting point for CFSQP. However, CFSQP failed to yield any improvement over the starting point. This suggests that the best inlet found by GADO may well be locally optimal.

[1] The version of CFSQP used in this domain was also the one enhanced by Mark Schwabacher [Schwabacher 1996] (the one used in other domains in this thesis).

Multi-start CFSQP

In this experiment a very long sequence of random probes were done and every time an evaluable "near-feasible"[2] point was found, CFSQP was started from that point. This process was repeated for more than one CPU day. The best inlet found in this experiment had a total pressure recovery that is approximately the same as the experimental ITAM inlet.

It was concluded that GADO is much more reliable than CFSQP for this class of problems.

[2]In an earlier version of this experiment, CFSQP was started from every evaluable point found. It was observed that CFSQP failed to find any feasible points under this setup. Thus the strategy was modified so that CFSQP is started only whenever an evaluable point with a small sum of constraint violations is found.

References

[Andersen *et al.* 1993] K. Andersen, V.B. Iversen, and R.V.V. Vidal. Design of a teleprocessing communication network using simulated annealing. In *Applied Simulated Annealing*, pages 201–216. Springer-Verlag, 1993.

[Back *et al.* 1997] Tomas Back, Ulrich Hammel, and Hans-Paul Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17, 1997.

[Blaize *et al.* 1998] Michael Blaize, Doyle Knight, and Khaled Rasheed. Automated optimal design of two dimensional high speed missile inlets. In *to appear in the 36th AIAA Aerospace Sciences Meeting and Exhibit*, 1998.

[Bouchard *et al.* 1988] E. E. Bouchard, G. H. Kidwell, and J. E. Rogan. The application of artificial intelligence technology to aeronautical system design. In *AIAA/AHS/ASEE Aircraft Design Systems and Operations Meeting*, Atlanta, Georgia, September 1988. AIAA-88-4426.

[Bramlette *et al.* 1989] M. Bramlette, E. Bouchard, E. Buckman, and L. Takacs. A comparative evaluation of search methods applied to parametric design of aircraft. In *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo,Calif., July 1989. Morgan Kaufmann.

[Bramlette *et al.* 1990] M. Bramlette, E. Bouchard, E. Buckman, and L. Takacs. Current applications of genetic algorithms to aeronautical systems. In *Proceedings of the Sixth Annual Aerospace Applications of Artifical Intelligence Conference*, October 1990.

[Cagan and Kurfess 1992] J. Cagan and T. R. Kurfess. Optimum tolerance allocation over multiple manufacturing alternatives. *ASME Advances in Design Automation Conference*, 44(2):165–172, 1992.

[Chapman and Jakiela 1996] C. D. Chapman and M. J. Jakiela. Genetic algorithm-based structural topology design with compliance and topology simplification considerations. *Journal of Mechanical Design*, 118(1), 1996.

[Chardaire and Lutton 1993] P. Chardaire and J.L. Lutton. Using simulated annealing to solve concentrator location problems in telecommunication networks. In *Applied Simulated Annealing*, pages 175–200. Springer-Verlag, 1993.

[Deb and Goyal 1997] Kalyanmoy Deb and Mayank Goyal. Optimizing engineering designs using a combined genetic search. In *Proceedings of the Seventh International Conference on Genetic Algorithms*. Morgan Kaufmann, 1997.

[Deb and Saxena 1997] Kalyanmoy Deb and Vikas Saxena. Car suspension design for comfort using genetic algorithms. In *Proceedings of the Seventh International Conference on Genetic Algorithms*. Morgan Kaufmann, 1997.

[Deb 1997] Kalyanmoy Deb. Geneas: A robust optimal design technique for mechanical component design. In *Evolutionary Algorithms in Engineering Applications*, pages 497–514. Springer-Verlag, 1997.

[Gage *et al.* 1995] P. Gage, I. Kroo, and I. Sobieski. Variable-complexity genetic algorithm for topological design. *AIAA Journal*, 33(11):2212–2217, 1995.

[Gage 1994] P. Gage. New approaches to optimization in aerospace conceptual design. Ph.D. Thesis, Stanford University, 1994.

[Gelsey *et al.* 1996a] A. Gelsey, D. Smith, M. Schwabacher, K. Rasheed, and K. Miyake. A search space toolkit: SST. *Decision Support Systems*, 18:341–356, 1996.

[Gelsey *et al.* 1996b] 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.

[Gero *et al.* 1997] John S. Gero, Vladimir A. Kazakov, and Thorsten Schinier. Genetic engineering and design problems. In *Evolutionary Algorithms in Engineering Applications*, pages 47–68. Springer-Verlag, 1997.

[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.

[Hoeltzel and Chieng 1987] D. Hoeltzel and W. Chieng. Statistical machine learning for the cognitive selection of nonlinear programming algorithms in engineering design optimization. In *Advances in Design Automation*, Boston, MA, 1987.

[Holland 1975] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[Ingber 1996] Lester Ingber. Adaptive simulated annealing (ASA): Lessons learned. *Control and Cybernetics*, 25(1):33–54, 1996.

[Jain and Agogino 1988] P. Jain and A. M. Agogino. Optimal design of mechanisms using simulated annealing: Theory and applications. *ASME Advances in Design Automation*, 14:233–240, 1988.

[Jain *et al.* 1992] P. Jain, P. Fenyes, and R. Richter. Optimal blank nesting using simulated annealing. *ASME Journal of Mechanical Design*, 114:160–165, 1992.

[Janikow and Michalewicz 1991] Cezary Janikow and Zbigniew Michalewicz. An experimental comparison of binary and floating point representations in genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 31–36. Morgan Kaufmann, 1991.

[Kirkpatrick *et al.* 1983] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671 – 680, 1983.

[Knight 1994] D. Knight. Survey of high-speed inlet design methodology. Technical Report ARPA Inlet Design Group Report No. 1, Department of Mechanical and Aerospace Engineering, Rutgers University, New Brunswick, NJ, March 1994. http://www.cs.rutgers.edu/hpcd/Area_II.1/report1.ps.Z.

[Kroo *et al.* 1994] Ilan Kroo, Steve Altus, Robert Braun, Peter Gage, and Ian Sobieski. Multidisciplinary optimization methods for aircraft preliminary design. In *Fifth AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Panama City, Florida, September 1994. AIAA 94-4325.

[Kundu and Kawata 1996] Sourav Kundu and Seiichi Kawata. AI in control system design using a new paradigm for design representation. In *Fourth International Conference on Artificial Intelligence in Design '96*, 1996.

[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.

[Land *et al.* 1997] Mark Land, John Sidorowich, and Richard Belew. Using genetic algorithms with local search for thin film metrology. In *Proceedings of the Seventh International Conference on Genetic Algorithms*. Morgan Kaufmann, 1997.

[Lawrence *et al.* 1995] C. Lawrence, J. Zhou, and A. Tits. User's guide for CFSQP version 2.3: A C code for solving (large scale) constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality constraints. Technical Report TR-94-16r1, Institute for Systems Research, University of Maryland, August 1995.

[Lienig and Thulasiraman 1993] Jens Lienig and K. Thulasiraman. A genetic algorithm for channel routing in VLSI circuits. *Evolutionary Computation*, 1(4):293–311, 1993.

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

[Mahfoud 1995] Samir Mahfoud. A comparison of parallel and sequential niching methods. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 136–143. Morgan Kaufmann, July 1995.

[Malhotra *et al.* 1991] A. Malhotra, J. H. Oliver, and Tu Weizhen. Synthesis of spatially and intrinsically constrained curves using simulated annealing. *Advances in Design Automation*, 1:145–155, 1991.

[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.

[Obayashi *et al.* 1997] S. Obayashi, Y. Yamaguchi, and T. Nakamura. Multiobjective genetic algorithm for multidisciplinary design of transonic wing platform. *Journal of Aircraft*, 34(5), 1997.

[Ogot and Alag 1995] M. M. Ogot and S. S. Alag. An effective mixed annealing/heuristic algorithm for problems in mechanical design. *Journal of Mechanical Design*, 117(3), 1995.

[Orelup *et al.* 1988] M. F. Orelup, J. R. Dixon, P. R. Cohen, and M. K. Simmons. Dominic II: Meta-level control in iterative redesign. In *Proceedings of the National Conference on Artificial Intelligence*, pages 25–30, St. Paul, MN, 1988.

[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.

[Powell 1990] 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.

[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 and Gelsey 1996] 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.

[Rasheed and Hirsh 1997a] Khaled Rasheed and Haym Hirsh. Guided crossover: A new operator for genetic algorithm based optimization. Technical Report HPCD-TR-50, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1997.

[Rasheed and Hirsh 1997b] Khaled Rasheed and Haym Hirsh. Using case-based learning to improve genetic-algorithm-based design optimization. In *Proceedings of the Seventh International Conference on Genetic Algorithms.* Morgan Kaufmann, 1997.

[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.

[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.

[Rogers *et al.* 1996] James L. Rogers, Collin M. McCulley, and Christina L. Bloebaum. Integrating a genetic algorithm into a knowledge based system for ordering complex design processes. In *Fourth International Conference on Artificial Intelligence in Design '96*, 1996.

[Rosenman 1997] M. A. Rosenman. The generation of form using an evolutionary approach. In *Evolutionary Algorithms in Engineering Applications*, pages 69–86. Springer-Verlag, 1997.

[Salomon 1996] Ralf Salomon. Performance degradation of genetic algorithms under coordinate rotation. In *Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 155–161, 1996.

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

[Schwabacher and Gelsey 1997] M. Schwabacher and A. Gelsey. Intelligent gradient-based search of incompletely defined design spaces. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 11(3):199–210, 1997.

[Schwabacher 1996] Mark Schwabacher. The use of artificial intelligence to improve the numerical optimization of complex engineering designs. Technical Report HPCD-TR-45, Department of Computer Science, Rutgers University, October 1996. Ph.D. Thesis. http://www.cs.rutgers.edu/∼schwabac/thesis.html.

[Sobieszczanski-Sobieski and Haftka 1996] J. Sobieszczanski-Sobieski and R. T. Haftka. Multidisciplinary aerospace design optimization: Survey of recent developments. In *34th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, January 1996. AIAA-96-0711.

[Sobieszczanski-Sobieski *et al.* 1985] J. Sobieszczanski-Sobieski, B. B. James, and A. R. Dovi. Structural optimization by multilevel decomposition. *AIAA Journal*, 23(11):1775–1782, November 1985.

[Tong *et al.* 1992] 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.

[Tong 1988] S. S. Tong. Coupling symbolic manipulation and numerical simulation for complex engineering designs. In *International Association of Mathematics and Computers in Simulation Conference on Expert Systems for Numerical Computing*, Purdue University, 1988.

[Vanderplaats 1984] Garret N. Vanderplaats. *Numerical Optimization Techniques for Engineering Design : With Applications*. McGraw-Hill, New York, 1984.

[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.

[Zha *et al.* 1997] G.-C. Zha, D. Smith, M. Schwabacher, K. Rasheed, A. Gelsey, D. Knight, and M. Haas. High performance supersonic missile inlet design using automated optimization. *AIAA Journal of Aircraft*, 34(6), 1997.

Vita

Khaled Mohamed Rasheed

1985-90 Attended Alexandria University, Egypt. Majored in Honors Computer Science.

1990 B.Sc., Alexandria University.

1993-94 Teaching Assistant, Department of Computer Science.

1994-97 Graduate Assistant, Department of Computer Science.

1995 M.S. in Computer Science, Rutgers, The State University of New Jersey.

1996 A. Gelsey, D. Smith, M. Schwabacher, K. Rasheed, and K. Miyake. A Search Space Toolkit. *Decision Support Systems*, 18:341-356.

1997 K. Rasheed, H. Hirsh, A. Gelsey. A Genetic Algorithm for Continuous Design Space Search. *Artificial Intelligence in Engineering*, 11(3).

1997 G.-C. Zha, D. Smith, M. Schwabacher, K. Rasheed, A. Gelsey, D.Knight and Martin Haas. High Performance Supersonic Missile Inlet Design Using Automated Optimization. *Journal of Aircraft*, 34(6).

1997 K. Rasheed, H. Hirsh. Using Case Based Learning to Improve Genetic Algorithm Based Design Optimization. *Proceedings of the Seventh International Conference on Genetic Algorithms.*

1998 Ph.D. in Computer Science.