Multiobjective Genetic Algorithms with Application to Control Engineering Problems

Thesis submitted for candidature for the degree of PhD

Carlos Manuel Mira da Fonseca

September 1995

Department of Automatic Control and Systems Engineering

The University of Sheffield

Summary

Genetic algorithms (GAs) are stochastic search techniques inspired by the principles of natural selection and natural genetics which have revealed a number of characteristics particularly useful for applications in optimization, engineering, and computer science, among other fields. In control engineering, they have found application mainly in problems involving functions difficult to characterize mathematically or known to present difficulties to more conventional numerical optimizers, as well as problems involving non-numeric and mixed-type variables. In addition, they exhibit a large degree of parallelism, making it possible to effectively exploit the computing power made available through parallel processing.

Despite their early recognized potential for multiobjective optimization (almost all engineering problems involve multiple, often conflicting objectives), genetic algorithms have, for the most part, been applied to aggregations of the objectives in a single-objective fashion, like conventional optimizers. Although alternative approaches based on the notion of Pareto-dominance have been suggested, multiobjective optimization with genetic algorithms has received comparatively little attention in the literature.

In this work, multiobjective optimization with genetic algorithms is reinterpreted as a sequence of decision making problems interleaved with search steps, in order to accommodate previous work in the field. A unified approach to multiple objective and constraint handling with genetic algorithms is then developed from a decision making perspective and characterized, with application to control system design in mind. Related genetic algorithm issues, such as the ability to maintain diverse solutions along the trade-off surface and responsiveness to on-line changes in decision policy, are also considered.

The application of the multiobjective GA to three realistic problems in optimal controller design and non-linear system identification demonstrates the ability of the approach to concurrently produce many good compromise solutions in a single run, while making use of any preference information interactively supplied by a human decision maker. The generality of the approach is made clear by the very different nature of the two classes of problems considered.

Acknowledgements

I would like to thank my supervisor Professor Peter Fleming for his help, support, and encouragement throughout this research programme. Thanks are also due to A. Shutler of Rolls-Royce for sharing his expertise and knowledge in connection with the Pegasus case-study in Chapter 5.

My colleague and friend Eduardo Mendes deserves a special mention, for the good collaborative work on which Chapter 6 was based. I wish to thank also Luís Aguirre, and all members of the Parallel Processing Group, in particular Julian Bass, Michael Baxter and Andrew Chipperfield, for their friendship and for many enlightening discussions.

I thank my Parents for their love and support from my first day of life; my brothers and sister for encouragement; and my wife and colleague Viviane Grunert for her dedication and patience, and for exposing me to the field of non-parametric statistics halfway through this project.

Finally, I wish to acknowledge Programa CIENCIA, Junta Nacional de Investigação Científica e Tecnológica, Portugal, for financial support under Grant BD/1595/91-IA, and the UK Engineering and Physical Sciences Research Council (Grant GR/J70857) in completion of this work.

Statement of Originality

Unless otherwise stated in the text, the work described in this thesis was carried out solely by the candidate. None of this work has already been accepted for any other degree, nor is it being concurrently submitted in candidature for any degree.

Candidate: ______________________________

Carlos Manuel Mira da Fonseca

Supervisor: ______________________________

Peter J. Fleming

Aos meus Pais, irmãos e irmã.

Für Viviane.

Contents

Chapter 1

Introduction

1.1 Motivation

Nature, with all its beauty, diversity, and complexity, has fascinated humankind throughout the ages. It is indeed difficult to always remain insensitive to the sight of a sandy beach, a rocky mountain, or the immense ocean, and do not stop to admire the multitude of life forms which populate them. And yet, all that beauty hides a fierce, interminable battle for survival, where the stronger invariably wins over the weaker, or else it would not be stronger.

Nature's way of simply discarding individuals unable to compete in favour of competitive ones, coupled with environmental effects and the individuals' own ability to reproduce, has proven extremely effective in evolving highly complex organisms. Of these, humans themselves are just surviving examples, whereas others have become permanently extinct due to their inability to adapt to the changing environment.

As the process of natural evolution became better understood and modern computing power became increasingly available, the ability to simulate evolutionary systems on the computer soon began to attract growing interest from the scientific community. In addition to generating further insight into the process of natural evolution, artificial evolutionary techniques also revealed characteristics which would be particularly useful for applications in optimization, engineering and computer science, among other fields.

1.2 Summary of the thesis

This work seeks to explore some of the potential of artificial evolutionary techniques, in particular genetic algorithms (GAs) (Holland, 1975; Goldberg, 1989), for optimization and search in control systems engineering. Numerical optimization is already well established in Computer-Aided Control System Design (CACSD), for example, which is an area where objective functions can be expensive to evaluate and often cannot be guaranteed to satisfy all of the requirements of conventional numerical optimizers. Furthermore, CACSD often involves multiple, conflicting objectives which must be accommodated by the optimizer. These and other issues commonly arising in control engineering problems, and ways in which they can be addressed through the genetic algorithm approach, are discussed in Chapter 2. This Chapter, which begins with a brief introduction to evolutionary computation followed by a more detailed description of genetic algorithms, also gives selected examples of previous applications of GAs in control. In the conclusion, multiobjective optimization is identified as one of the most promising and least explored application areas of GAs.

Chapter 3 pauses to provide the necessary background for the work on multiobjective genetic algorithms presented in the remaining Chapters. It covers some of the basic concepts in multiobjective optimization, explaining the need for decision making and describing some common approaches to preference articulation. Constrained optimization is also introduced in the context of multiple objectives.

The visualization of results obtained through multiobjective optimization is the final topic of Chapter 3. In particular, the assessment of the statistical performance of an arbitrary multiobjective optimizer is considered, and a novel visualization technique developed.

In Chapter 4, a survey and discussion of current evolutionary approaches to constrained and multiobjective optimization leads to the understanding of evolutionary algorithms as sequences of decision-making problems involving a finite number of individuals. This makes it possible to approach multiobjective genetic algorithm development in two stages.

In a first stage, a very general preference articulation approach based on goal and priority information is developed which encompasses several multiobjective and constrained formulations, including Pareto and lexicographic optimization. The approach is duly characterized, and the effects of preference changes in the cost surface it induces are illustrated graphically. In a second stage, aspects of the genetic algorithm proper are considered, such as fitness assignment, the need for niche induction techniques, and the implications of allowing preferences to be refined as the algorithm runs. The resulting genetic algorithm is then applied to a simple problem, to illustrate the rôle of its various components.

The application of this multiobjective genetic algorithm to two controller design problems involving various numbers of design variables and objectives, and to a non-linear identification problem involving real data, is reported on in Chapters 5 and 6, respectively. The generality of the approach becomes evident once the very different nature of the two types of problems is considered, the first involving real-valued parameters, and the second consisting essentially of a (combinatorial) subset selection problem. The practical need for some degree of decision making in optimization, as opposed to pure Pareto optimization, is made especially clear by the second application example in Chapter 5, which is based around the full non-linear model of a Pegasus gas turbine engine.

Conclusions and directions for future work are drawn in Chapter 6.

1.3 Contributions

The main contributions of this thesis are:

- A survey and discussion of genetic algorithms in control engineering (Fleming and Fonseca, 1993). A bibliography survey conducted locally and over the Internet in 1992/93 identified around 40 papers on control engineering applications of GAs, including those discussed in Chapter 2. The full bibliography appeared in volume 7, number 18 (July 1, 1993) of the electronic-mail digest `GA-List@aic.nrl.navy.mil`.

- A survey and discussion of evolutionary algorithms in multiobjective optimization (Fonseca and Fleming, 1995d). Here, current evolutionary approaches to multiobjective optimization are critically reviewed, and some of the issues raised by multiobjective optimization in the context of evolutionary search are identified.

- A unified, scale-independent preference articulation framework for multiobjective and constrained optimization (Fonseca and Fleming, 1995b). Combining goal and priority information with the notion of Pareto-dominance results in a transitive relational operator (preferability) of which Pareto-dominance is a particular case. This operator allows sets of candidate solutions to be ranked according to (absolute) goal information and any priority information available, without the need for additional, and usually less reliable, relative scale information. When used in the selection stage of a genetic algorithm, the preferability operator makes preference refinement particularly easy, even as the optimization progresses.

- The formulation of fitness sharing and mating restriction techniques in the objective domain, and the development of guidelines for the setting of the associated niche sizes based on the properties of the non-dominated set

(Fonseca and Fleming, 1993). The importance of such niching techniques in multimodal problems has long been recognized, but the setting of niche sizes is usually difficult in practice.

- The application of multiobjective GAs to mixed $\mathcal{H}_2/\mathcal{H}_\infty$ reduced-order controller design (Fonseca and Fleming, 1994). This is a relatively simple, but realistic problem with no known analytical solution. Since the objective functions involved can be computed fairly efficiently, it becomes possible to show how consistently the multiobjective genetic algorithm performs across multiple runs. It also provides a good illustration of competing vs. non-competing objectives.

- The application of multiobjective GAs to gas turbine controller optimization (Fonseca and Fleming, 1995c), with emphasis on the characterization of the cost surfaces involved, and on the ability for a human decision maker to interact with the genetic algorithm as it runs.

- The application of multiobjective GAs to term selection in non-linear system identification involving real data (Fonseca *et al.*, 1993). Here, the genetic algorithm reveals a trade-off between two different identification criteria, offering the engineer a selection of potentially good models for further evaluation.

Additional contributions are:

- The foundations for a Genetic Algorithm Toolbox for MATLAB (Chipperfield *et al.*, 1994). All genetic algorithm components, as well as all objective functions, were implemented in MATLAB (The MathWorks, 1992a). These routines are now part of a Toolbox currently used in over 100 sites worldwide.

- A non-parametric technique for the interpretation and visualization of the results of multiple multiobjective optimization runs (Chapter 3). This provides a description of the statistical performance of a general multiobjective optimizer, including multiobjective GAs, across the whole Pareto-set of a given problem.

- A technique for the visualization of the cost surfaces induced by non-aggregating multiobjective cost assignment strategies such as Pareto-ranking (Fonseca and Fleming, 1995b; Fonseca and Fleming, 1995d). The underlying interpretation of population ranking as providing (biased) estimates for the probability of improvement of solutions via random search, which is equally valid in the single-objective case, also offers a basis for the concerted theoretical analysis of single objective and multiobjective GAs.

Chapter 2

Review of Genetic Algorithms and their Applications in Control

2.1 Introduction

Throughout history, Nature has been a major source of inspiration and metaphors for scientific and technical development, which, in turn, has contributed to the better understanding of Nature itself. It is not difficult to identify the links between the ear and the microphone, the eye and the camera, the bat and the sonar system, the brain and artificial neural networks, and so forth. Similarly, the process of natural evolution has inspired a growing amount of research in artificial systems, which has been made possible by the increasing availability of modern computing power.

This Chapter seeks to draw attention to the potential of artificial evolution techniques in control engineering. It begins with a brief description of evolutionary computation as a whole in Section 2.2, after which a particular family of methods, known as genetic algorithms, is introduced and discussed in more detail.

Section 2.4 places genetic algorithms in the context of optimization, and examples of their previous application in control engineering are reviewed in Section 2.5.

The Chapter concludes with a discussion of the classes of problems to which genetic algorithms appear to be best suited. The motivations for the work on multiobjective optimization which will follow are also identified.

2.2 Evolutionary computation

The term Evolutionary Computation (EC) is used to describe a broad class of computational methods inspired in the process of natural evolution. The interest such methods have attracted from research fields as diverse as biology, chemistry, economics, engineering, cognitive science and mathematics, among others, is twofold. Firstly, EC provides a means of modelling and simulating natural and economic processes, through which the original processes can be studied and/or analyzed. In addition to that, EC has also proved useful in addressing complex search and optimization problems effectively, extending the scope of optimization to areas it could not reach before.

Evolutionary computation methods, or Evolutionary Algorithms (EAs), emerged in the late 1960s. In Germany, *Evolutionstrategien* (Evolution Strategies, ESs) were developed by Rechenberg (1973; 1994) and Schwefel (1981), whilst in the United States two further approaches arose: Fogel's Evolutionary Programming (EP) (Fogel, 1991), and Holland's Genetic Algorithms (GAs) (Holland, 1975). Similar methods were presumably developed also in Poland (Karcz-Dulęba, 1994), where the term "soft selection" was used to differentiate the evolutionary approach from traditional hill-climbing.

Each of the methods above places emphasis on different aspects of what can be seen as Evolutionary Computation as a whole. Evolution Strategies have mainly been seen as robust, and yet efficient, optimizers for engineering design problems involving many real decision variables. Usual assumptions/pre-requisites of other optimizers, such as continuity and smoothness of the cost surface, gave way to the much broader assumption of strong causality, valid for most problems in engineering. Evolutionary Programming was originally developed in the context of Artificial Intelligence to evolve the state transition tables of finite state machines (Bäck and Schwefel, 1993), but was then extended to work with real decision variables.

Genetic algorithms have placed a much stronger emphasis than their counterparts on global, as opposed to local, search and optimization. GAs were initially formulated as combinatoric search algorithms, which required discretization of the search space in order to be applied to problems involving real decision variables. Rather than being seen as a weakness, the need for discretization made GAs appear particularly suitable for implementation on digital computers, while maintaining an attractive analogy to natural genetics. In Nature, sequences of genes taking one of four possible values each find expression in both continuous and discrete somatic features, examples of which are, respectively, body weight and number of body segments. Theoretical arguments based on the building-block hypothesis and the Schema Theorem (Holland, 1975; Goldberg, 1989) also appeared to support discretization.

Their claims of globality, the ease with which they can be modified and applied to very different classes of problems, and many encouraging results, have conferred great popularity to GAs. Unfortunately, theory has been slow to accompany their practical development, leading to a situation where theory and practice often conflict, or even contradict each other. There is to date no strong predictive theory of GAs applicable to problems of realistic size, despite some contributions in that direction (Manderick *et al.*, 1991).

More recently, as the different EA communities merge and interact, the need for a better understanding of the principles of evolution clearly begins to assume greater importance than simply concentrating on the details of every procedural variation of each particular "standard" algorithm. It is in line with these considerations that genetic algorithms will be introduced in the next section.

2.3 Genetic algorithms

Genetic algorithms are stochastic search algorithms inspired by the principles of natural selection and natural genetics. A population of candidate solutions, or individuals, is maintained, and individuals made to compete with each other for survival. Once evaluated, stronger individuals are given a greater chance of contributing to the production of new individuals (the offspring) than weaker ones, which may not even contribute at all. Offspring are produced through *recombination*, whereby they inherit features from each of the parents, and through *mutation*, which can confer some truly innovative features as well. In the next selection step, offspring are made to compete with each other, and possibly also with their parents. Improvement in the population arises as a consequence of the repeated selection of the best parents, which are in turn more likely to produce good offspring, and the consequent elimination of low-performers.

2.3.1 The population

In GAs, individuals are considered at two levels: the *phenotypic* level and the *genotypic* level. An individual's *phenotype* is its value in the domain over which the objective function is defined, constituting a candidate setting for the decision variables of the problem. This is analogous to features such as stature, weight, number of limbs, and limb length. On the other hand, an individual's *genotype* is a *representation* of its phenotype at a lower level, analogous to the genetic sequences contained in biological chromosomes, which the computer stores and the GA manipulates. The phenotype is therefore *encoded* in the genotype, traditionally as a bit string, but more generally as any convenient data structure. Strings have been preferred for simplicity and for the analogy with natural chromosomes, but other structures such as matrices, trees, and even lists of rules (Grefenstette, 1989), are also used.

The distinction between the phenotype and its genotypic representation is important and will be adhered to here, because it facilitates the extension of the approach to novel classes of problems. However, it is not necessarily a rigid one: a genotype and the corresponding phenotype can also be the same, in which case the mapping between the two is trivial. The genotypic representation and the associated genotype-to-phenotype mapping, which may be a one-to-one or a many-to-one mapping, are very important design parameters of the GA. One-to-many and many-to-many mappings may also occur in practice, for example, as a consequence of uncertainty in the evaluation of the objective function.

2.3.2 Evaluation and fitness assignment

Individuals are evaluated via the objective function which defines the problem. As a result of the evaluation, they are assigned a certain cost (assuming a minimization problem), which discriminates between them. Each individual's *fitness* is then derived from its cost while taking into account the other individuals in the population.

The distinction between cost and fitness is made here for a reason more important than just interfacing between a minimization problem and the maximization nature of the GA. In fact, cost is something intrinsic to the problem and, therefore, may not be changed at will. Moreover, cost can generally assume values in any ordered set. On the other hand, the cost-to-fitness mapping relates to the selection process, being a customizable part of the optimization strategy. It should be a (broadly) monotonic mapping onto the subset of non-negative reals.

For generality, the *normalized fitness* of a given parent is defined here as the fraction of the population's total number of offspring a given parent is assigned to produce in the next selection step. Since it assumes values in the interval

$[0, 1]$, normalized fitness is sometimes seen as a probability. However, such an interpretation is valid only in the context of particular implementations of the selection procedure, such as sampling with replacement, as discussed later.

Another useful concept is that of *relative fitness*, i.e., the fitness of an individual normalized by the average fitness of the population. Relative fitness provides a direct indication of how much better or worse than the current average individuals are.

While the ordering of the individuals in the population, as determined by their cost, should be preserved, the cost-to-fitness mapping may or may not preserve scale. There are essentially two types of fitness assignment strategies:

Scaling Fitness is computed as a, possibly non-linear, function of the cost values. Care must be taken to ensure (or force!) non-negative fitness for all individuals, while giving the best individual in the population a controlled advantage over the others.

Ranking The mapping is performed by sorting the population by cost, and consequently assigning set fitness values to individuals according to their position in the population, or rank. Scale information is thus deliberately discarded.

Scaling is the more traditional approach. Raw fitness is calculated as a monotonic function of the cost, offset by a certain amount, and then linearly scaled. The first difficulty arises at this stage: whilst scaling aims to preserve the relative performance between different individuals, both the initial transformation and the subsequent offsetting can significantly affect the relative fitness ultimately assigned to each individual.

With scaling, an individual much stronger than all the others may be assigned a very large relative fitness and, through selection, rapidly dominate the population. Conversely, the advantage of the best individual over the rest of the population will be minimal if most individuals perform more or less equally well, and the search will degenerate into an aimless walk.

As none of the two situations above is desirable, the relative fitness assigned by scaling to the best individual in the population must be controlled in some way. Sigma-scaling, for example, calculates the desired amount of offset from the average and standard deviation of the raw fitness values in the population at each generation, so that the relative fitness of the best individual is set more sensibly (Hancock, 1994).

Ranking addresses these same difficulties by eliminating any sensitivity to the scale in which the problem is formulated. Since the best individual in the population is always assigned the same relative fitness, would-be "super" individuals can never reproduce excessively. Similarly, when all individuals perform almost equally well, the best individual is still unequivocally preferred to the rest (but this may be inappropriate if the objective function is contaminated with noise).

Rank-based fitness assignment is characterized by the choice of rank-to-fitness mapping, which is usually chosen to be linear or exponential. For a population of size N, ranking the best individual zero[1] and the worst $N-1$, and representing rank by r and relative fitness by $f = f(r)$, these mappings can be written as follows:

Linear

$$f(r) = s - (s-1) \cdot \frac{2r}{N-1},$$

where s, $1 < s \leq 2$, is the relative fitness desired for the best individual. The upper bound on s arises because fitness must be non-negative for all individuals, while maintaining $\sum_{i=0}^{N-1} f(i) = N$.

[1]This convention is adopted here for consistency with the interpretation of ranking given in Chapter 4.

Exponential

$$f(r) = \rho^r \cdot s\,,$$

where $s > 1$ is the relative fitness desired for the best individual, and ρ is such that $\sum_{i=0}^{N-1} \rho^i = N/s$. Since there is no upper-bound on s, the exponential mapping is somewhat more flexible than the linear.

For $1 < s \leq 2$, the main difference between linear and exponential rank-based fitness assignment is that the exponential mapping does not penalize the worst individuals as much, at the expense of assigning middle individuals fitness slightly less than average. As a consequence, exponential assignment generally contributes to a more diverse search.

2.3.3 Selection

Selection, or replication, is the process of choosing individuals to participate in the production of offspring, proportionally to their fitness. However, since the number of replicates must be an integer, the realized number of offspring will usually need to be higher or lower than the desired value, introducing what is called *selection error*.

In order to avoid systematic selection errors, selection in GAs is usually performed stochastically. The expected number of offspring of an individual should then correspond exactly to its fitness, while *stochastic* selection errors should remain as small as possible. Baker (1987) achieved these two goals with a sampling algorithm he called *Stochastic Universal Sampling* (SUS). SUS was proposed as an alternative to Goldberg's *Roulette Wheel Selection* (Goldberg, 1989) (RWS, or sampling with replacement), which allows fairly large selection errors to occur. RWS consists of a sequence of independent selection trials where the probability of each individual being selected in each trial (as determined by the relative size

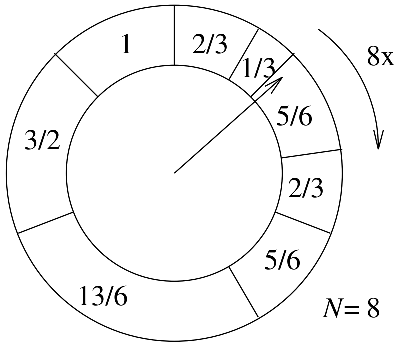


Figure 2.1: Roulette Wheel Selection (or sampling with replacement).

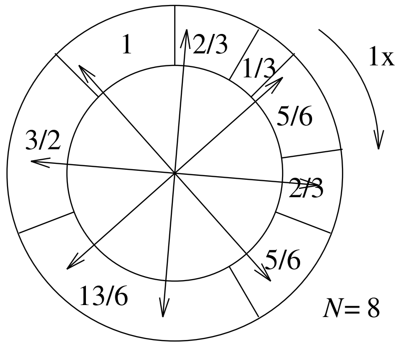


Figure 2.2: Stochastic Universal Sampling.

of the slots on a roulette wheel, see Figure 2.1) remains constant and equal to its normalized fitness.

Selection with SUS may also be visualized as the result of spinning a roulette wheel with slots proportional in width to the fitness of the individuals in the population, but with multiple, equally spaced pointers (Figure 2.2). Once it stops, the number of pointers over each sector must be either the floor or the ceiling of the corresponding desired number of offspring, thus guaranteeing minimum deviations from the desired value, or *spread*. The replicates obtained should be shuffled before the algorithm proceeds with recombination and mutation.

In the extreme case where a single individual is to be selected, SUS is clearly equivalent to RWS. When selecting multiple individuals, the advantage of SUS over RWS in terms of spread increases with the number of offspring to be generated. There is usually no reason to use RWS in these circumstances.

A fairly different approach to selection is known as *tournament selection* (Hancock, 1994). In its simplest form, pairs of individuals are drawn from the population at random, and individuals in each pair compared to each other. The best individual in each pair is declared the winner of that tournament and selected.

Tournament selection is particularly convenient when it is easier to compare individuals than to compute their absolute performance, as is often the case in genetic programming. However, like RWS, selection with random tournaments can lead to large selection errors. When selecting multiple individuals, it should be possible to rank the population given any number of pairwise comparisons (Uppuluri, 1989), or tournaments, and to use SUS to select the offspring given that ranking. The more individuals that are compared, the more accurate the selection process will be.

Tournament selection is similar to ranking in that it does not rely on scale information. Varying the size and number of the tournaments allows the effective relative fitness of the best individual to be controlled, although less flexibly than with ranking. Making the probability of the best individual in a tournament winning that tournament less than one allows the expected relative fitness of the best individual in the population to be controlled (decreased) more smoothly. Deciding tournaments with a probability depending on the relative fitness of the two contestants results in a decrease in fitness differential as the population converges.

2.3.4 Mutation and recombination

Repeated selection from the same population would produce nothing better than multiple copies of the best individual originally in it. For improvement to be able to occur, some novelty must be introduced into the population between selection steps. The *genetic operators* modify the selected parents by manipulating their genotype, and can be divided into two main categories:

Mutation causes individual genotypes to be changed according to some probabilistic rule. Usually, only a small part of the chromosome is changed by mutation, causing offspring to inherit most of the features of the parent. Deterministic “mutation” can also be implemented. It is more commonly known as *repair*.

Recombination causes individuals to exchange genetic information with one another, in pairs or larger groups. Features from two or more parents can thus be inherited by the same offspring.

Mutation operators are generally the simpler of the two categories, but certainly not the least important. Evolution through mutation and selection without recombination is not only possible but can also be very effective. Evolution Strategies, for example, were originally conceived as mutation-selection algorithms. Recombination, on the other hand, makes a different contribution to the search. By bringing into a single individual the genetic features of two or more parents, it can significantly accelerate the search process, especially in the case where decision variables are loosely coupled. For this reason, recombination of selected individuals is usually performed with high probability, typically between 0.6 and 0.9.

The implementation of recombination and mutation necessarily depends on the underlying genotypic representation. Further, the performance of the same operators generally depends on the problem class under consideration. A great deal of research has been put into identifying the best genetic operators for various classes of problems. The development of the Edge Recombination operator for the Travelling Salesman Problem (Whitley *et al.*, 1991) is a good example of this.

A typical recombination operator for binary and other string chromosomes is single-point crossover, whereby two individuals exchange a portion (right or left) of their chromosomes to produce offspring, as illustrated in Figure 2.3. The crossover point is selected at random. Other recombination operators commonly used with binary strings are:

Double-point crossover Two crossover points are selected instead of one (Booker, 1987).

Uniform crossover Each bit is exchanged independently, with a given probability (Syswerda, 1989).

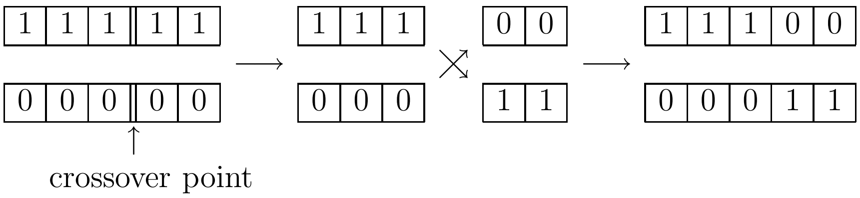


Figure 2.3: Single point crossover.

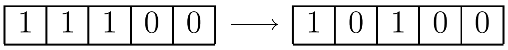


Figure 2.4: Bit mutation.

Shuffle crossover The chromosomes are shuffled before single-point crossover is applied, and consequently deshuffled (Caruana *et al.*, 1989).

Reduced-surrogate crossover The non-identical bits in the chromosomes are first identified, and one of the above crossover types applied to the smaller string thus defined (Booker, 1987). This has the effect of guaranteeing the production of offspring different from their parents.

As for bit mutation (see Figure 2.4), it is most commonly implemented by independently flipping each bit in the chromosome with a given probability. The setting of mutation probabilities will be discussed later in subsection 2.3.8.

2.3.5 Reinsertion

The insertion of offspring back into the population can be made in a variety of ways. In the simplest GAs, for example, the entire population of parents is unconditionally replaced by their offspring. Apart from the assumption of a constant population size, this replacement strategy finds parallel in many natural species, where reproduction takes place seasonally, and parents die before their offspring are born. This is known as generational replacement.

Naturally, there is also interest in GAs where offspring have the chance to compete with at least some of their parents. In the case known as the steady-state, or incremental, GA, a minimal number of offspring (typically one or two) is produced through recombination and mutation. These offspring are then evaluated, and possibly reinserted into the population, replacing

- random members of the parental population.
- the oldest members of the parental population.
- their own parents.
- the least fit members of the parental population.

Actual reinsertion may occur

- unconditionally.
- only if the offspring are fitter than the individuals they are to replace.
- probabilistically, depending on whether or not the offspring are stronger than the individuals they are to replace.

By denying some individuals the possibility of reproducing further, reinsertion has ultimately the same effect as selection. The overall *selective pressure*, or bias towards the best individual, imposed on the population is not only determined by the fitness assignment strategy, but is also affected by when and how reinsertion is performed. In particular, always replacing the least fit individuals in the population strongly increases the *effective*, as opposed to *assigned*, fitness differential between stronger and weaker individuals in the population. This is because, in addition to being less likely to be selected, weaker individuals tend to die earlier, thus participating in less selection trials than stronger ones. Reinsertion strategies which guarantee the preservation of the best individual are known as *elitist*.

The use of different combinations of selection and reinsertion strategies is widely reported in the GA literature. Steady-state GAs, for example, were argued to provide better results than generational ones, allegedly because they exploited newly generated fit individuals earlier than would otherwise be the case. As pointed out by Harvey (1992), this translates into an increase in selective pressure, even if random replacement is used. Replacement based on fitness would provide even greater selective pressure.

The additional selective pressure provided by steady-state reproduction with respect to its generational counterpart may be the simple reason why steady-state GAs seem to perform better in certain cases. Sufficient selective pressure is especially important if large selection errors can occur. The work of Syswerda (1991) reveals no fundamental difference between generational and steady-state GAs with random unconditional replacement. Increasing the selective pressure in a generational GA explicitly through fitness assignment may yield similar results to doing so implicitly through the replacement strategy.

2.3.6 Genetic drift and fitness sharing

There is a clear trade-off between sampling quality, which increases with the number of offspring to be selected simultaneously (with SUS), and the latency inherent in the generational approach, where a single generation step involves many function evaluations. If individuals can be evaluated in parallel, selecting many individuals simultaneously is clearly appropriate, but a steady-state GA might be more desirable when the objective function is computationally intensive and has to be evaluated sequentially.

In the long run, a finite population undergoing stochastic selection will tend to evolve towards a small region of the search space, even if other regions of similarly high fitness exist (for example, there may be more than one local optimum). This normally undesirable phenomenon, known as *genetic drift*, is due to the composition of selection errors over the generations, and can be observed in both natural and artificial evolution.

A concern with sampling accuracy is, therefore, justified. The smaller the population, the more vulnerable it is to selection errors and genetic drift, and the more important it is to ensure an accurate sampling mechanism. This is why a heuristic preference for generational GAs with SUS was expressed earlier. Unfortunately, and although optimally accurate, SUS cannot by itself eliminate genetic drift. As long as populations must remain finite, selection errors are unavoidable.

In these circumstances, it is the accumulation of selection errors that must be controlled in order to prevent genetic drift. Higher mutation rates can achieve this to a certain extent by systematically introducing diversity into the population, but, as discussed below, mutation rates cannot be made arbitrarily high. Similarly, introducing a small percentage of random immigrants into the population at each generation (Grefenstette, 1992) makes the GA more likely to recover information lost through selection and, thus, from genetic drift. A further alternative consists of penalizing the production of individuals similar to each other, for example, by forcing offspring to replace the most similar of their parents, a technique known as crowding (Goldberg, 1989).

A more elaborate technique, known as fitness sharing (Goldberg and Richardson, 1987), models individual competition for finite resources in a closed environment. Similar individuals are weakened by having to share the same resources with each other, while different ones retain their original fitness. As a consequence of fitness sharing, the selection process receives negative feedback discouraging the replication of already abundant individuals, and causing the population to cluster around different local optima in the search space. Such clusters represent favourable regions of the search space, which are known as *niches*.

The main difficulty with fitness sharing lies in the decision of how similar individuals should be in order to begin to decrease each other's fitness. Suitable distance measures can be defined in both genotypic and phenotypic space, but the appropriate threshold σ_{share} has only been estimated based on assumptions regarding the location and probable number of local optima in the search space, both of which are generally unknown (Deb and Goldberg, 1989).

2.3.7 The viability of mating and mating restriction

As the population splits into groups to populate different niches, the ability of recombination to produce fit offspring from individuals in such different groups may decrease. Indeed, it often leads to the creation of offspring outside both favourable regions (Deb and Goldberg, 1989). Such unfit individuals are usually referred to as *lethals*.

Following the clue from Nature, where mating tends to occur only between reasonably similar individuals (those of the same species), a mating restriction scheme can be added to the GA in order to reduce the formation of lethals, as proposed by Deb and Goldberg (1989). Again, a measure of distance between individuals must be defined.

Although constituting an improvement on random mating, mating restriction based on a distance measure does not address the whole issue of mate selection. In particular, the choice of how similar individuals should be in order to mate (σ_{mate}) is just as difficult to make as it is to set σ_{share} (in practice, σ_{share} and σ_{mate} are usually assigned the same value). Sexual mate choice, for example, also seems relevant in this context, but only very exploratory studies have been carried out so far (Miller, 1994).

The success of recombination depends not only on how different the mating parents are, but also, and possibly more so, on the interaction between chromosome coding, the recombination operator, and the structure of the cost landscape. Manderick *et al.* (1991) has shown that, for certain types of cost landscapes, the performance of different recombination operators can be related to statistical properties of the cost surface itself. The question of how recombination and/or chromosome coding can be evolved concurrently with the search is a natural one, for which there are to date no definite answers.

2.3.8 Mutation rates and selective pressure

The "optimal" setting of mutation and recombination rates was probably one of the first concerns raised by the use of GAs in optimization. Initial studies, of an experimental nature, tried to find settings which "worked well" for a number of test functions. Most of those studies concentrated on binary chromosomes, where mutation rate is clearly defined as the probability of each bit being independently inverted by mutation. Nevertheless, some results apply also to more general chromosomes.

The "large" mutation rates proposed by some authors raised controversy over the effective rôle of mutation in genetic search, which was considered secondary by others. Nowadays, mutation is better understood, and recognized to be an important, active search operator. The setting of mutation rates has been shown to depend on a number of other aspects of the GA, namely:

Selective pressure In the absence of recombination, the probability of individuals not undergoing mutation should be such that at least one exact copy of the best individual in each generation can expect to survive beyond its parent's death. This setting, suggested by the work of Schuster (see Harvey (1992)), maximizes exploration of the search space by the population while guaranteeing the exploitation of the best individual. In fact, higher mutation rates would quickly make the search degenerate into a random walk, because selection would no longer be able to recover from the genetic errors introduced by mutation. This abrupt change in the qualitative behaviour of the GA as the mutation rate increases is known as *error catastrophe*. Lower mutation rates than the corresponding *error threshold* more surely preserve the best individual, but also make the search less diverse.

Chromosome length The above considerations raise an important question in the case of elitist GAs, for example. Since the best individual is never lost in such cases, it would appear that offspring could be made arbitrarily different from their parents. This is clearly not the case, as it would be equivalent to pure random search. So, how much of the chromosome should mutation actually change?

Mutation rates for string chromosomes are usually set to about the inverse of the chromosome length, which has been shown theoretically to be optimal (under restrictions) (Bäck, 1993), and observed experimentally to perform well on a wide range of problems.

Distance to the optimum As noted by Bäck (1993), the farther away from the optimum an individual is, the higher the mutation rate must be in order to minimize the average distance of the corresponding offspring to that optimum. This has the strong implication that mutation rates should, in principle, not remain constant throughout a GA run. Although it has been observed that a variation of mutation rates over time may accelerate optimization, such practice does not appear to be well established. As Bäck points out, the standard mutation rate $1/\ell$, where ℓ is the chromosome length, is close enough to what could be achieved with optimally time-varying mutation rates.

In practice, mutation rates set according to the above guidelines are usually acceptable. In Chapter 5, the setting of the mutation rate of a binary, generational GA is described in detail.

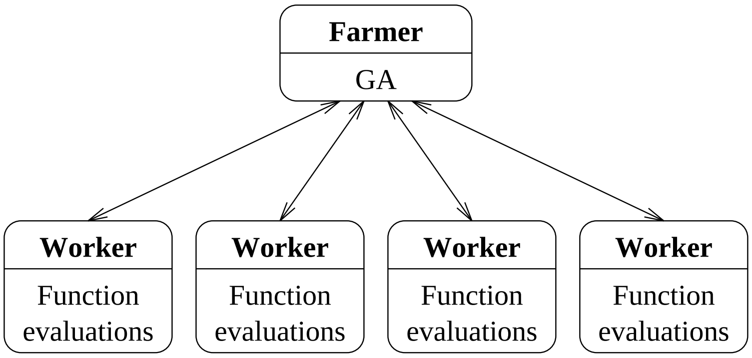


Figure 2.5: Farmer-worker implementation of the GA.

2.3.9 Parallel implementation and structured populations

Although they tend to be computationally demanding, genetic algorithms are also very amenable to parallel implementation. As already suggested in subsection 2.3.6, the evaluation stage of a generational GA may be performed in parallel. Assuming the GA itself is run on a single processor, offspring produced at each generation can simply be farmed out for evaluation on other processors, as illustrated in Figure 2.5. This approach is especially appropriate when the amount of computation required for each evaluation is high compared with that of the GA itself and with the amount of data which needs to be communicated between processors per evaluation. Incremental GAs may also be parallelized in this way, but the number of offspring produced at a time should equal the number of processors available.

So far, it has been implicitly assumed that all individuals take the same amount of computation time to evaluate. When this is not the case, a synchronous (generational or incremental) implementation may not utilize the processors available to the full. However, GAs may also be implemented asynchronously: following the evaluation of the whole population, a new individual is selected for evaluation every time a processor completes the evaluation of a previous one, which is then reinserted into the population. As a side effect, this approach is

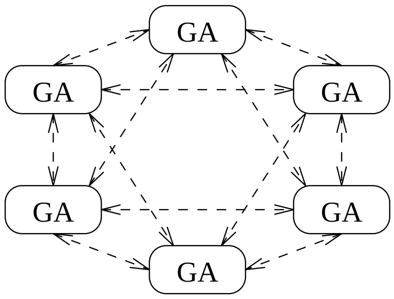


Figure 2.6: The island model.

actually biased towards individuals less expensive to evaluate, as these tend to participate in selection more often. This may, or may not, be desirable.

The above approaches are centred around the "GA farmer", and do not appear as very accurate models of natural evolution. In such *panmictic* GAs, individuals are allowed to mate, but also forced to compete, with every other individual in the population. In Nature, however, geographical isolation implies that individuals in one continent, for example, never get a chance to interact with individuals in another continent, possibly with a few exceptions. Populations are therefore *structured*, and evolution is conditioned by that structure.

In the *island-model*, several GAs are run independently, e.g., on different processors. In addition, these GAs periodically exchange small fractions of their populations, implementing *migration* (see Figure 2.6). In the *diffusion model*, the population is distributed over a grid, and each individual interacts only with its direct neighbours (see Figure 2.7).

The paradigm of geographical isolation makes parallelization even more attractive, as it cuts down inter-processor communication, but it does not require implementation on a parallel architecture. More interestingly, genetic algorithms based on structured populations seem to exhibit even better global search properties than their panmictic counterparts, which, by itself, justifies their sequential implementation as well. For simplicity, however, only panmictic GAs will be considered in the present work.

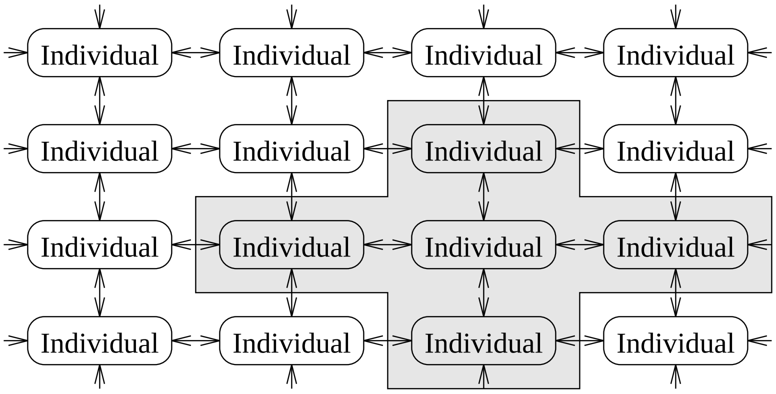


Figure 2.7: The diffusion model.

2.4 Genetic optimizers for control engineering

An ideal, Utopian, optimizer would find the global optimum of an arbitrary cost surface with a minimum of function evaluations. Not too surprisingly, such an optimizer does not exist. Instead, there is always a trade-off between how general and how efficient different methods are. The field of numeric optimization provides many examples of this trade-off, a few of which are:

Linear programming Aimed at, and only at, problems involving only linear functions.

Gradient-based non-linear optimization methods For problems involving non-linear functions. The cost function should be unimodal, and also be continuous and differentiable, at least in the neighbourhood of the optimum. Gradient-based methods also work for linear problems, but linear programming would be much more efficient in that case.

Simplex-based non-linear optimization methods Do not require the cost function to be continuous or differentiable. However, there should still be a single local optimum and no regions of flat cost. These methods generally perform worse than those based on gradients if applied to smooth continuous functions.

Random search Is the most general, and also the weakest of all optimization methods. It can be applied as long as the search domain can be defined and the cost function evaluated. It will, in the limit, find the global optimum of a multimodal cost function, but very inefficiently.

Genetic algorithms could be placed somewhere in between the two last items. Like simplex-based methods, GAs only require that cost be defined at each point in the search space. Also, GAs work best when a single local optimum exists *in genotypic space*, where the concept of neighbourhood depends on the genetic operators (Reeves, 1994). Curiously enough, simplex methods also maintain a "population" of points, the simplex, from which new points are generated according to given rules. Despite having been developed independently, genetic algorithms can thus be seen as an extension of the simplex approach. Genetic algorithms maintain an arbitrarily large population and are based on probabilistic transition rules, rather than deterministic transition rules as in the simplex method (Walsh, 1975).

The power of the many conventional optimization methods in areas where they find application must be recognized. Similarly, any available knowledge about a function's behaviour should affect the formulation of a particular GA (Davis, 1991), while bearing in mind that generality is lost as more and more domain specific knowledge is incorporated (Michalewicz, 1993). Such knowledge is often available for control engineering problems.

A number of issues arising commonly in control systems engineering problems, and the way in which these can be addressed through the GA approach, is discussed below (Fleming and Fonseca, 1993).

Discrete decision variables Discrete decision variables can be handled directly through standard binary, or even n-ary, encoding. When functions can be expected to be locally monotonic with respect to such variables, the use of

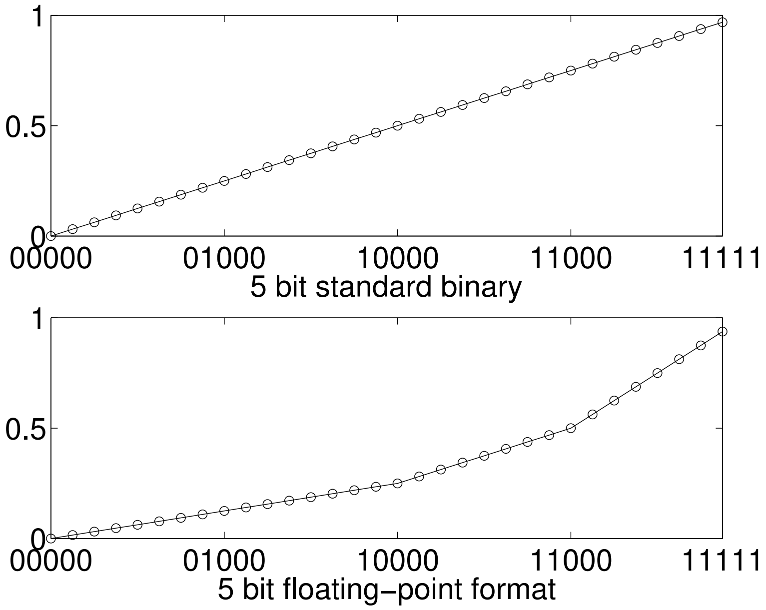


Figure 2.8: A convenient floating point binary format grows in the same fashion as standard binary.

Gray coding is known to exploit that monotonicity better than standard binary, assuming the traditional genetic operators described earlier.

Continuous decision variables Real values can be approximated to the necessary degree by using a fixed point binary representation. However, in most control problems, the relative precision of the parameters is more important than their absolute precision. In this case, the logarithm of the parameter should be encoded instead. Alternatively, a floating point binary representation can also be used directly.

The previous consideration about Gray coding applies to both fixed and floating point formats. In fact, a convenient floating point format can be shown to grow in the same fashion as standard binary (Figure 2.8), and can therefore be Gray encoded.

The direct manipulation of real-valued chromosomes, as in Evolution Strategies, has also been proposed (Davis, 1991). While this approach is preferred by many practitioners, mainly for its better local convergence properties, it has been severely criticized in theory (Goldberg, 1991). More recently, real-coded GAs have received theoretical support mainly from the German school (Mühlenbein and Schlierkamp-Voosen, 1993).

In Chapters 4 and 5, continuous decision variables are binary (Gray) encoded for simplicity. Although this encoding is shown to possess a number of limitations in the context of those chapters, it is unclear whether a real-encoding would overcome those limitations. While it is clearly related to the present work, the study of the "representation problem" lies largely outside its scope.

Non-numeric decision variables There are also problems in control engineering where some decision variables are simply non-numeric. In controller design, for example, the structure of the controller is a decision variable itself. Such problems can be accommodated by devising suitable representation schemes and appropriate genetic operators. Chapter 6 provides an example of how GAs can be used to select model structures in non-linear system identification, by choosing an appropriate representation and operators.

Multimodality Multimodal functions may be particularly difficult for conventional optimizers. GAs are more robust, but they are still not guaranteed to find the global optimum. Multimodality is usually addressed with niche induction techniques such as sharing, mating restriction, and crowding, as discussed in subsections 2.3.6 and 2.3.7. These encourage the GA to search many optima simultaneously, thereby increasing the probability of finding the global one. Genetic algorithms based on structured populations have been shown to be more robust to multimodality than conventional, panmictic GAs (Collins and Jefferson, 1991).

Constraints Most engineering optimization problems are constrained in some way. For example, control loops are required to be stable and actuators have finite ranges of operation. Constraints have been handled mainly in two ways, the most efficient of which usually consists of embedding them in the coding of the chromosomes. When this is not possible or easy, the fitness assignment process should reflect the fact that invalid individuals represent worse alternatives than valid individuals. Additive penalty functions, often used with GAs, are known to be very problem dependent and, thus, difficult to set (Richardson *et al.*, 1989). More recently, a more general approach based on ranking has been proposed (Powell and Skolnick, 1993).

Multiple objectives Finally, control engineering problems very seldom require the optimization of a single objective function. Instead, there are often competing objectives which should be optimized simultaneously. The potential of GAs to become a powerful method for multiobjective optimization has long been recognized (Schaffer, 1985), namely concerning the concurrent finding of multiple solutions illustrative of the trade-offs present in the problem, made possible by a population-based search.

Despite initial interest, contributions in the area have remained scarce. Multiobjective optimization with GAs was recognized as non-trivial by Richardson *et al.* (1989), who reported success on the set covering problem, but experienced "difficulties in extending the approach to other classes of problems".

The next section briefly reviews examples of the application of GAs to a number of problems in the control field.

2.5 Control applications of GAs

Broadly, the application of GAs to control engineering problems may be classified into two main areas: off-line design and on-line adaptation, learning and optimization. In CACSD, the GA is run off-line as an optimizer, where optimal individuals evolve from a random population. In on-line applications, however, the direct evaluation (through experimentation) of weak individuals may have catastrophic consequences. Therefore, it is a common practice for a GA to operate on a model of the system on-line and only to indirectly influence the control operation.

Some examples of off-line and on-line control applications are described next.

2.5.1 Optimal controller tuning

The tuning of controllers is a classic optimization problem for control engineers and can be easily addressed using a GA. Hunt (1992a; 1992b; 1992c) applied it to four classes of problems, namely LQG, $\mathcal{H}_\infty$ minimum and mixed sensitivity problems, and frequency domain optimization.

In each case, a simple, binary-coded, genetic algorithm was used to evolve sets of controller parameters for a given controller structure. Individuals were assessed by directly evaluating the cost function corresponding to the problem. Non-stabilizing controllers, for which no cost was defined, were assigned an extremely low fitness. The stability constraint was thus incorporated in the GA.

2.5.2 Robust stability analysis

One approach to the stability analysis of systems in the presence of uncertain parameters involves searching a neighbourhood of the nominal system in parameter space, the so-called Q-box, for points which yield unstable systems. Murdock *et al.* (1991) proposed the genetic search of the Q-box, formulating it as the maximization of the maximum root of the system within given parametric uncertainties. This objective function presents significant difficulties to conventional optimization approaches, for its possible multimodality and/or non-differentiability.

The GA was applied to a number of benchmark examples in order to verify its performance. It reportedly was able to handle large numbers of parameters while producing the correct results, and being computationally more efficient than the other existing methods. Its application to problems which lack a necessary and sufficient condition test, or the complexity of which makes other methods impractical, was suggested.

2.5.3 System identification

Kristinsson and Dumont (1992) applied genetic algorithms to the on-line, as well as off-line, identification of both discrete and continuous systems. In their implementation, windowed input and output data was used to construct an objective (cost) function of the parameters to be identified, which could either be the system parameters in their natural form, poles and zeros, or any convenient set of transformed variables.

Since the real system parameters would be expected to minimize such an objective function for any set of input-output data, the GA saw an environment which, although varying in time due to the diversity of the data sets, maintained its optimum in the same region of the search space. In this way, only the best estimate found for a particular set of data was needed to adaptively control the system. The evaluation of individuals other than the best in the GA population was of no consequence to the control loop.

Note that, as the predictive quality of the best identified model was known before it came to affect the control loop, it would always be possible to default to a "safe" controller in the case of unacceptable GA performance.

2.5.4 Classifier systems

Classifier systems (CSs, see Goldberg (1989)) are machine learning systems that encode production rules in string form (classifiers) and learn such rules on-line, during their interaction with an arbitrary environment. Each rule has the structure

$$< condition >:< action > ,$$

which means that the action may be taken when the condition is satisfied (matched).

They are parallel, message passing, rule-based systems which use GAs. The rule system is one of the three main components of a CS. It communicates with the outside world through a message list. Messages may cause an action to be taken, or another message to be produced by matching one or more classifiers in the classifier store.

The second component of a CS is the apportionment of credit system, which associates a strength with each classifier. The right a classifier has to respond to a particular message depends on its strength. The strength is reduced by a certain amount every time the classifier is activated, that amount being paid to whatever (classifier or environment) was responsible for its activation. Classifiers may also receive reward from the environment. In this way, rules which contribute toward good actions being taken tend to see their strengths increased.

The third component of a CS is the GA which is in charge of discovering new, possibly better, rules. While selection of good rules is provided through apportionment of credit, bad rules may be replaced by combinations of good ones, and their performance evaluated on-line.

While it is unsafe to evolve rules for complex systems on-line using a simple GA, its successful incorporation in a rule-based learning system has been reported in connection with the optimization of combustion in multiple burner installations (Fogarty, 1990). Other hybridized techniques involve fuzzy control.

2.5.5 Fuzzy control

Linkens and Nyongesa (1992) identify four main aspects of fuzzy controller design:

- selection of scaling factors,
- derivation of optimal membership functions,
- elicitation of a rule-base, and
- the on-line modification of the rule-base.

The optimization of membership functions is seen as an off-line task, while a fuzzy classifier system is used to acquire and modify good sets of rules on-line. Adaptive control is provided by the constant adaptation of the rule set to the varying dynamics of the problem, without the explicit identification of a plant model.

On the other hand, Karr (1991; 1992) concentrates all of the adaptation effort on the on-line optimization of the membership functions. It is argued that the rules-of-thumb humans use tend to remain the same, even across a wide range of conditions. It is the definition of the linguistic terms, here the membership functions, which is adapted to the present situation.

This concept has been implemented together with another GA-based system, an analysis (identification) element, to control a laboratory pH system of varying dynamics. While the analysis GA provided the parameters of a model of the process being controlled, membership functions were optimized using such a model as a basis for their evolution. The current best set of membership functions was then used to actually control the plant.

2.5.6 Robot trajectory planning

The automatic generation of robot trajectories (Solano, 1992) is an order-dependent, generally multimodal, problem, where candidate solutions are typically of variable length. A GA approach to this problem (Davidor, 1991a) required the introduction of specialized genetic operators (Davidor, 1989) and was then improved through the use of Lamarckian learning, whereby also acquired characteristics are passed on to offspring, to bias the action of the genetic operators. Although Lamarckism is generally accepted not to take part in natural evolution, there is no reason why it should not be used in artificial evolution, when able to improve it.

Comparison with hill-climbing shows the ability of the GA to find good trajectories, balancing an eventually slower rate of convergence with a much better global behaviour.

2.6 Concluding remarks

Genetic algorithms, with their broad domain of application, have considerably extended the scope of optimization in engineering. This expansion has been mirrored by the diversity of control applications which have arisen in the past few years. The current availability of parallel computing power, which GAs can effectively exploit, reinforces the viability of the evolutionary approach to optimization.

Genetic algorithms are particularly well suited to design problems difficult to address by other methods, namely problems involving variables of mixed types and/or ill-defined objectives. On-line applications, however, may present significant challenges. When a system model is available, or can be identified, a conventional GA can be applied on-line with little alteration. If this is not the case, the nature of the GA search demands that the system possess a rare degree of robustness to sustain the level of exploration demanded by a typical GA. Alternatively, classifier systems provide an approach to the control of plants which cannot be modelled.

Among the many application areas of GAs, multiobjective optimization stands out as one of the most promising and, simultaneously, one of the least explored in the literature. While the relevance of multiobjective optimization in engineering design cannot be denied, most current multiobjective approaches resort to the optimization of an analytical combination of the objectives by conventional (single-objective) means. While combining the objectives in a meaningful way prior to optimization can be generally difficult, this difficulty is further aggravated whenever there is a need for the resulting cost function to comply with requirements common of conventional non-linear optimizers, such as continuity and differentiability.

By alleviating the demands on the formulation of the cost function, genetic algorithms should make it possible for decision models other than analytical aggregating functions to be used for fitness assignment. Also, GA populations should offer a means of concurrently exploring alternative solutions which cannot be adequately discriminated in the absence of additional preference information.

The next Chapter introduces the multiobjective optimization concepts necessary to the consequent development of genetic algorithm approaches to multiobjective optimization.

Chapter 3

Multiobjective and Constrained Optimization

3.1 Introduction

In Chapter 2, multiobjective optimization was identified as an area of GA development deserving closer attention. The purpose of the present Chapter is to provide the necessary background for the work on multiobjective genetic algorithms to be described later.

Some of the basic concepts in multiobjective optimization are presented in Section 3.2, which also explains the need for decision making and describes some common approaches to preference articulation. Constrained optimization is introduced in Section 3.3, without unnecessarily abandoning the multiobjective case. The relationships between constrained and multiobjective optimization will be explored further in Chapter 4.

Section 3.4 is concerned with the visualization of approximate non-dominated solutions, as produced by a general multiobjective optimizer. In particular, a novel technique for the visualization of results from multiple runs is developed in subsections 3.4.2 and 3.4.3, so that insight into the statistical behaviour of an optimizer can be gained. This will be useful later, in Chapters 5 and 6.

3.2 Multiobjective optimization

Practical problems are often characterized by several non-commensurable and often competing measures of performance, or objectives. The multiobjective optimization problem is, without loss of generality, the problem of simultaneously minimizing the n components f_k, $k = 1, \ldots, n$, of a possibly non-linear vector function $\mathbf{f}$ of a general decision variable x in a universe $\mathcal{U}$, where

$$\mathbf{f}(\mathsf{x}) = (f_1(\mathsf{x}), \ldots, f_n(\mathsf{x})).$$

The problem usually has no unique, perfect (or Utopian) solution, but a set of non-dominated, alternative solutions, known as the Pareto-optimal set (Ben-Tal, 1980). Assuming a minimization problem, dominance is defined as follows:

Definition 3.1 (Pareto dominance) *A vector* $\mathbf{u} = (u_1, \ldots, u_n)$ *is said to dominate* $\mathbf{v} = (v_1, \ldots, v_n)$ *if and only if* $\mathbf{u}$ *is partially less than* $\mathbf{v}$ *(*$\mathbf{u} \;{}_{\mathrm{P}}\!\!< \mathbf{v}$*), i.e.,*

$$\forall\, i \in \{1, \ldots, n\}\,,\ \ u_i \leq v_i \quad \wedge \quad \exists\, i \in \{1, \ldots, n\}:\ u_i < v_i\,.$$

Definition 3.2 (Pareto optimality) *A solution* $\mathsf{x}_{\mathrm{u}} \in \mathcal{U}$ *is said to be Pareto-optimal if and only if there is no* $\mathsf{x}_{\mathrm{v}} \in \mathcal{U}$ *for which* $\mathbf{v} = \mathbf{f}(\mathsf{x}_{\mathrm{v}}) = (v_1, \ldots, v_n)$ dominates $\mathbf{u} = \mathbf{f}(\mathsf{x}_{\mathrm{u}}) = (u_1, \ldots, u_n)$.

Pareto-optimal solutions are also called efficient, non-dominated, and non-inferior solutions. The corresponding objective vectors are simply called non-dominated. The set of all non-dominated vectors is known as the non-dominated set, or the trade-off surface, of the problem.

Pareto-optimality can be illustrated graphically by considering the set of all feasible objective values, i.e., the set of all points in objective space corresponding

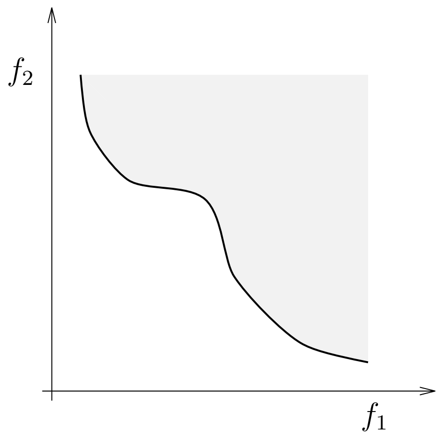


Figure 3.1: Graphical interpretation of the non-dominated set.

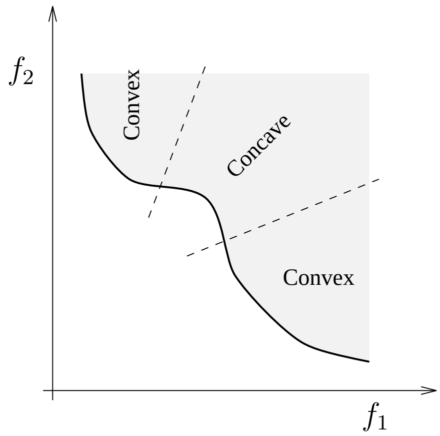


Figure 3.2: Convex and concave regions of a trade-off surface.

to at least one setting of the decision variable. The shaded area in Figure 3.1 represents such a set, for a possible bi-objective problem. The non-dominated set of that problem is composed of the points lying along the "south-west" boundary of the shaded area (the solid line).

Figure 3.1 also serves to illustrate the notion of convexity of a non-dominated set, which can be formally defined as follows (again, minimization of the objectives is assumed):

Definition 3.3 (Convexity) *A non-dominated set $\mathcal{P}$ is said to be convex if and only if*

$$\forall\, \mathbf{u}, \mathbf{v} \in \mathcal{P}\,,\ \ \forall\, \gamma \in (0,1)\,,\ \ \exists\, \mathbf{w} \in \mathcal{P}:\ \ \mathbf{w} \leq \gamma\mathbf{u} + (1-\gamma)\mathbf{v}\,,$$

where the inequality applies componentwise. Otherwise, $\mathcal{P}$ is said to be non-convex.

Similarly, for concavity:

Definition 3.4 (Concavity) *A non-dominated set $\mathcal{P}$ is said to be concave if and only if*

$$\forall\, \mathbf{u}, \mathbf{v} \in \mathcal{P}\,,\ \ \forall\, \gamma \in (0,1)\,,\ \ \exists\, \mathbf{w} \in \mathcal{P}:\ \ \gamma\mathbf{u} + (1-\gamma)\mathbf{v} \;{}_{\mathrm{P}}\!\!< \mathbf{w}\,.$$

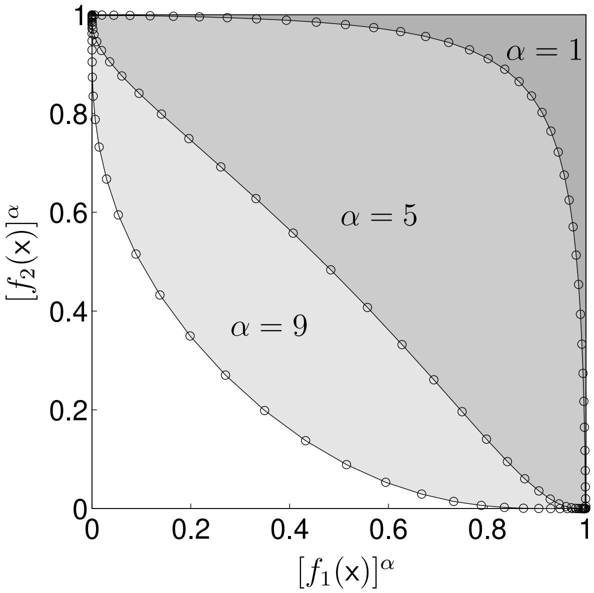


Figure 3.3: The concavity of the trade-off set is related to how the objectives are scaled.

In general, trade-off surfaces may be neither convex nor concave. However, *regions* of local convexity and local concavity can usually be identified in such trade-off surfaces. The convex and concave regions of the trade-off surface in Figure 3.1 are indicated in Figure 3.2.

The convexity of a trade-off surface depends on how the objectives are scaled. Non-linearly rescaling the objective values may convert a non-convex surface into a convex one, and vice-versa, as illustrated in Figure 3.3. The darker surface is the original, non-convex trade-off surface, corresponding to plotting $f_1(\mathsf{x})$ against $f_2(\mathsf{x})$. The lighter surfaces correspond to plotting $[f_1(\mathsf{x})]^\alpha$ against $[f_2(\mathsf{x})]^\alpha$, for $\alpha =$ 5 and $\alpha = 9$, the latter being clearly convex. Nevertheless, all are formulations of essentially the same Pareto-minimization problem which admit exactly the same solution set in decision variable space.

3.2.1 Multiobjective optimization and decision making

The notion of Pareto-optimality is only a first step towards solving a multiobjective optimization problem. The choice of a suitable compromise solution from all non-inferior alternatives is not only problem-dependent, it generally depends also on the subjective preferences of a decision agent, the Decision Maker (DM). Thus, the final solution to the problem is the result of both an *optimization* process and a *decision* process.

According to how the optimization and the decision processes are combined in the search for compromise solutions, three broad classes of multiobjective optimization methods can be identified (Hwang and Masud, 1979):

A priori articulation of preferences The Decision Maker expresses preferences by combining the different objectives into a scalar cost function, ultimately making the problem single-objective prior to optimization.

A posteriori articulation of preferences The Decision Maker is presented by the optimizer with a set of candidate non-inferior solutions, before expressing any preferences. The compromise solution is chosen from that set.

Progressive articulation of preferences Decision making and optimization occur at interleaved steps. At each step, partial preference information is supplied to the optimizer by the Decision Maker, which, in turn, generates better alternatives according to the information received.

Methods for a priori articulation of preferences have the advantage of requiring no further interaction with the DM. For that reason, computation effort can be concentrated on the production of the *final* solution. However, if the solution found cannot be accepted as a good compromise, new runs of the optimizer on modified cost functions may be needed, until a suitable solution is found. These methods also have the disadvantage of requiring new runs of the optimizer every time the preferences of the DM change.

Methods for a posteriori articulation of preferences address these difficulties at the possible expense of computation time. Since no a priori knowledge is assumed, the whole trade-off surface must be carefully sampled. These methods are especially well-suited to problems involving static objectives, but where the preferences of the DM change relatively often. In such cases, preference changes imply no further computation.

Progressive articulation of preferences establishes a compromise between these two classes of methods, but now at the expense of increased interaction between the DM and the optimizer. Especially in the case of a human DM, the need for constant human intervention may be actually a disadvantage, depending on the relative cost associated with computation and labour.

3.2.2 Preference articulation

Regardless of the stage at which it takes place, preference articulation defines a cost function which discriminates between candidate solutions.[1] Although such a cost function can be difficult to formalize in every detail, approaches based on the following are widely used.

3.2.2.1 Weighting coefficients

Weighting coefficients are real values which express the relative importance of the objectives and control their involvement in the overall cost measure. The weighted-sum approach is the classical example of a method based on objective weighting (Hwang and Masud, 1979). As the name suggests, the cost function is formulated as a weighted sum of the objectives:

$$\mathcal{F}_{\mathrm{ws}}(\mathsf{x}) = \sum_{i=1}^{n} w_i \, f_i(\mathsf{x})$$

where n is the number of objectives, and the weights w_i are positive constants. It can be easily shown (Goicoechea *et al.*, 1982, p. 46) that, for any set of positive

[1]When maximization is assumed, the cost function is referred to as *utility function*.

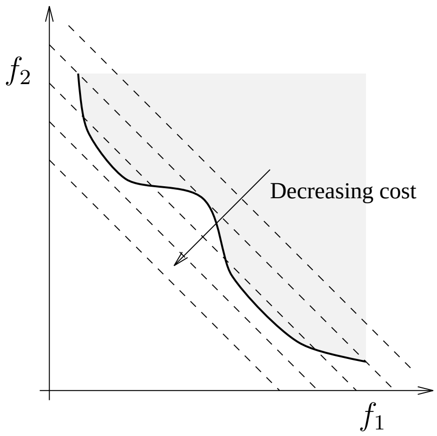


Figure 3.4: Lines of equal cost induced by the weighted-sum approach.

weights, the (global) optimum of $\mathcal{F}_{\text{ws}}(\mathsf{x})$ is always a non-dominated solution of the original multiobjective problem. However, the opposite is not true. For example, non-dominated solutions located in concave regions of the trade-off surface cannot be obtained by this method, because their cost is sub-optimal (Fleming and Pashkevich, 1985) (see Figure 3.4).

Another disadvantage of the weighted-sum approach is that it can be particularly sensitive to the setting of the weights, depending on the problem.

3.2.2.2 Priorities

Priorities are integer values which determine in which order objectives are to be optimized, according to their importance. The lexicographic method (Ben-Tal, 1980), for example, starts by assigning a different priority to each objective. Given n objectives $f_1, \ldots, f_n$, and without loss of generality, each objective f_i, $i = 1, \ldots, n$, is assigned priority i, where greater values of i correspond to higher priorities. The underlying cost function $\mathcal{F}_{\text{lex}}$ is such that

$$\begin{aligned}&\mathcal{F}_{\text{lex}}(\mathsf{x}) < \mathcal{F}_{\text{lex}}(\mathsf{y}) \;\Leftrightarrow \\ &\quad \exists\, i \in \{1, \ldots, n\} : \;\forall\, j \in \{i, \ldots, n\}\,, \;\; f_j(\mathsf{x}) \le f_j(\mathsf{y}) \;\wedge\; f_i(\mathsf{x}) < f_i(\mathsf{y})\,.\end{aligned}$$

In practice, the objective with the highest priority, f_n, is minimized first, and the solution recorded (say x_n). Then, f_{n-1} is minimized subject to $f_n(\mathsf{x}) \leq f_n(\mathsf{x}_n)$, and a new solution found (x_{n-1}). The third step consists of minimizing f_{n-2} subject to $f_k(\mathsf{x}) \leq f_k(\mathsf{x}_k)$, for $k = n-1, n$, and so forth. The final solution, x_1, is taken as the solution of the problem. It is also a Pareto-optimal solution.

The lexicographic method will generally produce different non-dominated solutions depending on what priorities are assigned to the objectives. Although it models well those practical situations in which decisions are made sequentially, it does assume that the priority of the various objectives can be clearly identified, which is not always the case.

3.2.2.3 Goals

Goal values indicate desired levels of performance in each objective dimension. The way in which goals are interpreted may vary. In particular, they may represent levels of performance to be approximated and, if possible, improved upon, or ideal performance levels to be matched as closely as possible (Dinkelbach, 1980).

The minimax approach is an example of the first interpretation. It consists of minimizing the cost function

$$\mathcal{F}_{\mathrm{mM}}(\mathsf{x}) = \max_{i \in \{1,\ldots,n\}} \left\{ \frac{f_i(\mathsf{x}) - g_i}{w_i} \right\}$$

where the constants g_i are the goals to be attained, and w_i are positive weights which indicate the desired direction of search in objective space.

Although this approach is able to produce solutions in concave regions of the trade-off surface (see Figure 3.5), it is not guaranteed to always produce strictly non-dominated solutions. In addition, $\mathcal{F}_{\mathrm{mM}}$ tends not to be differentiable at the optimum, which makes direct gradient-based optimization difficult. For this reason, alternative formulations are usually preferred.

The goal attainment method (Gembicki, 1974) avoids the latter difficulty by

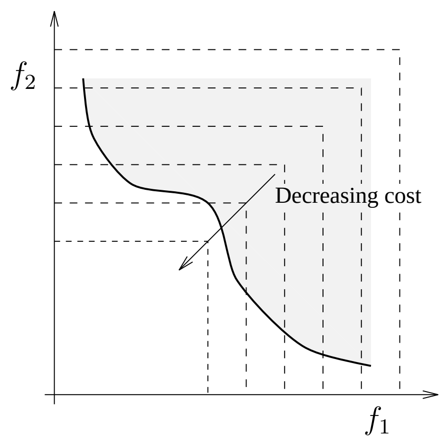


Figure 3.5: Lines of equal cost induced by the minimax approach.

introducing an auxiliary scalar parameter λ and solving:

$$\min_{\lambda,\mathsf{x}} \lambda$$

subject to

$$f_i(\mathsf{x}) - w_i\lambda \leq g_i\,,$$

where g_i and w_i have the same meaning as in the minimax approach. The quantity $w_i\lambda$ indicates the amount by which the solution under- or over-attains each goal.

3.3 Constrained optimization

The solution of a practical problem may be constrained by a number of restrictions imposed on the decision variable. Constraints usually fall into one of two different categories:

Domain constraints express the domain of definition of the objective function. In control systems, closed-loop system stability can be given as an example of a domain constraint, because most performance measures are not defined for unstable systems.

Preference constraints impose further restrictions on the solution of the problem according to knowledge at a higher level. A given stability margin, for example, expresses a (subjective) preference of the designer.

Constraints can usually be expressed in terms of function inequalities of the type

$$f(\mathsf{x}) \leq g \, ,$$

where f is a real-valued function of a variable x, and g is again a constant value. The inequality may also be strict ($<$ instead of $\leq$). Equality constraints of the type

$$f(\mathsf{x}) = g$$

can be formulated as particular cases of inequality constraints, e.g.,

$$g - \delta_1 \leq f(\mathsf{x}) \leq g + \delta_2$$

for arbitrarily small, non-negative, δ_1 and δ_2.

Without loss of generality, the constrained optimization problem is that of minimizing a multiobjective function $(f_1, \ldots, f_k)$ of some generic decision variable x in a universe $\mathcal{U}$, subject to a positive number $n - k$ of conditions involving x and eventually expressed as a functional vector inequality of the type

$$(f_{k+1}(\mathsf{x}), \ldots, f_n(\mathsf{x})) \leq (g_{k+1}, \ldots, g_n) \, ,$$

where the inequality applies componentwise. It is implicitly assumed that there is at least one point in $\mathcal{U}$ which satisfies all constraints, although in practice that cannot always be guaranteed. When constraints cannot be all simultaneously satisfied, the problem is often deemed to admit no solution as it stands. The number of constraints violated, and the extent to which each constraint is violated, are then taken into account in order to possibly relax the preference constraints.

Constraints can be seen as high-priority (or hard) objectives, which must be jointly satisfied before the optimization of the remaining, soft objectives takes place. Satisfying a number of violated inequality constraints is clearly the multiobjective problem of minimizing the associated functions until given values (goals) are reached. The concept of non-inferiority is, therefore, readily applicable, and even particularly appropriate when constraints are themselves non-commensurable. When not all goals can be simultaneously met, a *family* of violating, non-inferior points is the closest to a solution for the problem.

On the other hand, problems characterized by soft-objectives only are often reformulated as (single objective) constrained optimization problems in order to be solved. The goal-attainment method, described earlier, and the ε-constraint method (Goicoechea *et al.*, 1982, p. 55ff) are good examples of this.

3.4 Visualization and performance assessment issues

The remainder of this chapter will be concerned with the graphical visualization of trade-off data, i.e., of one or more sets of relatively non-dominated solutions produced by as many runs of a general multiobjective optimizer.

When the aim is to convey to the human decision maker information concerning the best trade-off known as a consequence of the data, the problem reduces to the visualization of a single set of overall non-dominated solutions. Two graphical representation methods commonly used for this purpose will be described in subsection 3.4.1, one for two objectives, and another for three or more objectives.

On the other hand, in order to gain insight into how well an optimizer *can be expected to perform* on a given problem, data from multiple optimization runs must be considered in its entirety, i.e., it is not sufficient to consider only the overall non-dominated solutions. For this reason, a method for the visualization

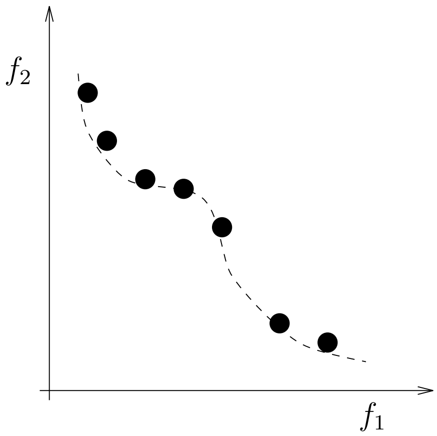


Figure 3.6: Visualization of trade-off data in two dimensions.

of the "distribution" of the outcome of various multiobjective optimization runs is developed in subsection 3.4.2, and the corresponding statistical interpretation given in subsection 3.4.3. The approach is further clarified by means of examples.

3.4.1 Visualization of trade-off data from a single run

Methods for progressive and a posteriori articulation of preferences require that trade-off information be communicated to the decision maker in a form which can be easily comprehended. When there are only two objectives, non-dominated solutions (the data) can be represented in objective space by plotting the first objective component against the second, as shown in Figure 3.6.

Interpolating between the data to obtain a smooth representation of the Pareto-set is not generally correct, first because there is generally no guarantee that it will actually be smooth, and second because actual solutions corresponding to those intermediate objective vectors, even if they exist, are not known. At most, one may wish to draw a boundary separating those points in objective space which are dominated by or equal to at least one of the data points, from those which no data point dominates or equals. Such a boundary (see Figure 3.7) can also be seen as the locus of the family of tightest *goal vectors* which, given

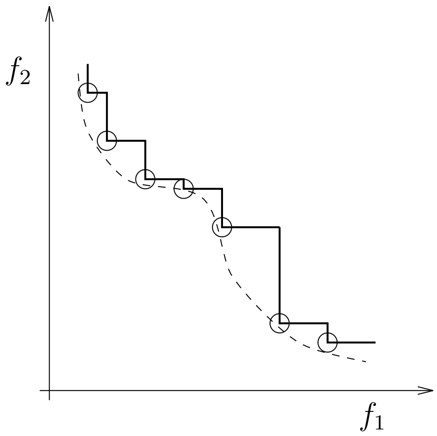


Figure 3.7: The family of tightest goals known to be attainable as a result.

the data, are known to be attainable.

For three and more objectives, a different representation is required. A common approach, known as the method of parallel coordinates, consists of associating an integer index i to each objective and representing each non-dominated point by the line connecting the points $(i, f_i^*(\mathsf{x}))$, where f_i^* represents a normalization of f_i to a given interval, e.g., $[0, 1]$. With such a representation, competing objectives with consecutive indices result in the crossing of lines, whereas non-concurrent lines indicate non-competing objectives (Figure 3.8). Although the ordering of the objectives may be automatically decided upon on the basis of some measure of competition (in order to maximize the competition between adjacent objectives, for example), being able to change this ordering interactively is also useful, and not difficult to implement.

3.4.2 Visualizing results from multiple runs

When assessing an optimizer, one is usually concerned with the quality of the solutions it is able to produce, and with the amount of computation effort it requires. In addition, it is important to assess how likely a single run is to produce good results. For instance, an optimizer may be able to find an optimum very

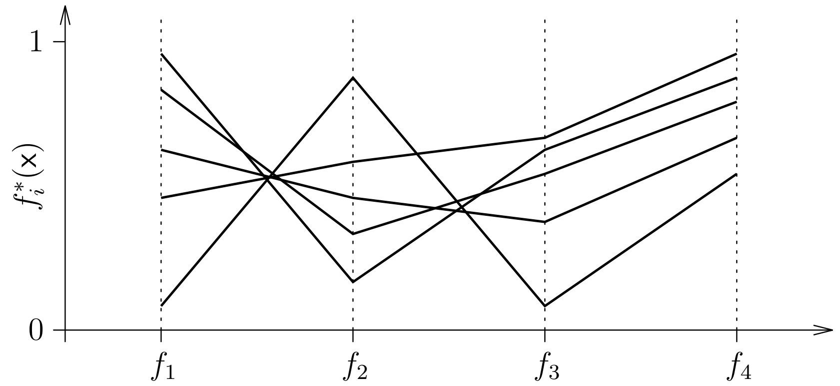


Figure 3.8: Visualization of trade-off data in three and more dimensions.

quickly and accurately, but be highly susceptible to be trapped in local optima. If applied to a highly multimodal objective function, such an optimizer may produce good (and then very good) results relatively seldom (assuming arbitrary initial conditions). On the other hand, a method slower to approach an optimum, but more likely to find the global one, may produce acceptable solutions more consistently.

In the single objective case, the quality of the result of an optimization run is measured directly by the objective value associated with the best solution found. Since this is a single, scalar value, the distribution of the quality of results produced by several runs can be easily visualized through histograms, empirical distribution functions, scatter plots, etc.

When there are multiple objectives, however, runs will generally produce not one, but a variable number of approximate non-dominated solutions. The quality of the trade-off description produced by each run depends both on how close to the real trade-off surface the non-dominated points found are, and on how well they cover it. Simply superimposing non-dominated points obtained from various runs does give an idea of how good individual points found in each run tend to be, but information on how they tend to be distributed along the trade-off surface is lost (see Figure 3.9).

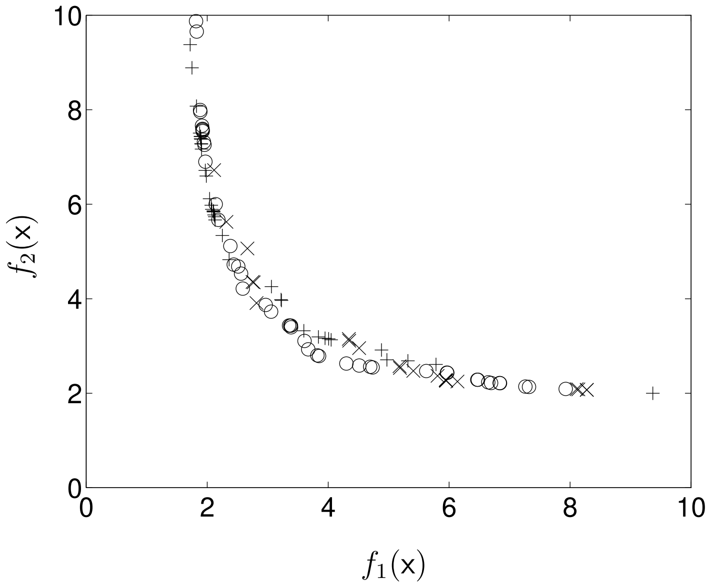


Figure 3.9: The superposition of 3 sets of non-dominated points.

A better alternative consists of seeing the outcome of a run, not as a set of approximate non-dominated solutions, but as the boundary which such solutions define in objective space, as discussed above. Information on the quality of the individual non-dominated solutions is still preserved: given the boundary, it is trivial to identify the location of the original data, and the better the approximate solutions, the closer to the real trade-off surface the boundary will be. On the other hand, any "holes" in the distribution of these solutions will have the opposite effect, i.e., will result in the corresponding region of the boundary being drawn away from the real trade-off surface, as Figure 3.7 also illustrates. Thus, both types of information are expressed in terms of the *location* of the points which constitute the boundary, relative to the real trade-off surface. However, note that the way in which information is combined is independent of the scale in which objectives are expressed, as it relies solely on the concept of Pareto-dominance.

In Figure 3.10, the superposition of the three boundaries corresponding to the data represented in Figure 3.9 is shown. The uneven distribution along the trade-off of some of the sets of non-dominated solutions is now more apparent.

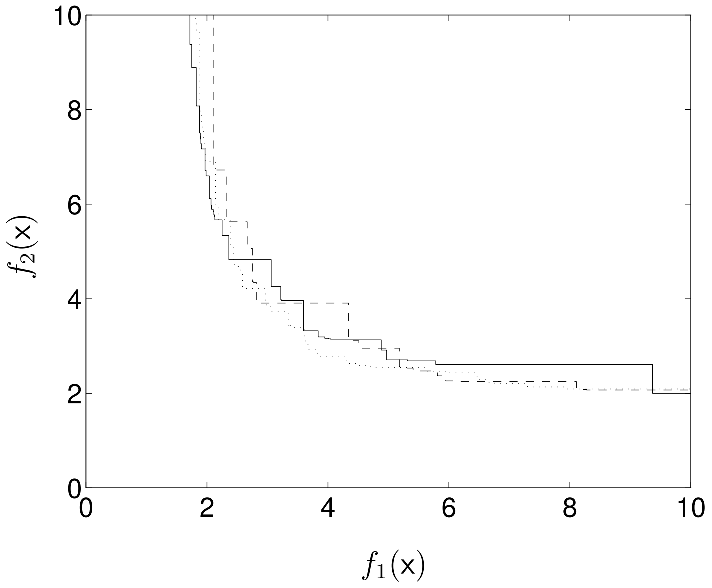


Figure 3.10: The superposition of the 3 corresponding boundaries.

3.4.3 Statistical interpretation of results from multiple runs

The plots introduced above clearly divide the objective space into three main areas. The first area, located down and to the left of the boundaries, is composed of all those objective (or goal) vectors which were never attained in any of the runs. As the number of runs increases, this area should approximate the real set of infeasible objective vectors better and better. The second area, located up and to the right of the boundaries, is composed of those objective vectors which were attained in all of the runs. This may give some idea of worst-case performance for the algorithm, but this area must be expected to depend strongly on the number of runs considered. Finally, the third area, located within the boundaries, is composed of objective vectors which were attained in some of the runs, but not in others. This area can be divided further into sub-areas, according to the percentage of runs in which the corresponding objective vectors were attained.

Suppose one wishes to estimate the family of goal vectors likely to be attained (each on its own) in exactly 50% of the runs. For that purpose, consider

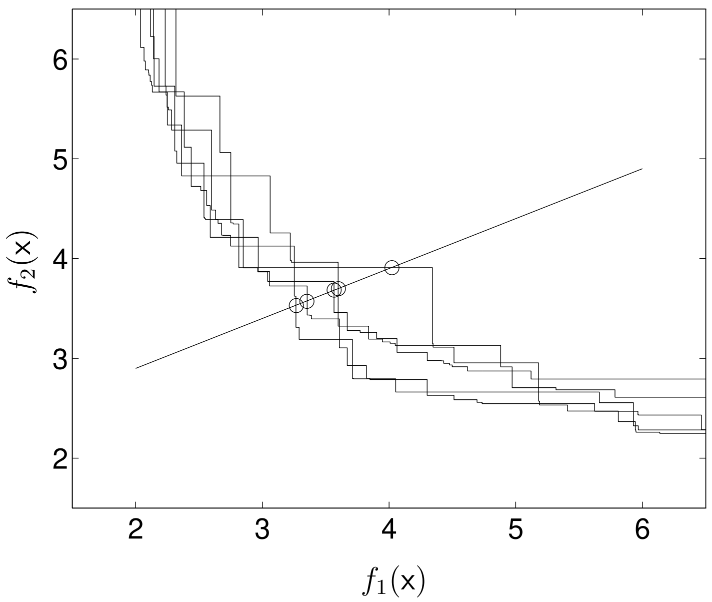


Figure 3.11: The points of intersection with a line diagonal to the axes.

the points of intersection of a set of boundaries obtained experimentally with an arbitrary straight line, diagonal to the axes and running in the direction of improvement in all objectives, as illustrated in Figure 3.11. These points can be seen to define a sample distribution which is essentially uni-dimensional and, thus, can be treated as though it was univariate. Successive estimation of the median of all possible samples defined in this way produces the desired result. In the same way, estimates for the families of goal vectors likely to be attained (each on its own) in exactly 25% anf 75% of the runs can be produced by estimating quartiles instead of the median. The plot given in Figure 3.12 represents the 50% attainment line (thick) and the upper- and lower-bounds corresponding to 5 sets of non-dominated points.

Note that, although all examples given here involve two objectives only, the concept is valid for any number of objectives. For three and more objectives, however, more elaborate visualization techniques would be needed.

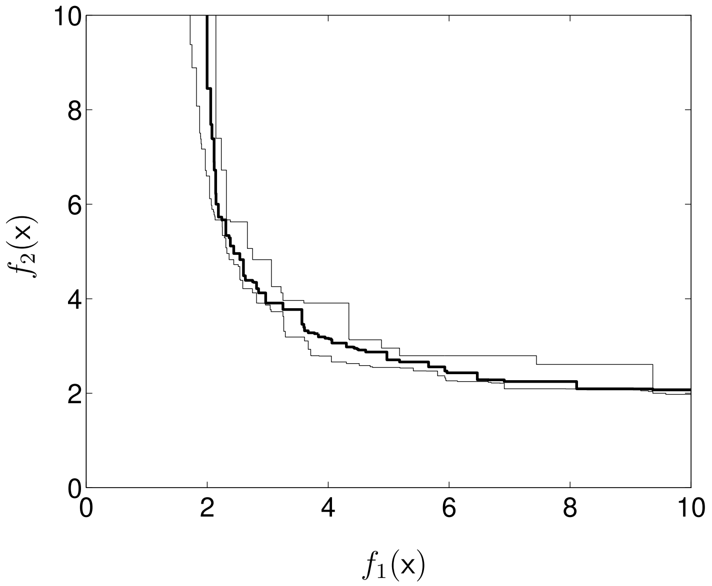


Figure 3.12: The 50% attainment line (thick) corresponding to 5 sets of non-dominated points.

3.5 Concluding remarks

Constraint satisfaction and multiobjective optimization are very much two aspects of the same problem. Both involve the simultaneous optimization of a number of functions. Constrained optimization may be seen as a particular case of multiobjective optimization with a priori articulation of preferences, where objectives are divided into "soft" and "hard", and goals are specified for the hard objectives. In the next Chapter, a unified decision making framework for multiobjective and constrained optimization is formulated and characterized, with its application to evolutionary optimization in mind.

The visualization of approximate non-dominated solutions obtained experimentally through optimization is usually aimed at communicating the (best) trade-off information gained to the Decision Maker, so that a compromise solution may be chosen from the alternatives available. Visualization techniques for this purpose are well established.

Visually assessing the statistical performance of a multiobjective optimizer, on the other hand, calls for data to be obtained from several runs of the algorithm. The simple superposition of different trade-off graphs has been shown to be confusing and even potentially misleading. For this reason, the outcome of a multiobjective optimization run was re-interpreted here so as to allow a probability of attainment to be associated with each point in objective space, and, more importantly, to be *estimated* from experimental data. The visualization of some of the experimental results obtained in Chapters 5 and 6 will be based on this interpretation.

Chapter 4

Multiobjective Genetic Algorithms

4.1 Introduction

As noted in Chapter 2, genetic algorithms were recognized to be possibly well-suited to multiobjective optimization early in their development. Multiple individuals should be able to search for multiple solutions in parallel, while taking advantage of any similarities available in the family of possible solutions to the problem. The ability to handle complex problems, involving features such as discontinuities, multimodality, disjoint feasible spaces and noisy function evaluations reinforces the potential effectiveness of GAs, and of other evolutionary algorithms, in multiobjective search and optimization. However, contributions in the area have remained scarce.

The related field of constrained optimization has often been treated separately. It has received more attention in EA literature than multiobjective optimization, but many of the approaches which have been developed require constraints to satisfy certain pre-requisites, such as linearity or convexity, or are too problem dependent.

This Chapter makes three important contributions to multiobjective evolutionary optimization:

- Section 4.2 surveys and discusses current evolutionary approaches to constrained and multiobjective optimization.
- Section 4.3 introduces and characterizes a unified decision making framework for constrained multiobjective optimization. Proofs for some of the Lemmas developed in this section can be found in Appendix A.
- Section 4.4 develops a multiobjective genetic algorithm based on that decision making framework.

Simple examples illustrating different aspects of the proposed genetic approach are presented in Section 4.5.

4.2 Current evolutionary approaches to constrained and multiobjective optimization

Fitness, as a measure of the expected reproductive success (or number of offspring) of an individual, is inherently a non-negative scalar. When a problem is characterized by a scalar measure of quality or cost, i.e., a single objective, the mapping from such a measure to fitness simply involves a monotonic transformation, such as scaling or ranking.

On the other hand, when the problem is characterized by a vectorial measure, a good mapping between quality and fitness is less straightforward to derive. This is the case when there are constraints and/or multiple objectives. Remarkably, constrained optimization has been considered separately from multiobjective optimization in EA literature, and, for that reason, the two are separately reviewed here.

4.2.1 Constraint handling

The simplest approach to constraint handling in EAs has been to assign infeasible individuals an arbitrarily low fitness (Goldberg, 1989, p. 85). This is possible given the ability of EAs to cope with the abrupt fitness changes which arise on the constraint boundaries. In this approach, provided feasible solutions can be easily found, any infeasible individuals are selected out and the search is not affected much.

Certain types of constraints, however, such as bounds on the decision variables and other linear constraints, can be handled more efficiently by mapping the search space so as to minimize the number of infeasible solutions it contains and/or designing the mutation and recombination operators carefully in order to minimize the production of infeasible offspring from feasible parents (Michalewicz and Janikow, 1991). This and the previous approach are complementary, and are often used in combination with each other.

In the case where no feasible individuals are known, and cannot easily be found, simply assigning low-fitness to infeasible individuals makes the initial stages of evolution degenerate into a random walk. To avoid this, the penalty imposed on infeasible individuals can be made to depend on the extent to which they violate the constraints. Such penalty values are typically added to the (unconstrained) performance values before fitness is computed (Goldberg, 1989, p. 85f).

Although penalty functions do provide a way of guiding the search towards feasible solutions when these are not known, they are very much problem dependent. Some infeasible solutions can, despite the penalty, be seen as better than some feasible ones, which can make the population evolve towards a false optimum. In response to these difficulties, guidelines on the use of penalty functions have been described by Richardson *et al.* (1989).

One of the most recent approaches to constraint handling has been proposed by Powell and Skolnick (1993) and consists of rescaling the original objective function to assume values less than unity in the feasible region, whilst assigning infeasible individuals penalty values greater than one. Subsequently ranking the population correctly assigns higher fitness to all feasible points than to those infeasible. This perspective is supported and extended in the present work.

4.2.2 Multiple objectives

Despite their foreseeable ability to search for multiple non-inferior solutions in a single run, EAs have mainly been used to optimize aggregating functions of the different objectives. In fact, many aggregating approaches can be used directly with EAs, despite having been developed with other optimizers in mind.

Several applications of evolutionary algorithms in the optimization of aggregating functions have been reported in the literature, from the popular weighted-sum approach (Syswerda and Palmucci, 1991; Jakob *et al.*, 1992; Jones *et al.*, 1993) to goal-based approaches. Using target vector optimization, which consists of minimizing the distance in objective space to a given goal vector, Winke *et al.* (1992) reported work on a problem in atomic emission spectroscopy. A formulation of goal attainment was used amongst other methods by Wilson and Macleod (1993). In this latter work, the aggregating function was aimed at directing the population towards a particular region of the trade-off surface, while the population was monitored for non-dominated solutions. However, any extra non-dominated individuals found in this way cannot be expected to be optimal in any sense, because they are suboptimal in terms of fitness.

4.2.2.1 Population-based non-Pareto approaches

Schaffer (1985) was probably the first to recognize the possibility of exploiting EA populations to treat non-commensurable objectives separately and search for multiple non-dominated solutions concurrently in a single optimization run.

In his approach, known as the Vector Evaluated Genetic Algorithm (VEGA), appropriate fractions of the next generation, or sub-populations, were selected from the whole of the old generation according to each of the objectives, separately. Crossover and mutation were applied as usual after shuffling all the sub-populations together. Non-dominated individuals were identified by monitoring the population as it evolved, but this information was not used by the VEGA itself.

Shuffling and merging all sub-populations corresponds, however, to averaging the *normalized* fitness components associated with each of the objectives. In fact, the expected total number of offspring produced by each parent becomes the sum of the expected numbers of offspring produced by that parent according to each objective. Since Schaffer used proportional fitness assignment, these were in turn, proportional to the objectives themselves. The resulting overall fitness corresponded, therefore, to a linear function of the objectives where the weights depended on the distribution of the population at each generation. This has previously been noted by Richardson *et al.* (1989) and confirmed by Schaffer (1993). As a consequence, different non-dominated individuals were generally assigned different fitness values, in contrast with what the definition of non-dominance would suggest.

The linear combination of the objectives implicitly performed by VEGA explains why the population tended to split into species particularly strong in each of the objectives in the case of concave trade-off surfaces, a phenomenon which Schaffer called *speciation*. As noted earlier in Section 3.2, points in concave regions of a trade-off surface cannot be found by optimizing a linear combination of the objectives, for *any* set of weights.

Although VEGA, like the plain weighted-sum approach, is not well suited to address problems with non-convex trade-off surfaces, the weighting scheme it implicitly implements deserves closer attention. In VEGA, each objective is effectively weighted proportionally to the size of each sub-population and, more importantly, proportionally to the inverse of the average fitness (in terms of that objective) of the whole population at each generation.

By doing so, and assuming that sub-population sizes remain constant for each objective, VEGA selection adaptively attempts to balance improvement in the several objective dimensions, because more good-performers in one objective cause the corresponding average performance to increase and that objective's weight to decrease accordingly. This is not unlike the way sharing techniques (subsection 2.3.6) promote the balanced exploitation of multiple optima in the search space. For the same reason, VEGA can, at least in some cases, maintain different species for many more generations than a GA optimizing a pure weighted sum of the same objectives with fixed weights would, due to genetic drift. Unfortunately, the balance reached necessarily depends on the scaling of the objectives.

Fourman (1985) also addressed multiple objectives in a non-aggregating manner. Selection was performed by comparing pairs of individuals, each pair according to one of the objectives. In a first version of the algorithm, objectives were assigned different priorities by the user and individuals compared according to the objective with the highest priority. If this resulted in a tie, the objective with the second highest priority was used, and so on. This is known as the *lexicographic* ordering (Ben-Tal, 1980).

A second version, reported to work surprisingly well, consisted of randomly selecting the objective to be used in each comparison. Similarly to VEGA, this corresponds to averaging fitness across fitness components, each component being weighted by the probability of each objective being chosen to decide each tournament. However, the use of pairwise comparisons makes it essentially *different* from a linear combination of the objectives, because scale information is ignored. Thus, the population may still see as convex a trade-off surface actually concave, depending on its current distribution and, of course, on the problem (Fonseca and Fleming, 1995d).

Kursawe (1991) formulated a multiobjective version of evolution strategies (ESs). Once again, selection consisted of as many steps as there were objectives. At each step, one objective was selected randomly (with replacement) according to a probability vector, and used to dictate the deletion of an appropriate fraction of the current population. After selection, a certain number μ of survivors became the parents of the next generation.

While Kursawe's implementation of multiobjective selection possesses a number of similarities to both VEGA and Fourman's second method, individuals in the extremes of the trade-off surface would appear to be likely to be eliminated as soon as any objective at which they performed poorly was selected to dictate deletion, whereas middling individuals seem to be more likely to survive. However, since objectives stood a certain chance of not taking part in selection at each generation, it was possible for some specialists to survive the deletion process and generate offspring, although they might die immediately the generation after.

Kursawe (1991) notes that this deletion of individuals according to randomly chosen objectives creates a non-stationary environment in which the population, instead of converging, must try to adapt to constant change. As hinted above, different choices of objectives could result in significant changes in the cost landscape seen by the ES at each generation. Diploid individuals (Goldberg and Smith, 1987) were used for their improved ability to adapt to sudden environmental changes and, since the population was not expected to converge, a picture of the trade-off surface was produced from the points evaluated during the run.

Finally, and still based on the weighted sum approach, Hajela and Lin (1992) exploited the explicit parallelism provided by a population-based search by explicitly including the weights in the chromosome and promoting their diversity in the population through fitness sharing. As a consequence, one family of individuals evolved for each weight combination, concurrently.

4.2.2.2 Pareto-based approaches

The methods of Schaffer, Fourman, Kursawe, and Hajela and Lin, all attempt to promote the generation of multiple non-dominated solutions, a goal at which they reportedly achieved a reasonable degree of success. However, none makes *direct* use of the actual definition of Pareto-optimality. At most, the population is monitored for non-dominated solutions, as in Schaffer (1985) and Kursawe (1991).

Pareto-based fitness assignment was first proposed by Goldberg (1989), the idea being to assign equal probability of reproduction to all non-dominated individuals in the population. The method consisted of assigning rank 1 to the non-dominated individuals and removing them from contention, then finding a new set of non-dominated individuals, ranked 2, and so forth.

Fonseca and Fleming (1993) proposed a slightly different scheme whereby an individual's rank corresponds to the number of individuals in the current population by which it is dominated. Non-dominated individuals are, therefore, all assigned the same rank, while dominated ones are penalized according to how concentrated the population is in the corresponding region of the trade-off surface. This approach is also easier to formulate, and to analyze, mathematically. It will be presented and discussed further in subsection 4.3.2.

Tournament selection based on Pareto dominance has meanwhile been proposed by Horn *et al.* (1994). In addition to the two individuals competing in each tournament, a number of other individuals in the population was used to help determine whether the competitors were dominated or not. In the case where both competitors were either dominated or non-dominated, the result of the tournament was decided through a sort of fitness sharing.

Cieniawski (1993) and Ritzel *et al.* (1994) have implemented tournament selection based on Goldberg's Pareto-ranking scheme. In their approach, individual ranks were used to decide the winner of binary tournaments.

As discussed in Chapter 3, the convexity of the trade-off surface depends on how the objectives are scaled. Rescaling the objective values by means of a monotonic non-linear transformation may convert a concave surface into a convex one, and vice-versa. Nevertheless, the solution set of the problem remains the same in phenotypic space.

Since order is preserved by monotonic transformations, Pareto-ranking is effectively blind to the convexity or the non-convexity of the trade-off surface. This is not to say that Pareto-ranking always precludes speciation. Speciation can still occur if certain regions of the trade-off are simply easier to find than others, but Pareto-ranking does eliminate sensitivity to the possible non-convexity of the trade-off.

A second possible advantage of Pareto-ranking, is that, because it rewards good performance in any objective dimension *regardless* of the others, solutions which exhibit good performance in many, if not all, objective dimensions are more likely to be produced by recombination. This argument also applies to an extent to the population-based methods described in the previous subsection, although they do not necessarily treat all non-dominated individuals equally. The argument assumes some degree of independence between objectives, and was already hinted at by Schaffer in his VEGA work, and has been noted in more detail by Louis and Rawlins (1993). While Pareto-based selection may help find Utopian solutions if they exist, that is rarely the case in multiobjective optimization. Also, the assumption of loosely coupled objectives is less likely to hold near the admissible region, but the argument may still be valid in the initial stages of the search.

4.2.3 Generalization

The multiobjective evolutionary optimization process can be generalized (Fonseca and Fleming, 1993) and seen as the result of the interaction between an artificial selector, here referred to as the Decision Maker (DM), and an evolutionary search

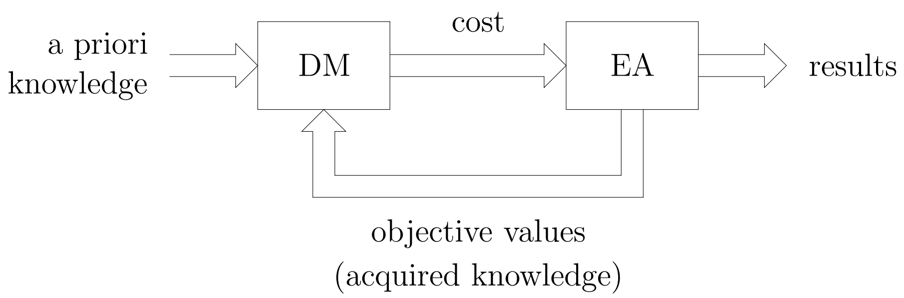


Figure 4.1: A general multiobjective evolutionary optimizer.

algorithm (the EA, see Figure 4.1). The search algorithm generates a new set of candidate solutions according to the cost (alternatively, utility) assigned to the current set of candidates by the DM.

Whilst the action of the DM influences the production of new individuals, these, as they are evaluated, provide new trade-off information which the DM can use to refine the current preferences. The EA sees the effect of any changes in the decision process, which may or may not result from taking recently acquired information into account, as an environmental change. The DM block represents any cost assignment strategy, which may range from that of an intelligent Decision Maker to a simple weighted sum approach.

The EA block is concerned with a different, but complementary, aspect of the optimization, the effective search for strong candidate solutions. Evolutionary algorithms, in the first instance, make very few assumptions about the fitness landscape they work on, which justifies and permits a primary concern with fitness assignment. However, EAs are certainly not capable of optimizing arbitrary functions (Hart and Belew, 1991). Some form of characterization of the multiobjective fitness landscapes associated with the decision making strategy used is, therefore, important, and the design of the EA should take that information into account. The need for improved EAs in multiobjective optimization will become clearer later in this Chapter.

The next Section elaborates on Pareto-based ranking by combining dominance with preference information to produce a suitable unified cost assignment strategy, capable of handling both constraints and multiple objectives.

4.3 Multiobjective decision making based on given goals and priorities

The specification of goals and priorities (subsection 3.2.2) can accommodate a whole variety of constrained and/or multiobjective problem formulations. Goal and priority information can often be extracted directly from the problem description, even if only in a broad sense. For example, it may be difficult to strictly prioritize all of the objectives, or to set goals for all of them. Therefore, the articulation of this information should take its partial character into account. It should be possible for different objectives to be given the same priority, and for goals to be omitted (or set to $\pm\infty$) where necessary. On the other hand, measures of distance to the goals should probably be avoided, as they inevitably depend on the goals being finite and on the scale in which the objective values are presented.

An extension of the decision making strategy proposed in (Fonseca and Fleming, 1993) is formulated here in terms of a relational operator, which incorporates any goal and priority information given, and characterized. The ranking of a whole population based on such a relation is then described.

4.3.1 The comparison operator

Consider an n-dimensional vector function $\mathbf{f}$ of some decision variable x and two n-dimensional objective vectors $\mathbf{u} = \mathbf{f}(\mathsf{x}_\mathrm{u})$ and $\mathbf{v} = \mathbf{f}(\mathsf{x}_\mathrm{v})$, where x_u and x_v are particular values of x. Consider also the n-dimensional *preference vector*

$$\begin{aligned}\mathbf{g} &= [\mathbf{g}_1, \ldots, \mathbf{g}_p] \\ &= [(g_{1,1}, \ldots, g_{1,n_1}), \ldots, (g_{p,1}, \ldots, g_{p,n_p})] \,,\end{aligned}$$

where p is a positive integer (see below), $n_i \in \{0, \ldots, n\}$ for $i = 1, \ldots, p$, and

$$\sum_{i=1}^{p} n_i = n \,.$$

Similarly, $\mathbf{u}$ may be written as

$$\begin{aligned}\mathbf{u} &= [\mathbf{u}_1, \ldots, \mathbf{u}_p] \\ &= [(u_{1,1}, \ldots, u_{1,n_1}), \ldots, (u_{p,1}, \ldots, u_{p,n_p})] \,,\end{aligned}$$

and the same for $\mathbf{v}$ and $\mathbf{f}$.

The sub-vectors $\mathbf{g}_i$ of the preference vector $\mathbf{g}$, where $i = 1, \ldots, p$, associate priorities i and goals g_{i,j_i}, where $j_i = 1, \ldots, n_i$, to the corresponding objective functions f_{i,j_i}, components of $\mathbf{f}_i$. This assumes a convenient permutation of the components of $\mathbf{f}$, without loss of generality. Greater values of i, up to and including p, indicate higher priorities.

Generally, each sub-vector $\mathbf{u}_i$ will be such that a number $k_i \in \{0, \ldots, n_i\}$ of its components meet their goals while the remaining do not. Also without loss of generality, $\mathbf{u}$ is such that, for $i = 1, \ldots, p$, one can write

$$\exists\, k_i \in \{0, \ldots, n_i\} \mid \;\forall\, \ell \in \{1, \ldots, k_i\} \;, \quad \forall\, m \in \{k_i + 1, \ldots, n_i\} \;, \\ (u_{i,\ell} \le g_{i,\ell}) \wedge (u_{i,m} > g_{i,m}) \,. \quad (4.1)$$

For simplicity, the first k_i components of vectors $\mathbf{u}_i$, $\mathbf{v}_i$ and $\mathbf{g}_i$ will be represented as $\mathbf{u}_i^{\overset{\mathbf{u}}{\smile}}$, $\mathbf{v}_i^{\overset{\mathbf{u}}{\smile}}$ and $\mathbf{g}_i^{\overset{\mathbf{u}}{\smile}}$, respectively. The last $n_i - k_i$ components of the same vectors will be denoted $\mathbf{u}_i^{\overset{\mathbf{u}}{\frown}}$, $\mathbf{v}_i^{\overset{\mathbf{u}}{\frown}}$ and $\mathbf{g}_i^{\overset{\mathbf{u}}{\frown}}$, also respectively. The smile ($\overset{\mathbf{u}}{\smile}$) and the frown ($\overset{\mathbf{u}}{\frown}$), respectively, indicate the components in which $\mathbf{u}$ either does or does not meet the goals.

Definition 4.1 (Preferability) *Vector* $\mathbf{u} = [\mathbf{u}_1, \ldots, \mathbf{u}_p]$ *is preferable to* $\mathbf{v} = [\mathbf{v}_1, \ldots, \mathbf{v}_p]$ *given a preference vector* $\mathbf{g} = [\mathbf{g}_1, \ldots, \mathbf{g}_p]$ *(*$\mathbf{u} \underset{\mathbf{g}}{\prec} \mathbf{v}$*) iff*

$$p = 1 \;\;\Rightarrow\;\; (\mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} \;\mathrm{p}\!< \mathbf{v}_p^{\overset{\mathbf{u}}{\frown}}) \vee \left\{ (\mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} = \mathbf{v}_p^{\overset{\mathbf{u}}{\frown}}) \wedge \left[(\mathbf{v}_p^{\overset{\mathbf{u}}{\smile}} \not\leq \mathbf{g}_p^{\overset{\mathbf{u}}{\smile}}) \vee (\mathbf{u}_p^{\overset{\mathbf{u}}{\smile}} \;\mathrm{p}\!< \mathbf{v}_p^{\overset{\mathbf{u}}{\smile}}) \right] \right\}$$

and

$$p > 1 \;\;\Rightarrow\;\; (\mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} \;\mathrm{p}\!< \mathbf{v}_p^{\overset{\mathbf{u}}{\frown}}) \vee \left\{ (\mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} = \mathbf{v}_p^{\overset{\mathbf{u}}{\frown}}) \wedge \left[(\mathbf{v}_p^{\overset{\mathbf{u}}{\smile}} \not\leq \mathbf{g}_p^{\overset{\mathbf{u}}{\smile}}) \vee (\mathbf{u}_{1,\ldots,p-1} \underset{\mathbf{g}_{1,\ldots,p-1}}{\prec} \mathbf{v}_{1,\ldots,p-1}) \right] \right\},$$

where $\mathbf{u}_{1,\ldots,p-1} = [\mathbf{u}_1, \ldots, \mathbf{u}_{p-1}]$ *and similarly for* $\mathbf{v}$ *and* $\mathbf{g}$.

In simple terms, vectors $\mathbf{u}$ and $\mathbf{v}$ are compared first in terms of their components with the highest priority, that is, those where $i = p$, disregarding those in which $\mathbf{u}_p$ meets the corresponding goals, $\mathbf{u}_p^{\overset{\mathbf{u}}{\smile}}$. In case both vectors meet all goals with this priority, or if they violate some or all of them, but in exactly the same way, the next priority level $(p-1)$ is considered. The process continues until priority 1 is reached and satisfied, in which case the result is decided by comparing the priority 1 components of the two vectors in a Pareto fashion.

Since satisfied high-priority objectives are left out from comparison, vectors which are equal to each other in all but these components express virtually no trade-off information given the corresponding preferences. The following symmetric relation is defined:

Definition 4.2 (Equivalence) *Vector* $\mathbf{u} = [\mathbf{u}_1, \ldots, \mathbf{u}_p]$ *is equivalent to* $\mathbf{v} = [\mathbf{v}_1, \ldots, \mathbf{v}_p]$ *given a preference vector* $\mathbf{g} = [\mathbf{g}_1, \ldots, \mathbf{g}_p]$ *(*$\mathbf{u} \underset{\mathbf{g}}{\equiv} \mathbf{v}$*) iff*

$$(\mathbf{u}^{\overset{\mathbf{u}}{\frown}} = \mathbf{v}^{\overset{\mathbf{u}}{\frown}}) \wedge (\mathbf{u}_1^{\overset{\mathbf{u}}{\smile}} = \mathbf{v}_1^{\overset{\mathbf{u}}{\smile}}) \wedge (\mathbf{v}_{2,\ldots,p}^{\overset{\mathbf{u}}{\smile}} \leq \mathbf{g}_{2,\ldots,p}^{\overset{\mathbf{u}}{\smile}}).$$

The concept of preferability can be related to that of inferiority as follows:

Lemma 4.1 *For any two objective vectors* $\mathbf{u}$ *and* $\mathbf{v}$, *if* $\mathbf{u} \;{}_{\mathrm{p}}\!< \mathbf{v}$, *then* $\mathbf{u}$ *is either* preferable *or* equivalent *to* $\mathbf{v}$, *given any preference vector* $\mathbf{g} = [\mathbf{g}_1, \ldots, \mathbf{g}_p]$.

The proofs of this lemma and of the following one are given in Appendix A.

Lemma 4.2 (Transitivity) *The preferability relation is transitive, i.e., given any three objective vectors* $\mathbf{u}$, $\mathbf{v}$ *and* $\mathbf{w}$, *and a preference vector* $\mathbf{g} = [\mathbf{g}_1, \ldots, \mathbf{g}_p]$,

$$\mathbf{u} \underset{\mathbf{g}}{\prec} \mathbf{v} \underset{\mathbf{g}}{\prec} \mathbf{w} \quad \underset{\mathbf{g}}{\Longrightarrow} \quad \mathbf{u} \underset{\mathbf{g}}{\prec} \mathbf{w}\,.$$

4.3.1.1 Particular cases

The decision strategy described above encompasses a number of simpler multi-objective decision strategies, which correspond to particular settings of the preference vector.

Pareto (Definition 3.1) All objectives have equal priority and no goal levels are given. $\mathbf{g} = [\mathbf{g}_1] = [(-\infty, \ldots, -\infty)]$.

Lexicographic (Subsection 3.2.2.2) Objectives are all assigned different priorities and no goal levels are given. $\mathbf{g} = [\mathbf{g}_1, \ldots, \mathbf{g}_n] = [(-\infty), \ldots, (-\infty)]$.

Constrained optimization (Section 3.3) The functional parts of a number n_c of inequality constraints are handled as high priority objectives to be minimized until the corresponding constant parts, the goals, are reached. Objective functions are assigned the lowest priority.
$\mathbf{g} = [\mathbf{g}_1, \mathbf{g}_2] = [(-\infty, \ldots, -\infty), (g_{2,1}, \ldots, g_{2,n_c})]$.

Constraint satisfaction (or Method of Inequalities (Zakian and Al-Naib, 1970)) All constraints are treated as in constrained optimization (above), but there is no low priority objective to be optimized. $\mathbf{g} = [\mathbf{g}_2] = [(g_{2,1}, \ldots, g_{2,n})]$.

Goal programming Several interpretations of goal programming can be implemented. A simple formulation, described in (Hwang and Masud, 1979), consists of attempting to meet the goals sequentially, in a similar way to lexicographic optimization. $\mathbf{g} = [\mathbf{g}_1, \ldots, \mathbf{g}_n] = [(g_{1,1}), \ldots, (g_{n,1})]$.

A second formulation attempts to meet all the goals simultaneously, as with constraint satisfaction, but requires solutions to be satisfactory and Pareto optimal. $\mathbf{g} = [\mathbf{g}_1] = [(g_{1,1}, \ldots, g_{1,n})]$.

Aggregating functions, such as weighted sums and the maximum of a number of objectives, can, of course, be used as individual objectives. Although this may be appropriate in the case where they express some global criterion, e.g., financial cost, they inevitably hide information from the Decision Maker, which may constitute a disadvantage. From a practical standpoint, it is especially worth pointing out that, as the number of objectives increases, it becomes more likely that some objectives are, in fact, non-competing, at least in portions of the trade-off surface. The understanding that some objectives are non-competing constitutes a valuable insight into the problem, because the number of dimensions involved in the trade-off surface is reduced.

4.3.2 Population ranking

As opposed to the single objective case, the ranking of a population in the multiobjective case is not unique. In the present case, it is desired that all preferred individuals be assigned the same rank, and that individuals be placed higher in the rank than those they are preferable to.

Consider an individual x_u at generation t with corresponding objective vector $\mathbf{u}$, and let $r_\mathrm{u}^{(t)}$ be the number of individuals in the current population which are preferable to it. The current position of x_u in the individuals' rank can be given simply by

$$\mathrm{rank}(\mathsf{x}_\mathrm{u}, t) = r_\mathrm{u}^{(t)} ,$$

which ensures that all preferred individuals in the current population are assigned rank zero.

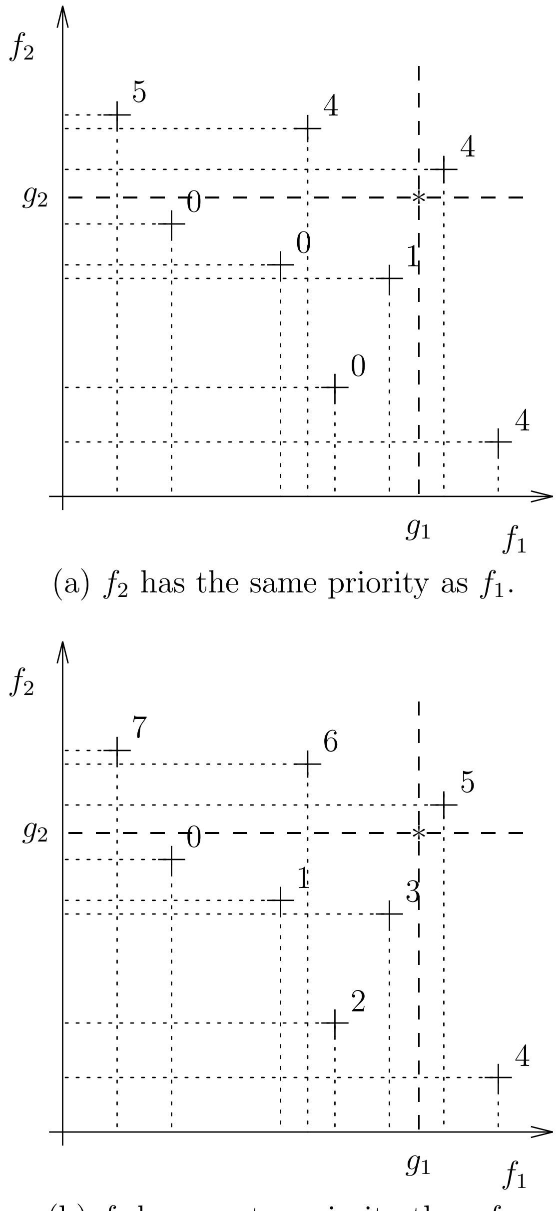
In the case of a large and uniformly distributed population with N individuals, the normalized rank $r^{(t)}/N$ constitutes an estimate of the fraction of the search space preferable to each individual considered. Such a fraction indicates how easily the current solution can be improved by pure random search and, as a measure of individual cost, does not depend on how the objectives are scaled. This interpretation of ranking, also valid when there is only one objective, provides a way of characterizing the cost landscape associated with the preferences of the DM. It is not applicable to the ranking approach proposed by Goldberg (1989, p. 201).

In the general case of a non-uniformly distributed population, a biased estimate is obtained which, nevertheless, preserves the strict order relationships between individuals, as desired:

Lemma 4.3 *If an objective vector* $\mathbf{u} = \mathbf{f}(\mathsf{x}_\mathrm{u})$ *associated with an individual* x_u *is preferable to another vector* $\mathbf{v} = \mathbf{f}(\mathsf{x}_\mathrm{v})$ *associated with an individual* x_v *in the same arbitrary population, then* $\mathrm{rank}(\mathsf{x}_\mathrm{u}, t) < \mathrm{rank}(\mathsf{x}_\mathrm{v}, t)$. *Equivalently, if* $\mathrm{rank}(\mathsf{x}_\mathrm{u}, t) \geq \mathrm{rank}(\mathsf{x}_\mathrm{v}, t)$, *then* $\mathbf{u}$ *is not preferable to* $\mathbf{v}$.

The proof follows from the transitivity of the preferability relation (Lemma 4.2).

Figure 4.2 illustrates the ranking of the same population for two different preference vectors. In the first case, both objectives are given the same priority. Note that all satisficing individuals (the ones which meet their goals) are preferable to, and therefore have lower rank than, all of the remaining ones. In the second case, objective 2 is given a higher priority, reflecting, for example, a feasibility constraint. In this case, individuals which do not meet goal g_2 are the worst (they are infeasible), independently of their "theoretical" performance according to f_1. Once g_2 is met, f_1 is used for ranking. Individuals which meet both goals are satisficing solutions, whereas those which meet only g_2 are feasible, but unsatis-



(a) f_2 has the same priority as f_1.

(b) f_2 has greater priority than f_1.

Figure 4.2: Multiobjective ranking with goal values (minimization).

factory. Note how particular ranks need not be represented in the population at each particular generation.

4.3.3 Characterization of multiobjective cost landscapes

The cost landscape associated with a problem involving multiple objectives depends not only on the objectives themselves, but also on the preferences expressed by the DM. Their effect can be more easily understood by means of an example. Consider the simple bi-objective problem of simultaneously minimizing

$$\begin{aligned} f_1(x_1, x_2) &= 1 - \exp\left(-(x_1 - 1)^2 - (x_2 + 1)^2\right) \\ f_2(x_1, x_2) &= 1 - \exp\left(-(x_1 + 1)^2 - (x_2 - 1)^2\right) . \end{aligned}$$

As suggested in the previous subsection, the cost landscape associated with a given set of preferences can be inferred from the ranking of a large, uniformly distributed population, and since the problem involves only two decision variables, visualized.

Pareto-ranking assigns the same cost to all non-dominated individuals, producing a long flat inverted ridge, as is shown in Figure 4.3. If achievable goals are specified, a discontinuity arises where solutions go from satisficing to unsatisfactory (Figure 4.4). A ridge, though shorter than in the previous case, is produced by those satisfactory solutions which are also non-dominated.

Giving one objective priority over the other considerably alters the landscape. In this case, the discontinuity corresponds to the transition from feasible to infeasible, and it happens to occur in the neighbourhood of the optimum (Figure 4.5). Finally, if both objectives are made into hard constraints, the feasible region becomes totally flat (Figure 4.6). This is because, in the absence of any other objectives, all solutions which satisfy both constraints must be considered equivalent.

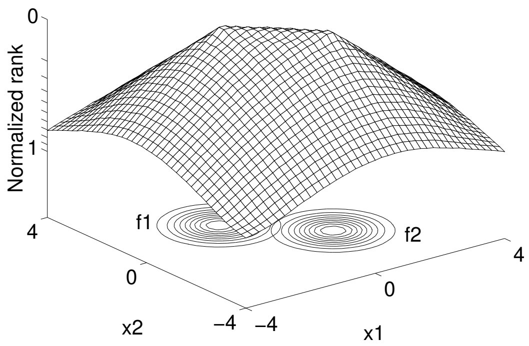


Figure 4.3: The cost landscape defined by Pareto-ranking (the contour plots are those of the individual objective functions f_1 and f_2).

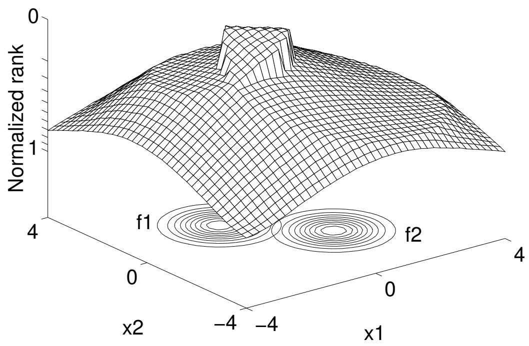


Figure 4.4: The effect of specifying two goals with the same priority.

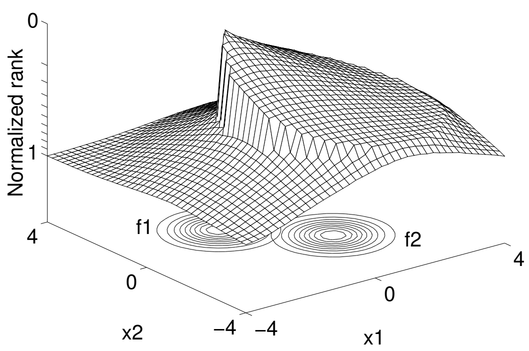


Figure 4.5: The effect of giving f_2 priority over f_1 (same goals).

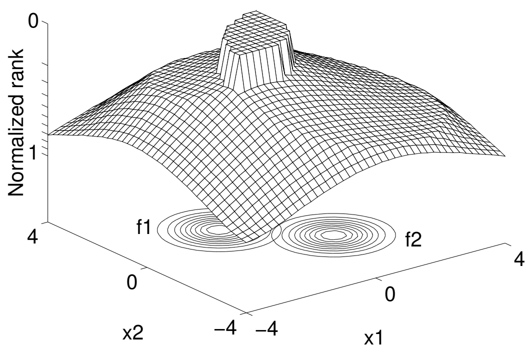


Figure 4.6: The effect of making both f_1 and f_2 into hard objectives (same goals).

Despite the underlying objectives being continuous, smooth and unimodal, the landscapes can be seen to exhibit features such as discontinuities, non-smoothness and flat regions. EA-based optimizers have proved to be able to cope with such features in single objective optimization, and should, therefore, be applicable here as well.

4.4 Multiobjective genetic algorithms

The ranking of a population was seen in Chapter 2 to provide sufficient relative quality information to guide evolution. Given a ranked population, different EAs proceed with different fitness assignment strategies, selection and reproduction schemes to produce a new set of individuals to be assessed.

This section will be concerned with the formulation of a Multiobjective Genetic Algorithm (MOGA), based on the ranking approach described earlier.

4.4.1 Fitness assignment

Fitness is understood here as the number of offspring an individual is expected to produce through selection. It is fundamentally different from cost or utility, which reflect the result of the decision making process, and are independent from the search process. The selection process determines which individuals actually influence the production of the next generation and, consequently, must be considered part of the search strategy.

The traditional rank-based fitness assignment is only slightly modified, as follows:

1. Sort population according to rank.

2. Assign fitness by interpolating from the best individual (rank $= 0$) to the worst (rank $= \max r^{(t)} < N$) according to some function, usually linear or exponential, but possibly of other types.

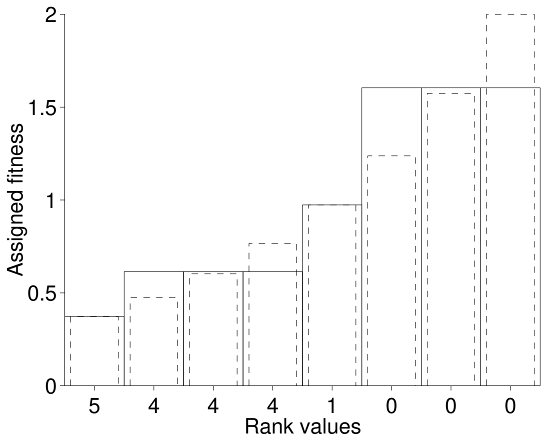


Figure 4.7: Rank-based fitness assignment.

3. Average the fitness assigned to individuals with the same rank, so that all of them are sampled at the same rate while keeping the global population fitness constant.

Exponential rank-based fitness assignment is illustrated in Figure 4.7 for the population shown earlier in Figure 4.2 (a). Individuals are sorted by their multiobjective rank and first assigned fitness values according to an exponential rule (narrower bars). Then, a single value of fitness is derived for each group of individuals with the same rank, through averaging (wider bars).

Rank-based fitness assignment, as described, transforms the cost landscape defined by the ranks into a fitness landscape which is also independent from objective scaling.

4.4.2 Niche induction methods

The flat regions defined in the fitness landscape by Pareto and preferability ranking raise two problems already encountered with the more conventional multipeak type of multimodality. Firstly, genetic drift may cause the population to converge only to a small region of the trade-off surface, unless measures such as fitness sharing are taken against it (Goldberg, 1989; Fonseca and Fleming, 1993).

Secondly, mating of very different well-performing individuals often proves not to be viable, requiring mating to be restricted to individuals similar to one another for the population to be able to maintain itself around the trade-off surface. This encourages the formation of offspring similar to their parents, resulting in a search which is less exploratory.

4.4.2.1 Fitness sharing

As explained in Chapter 2, fitness sharing (Goldberg and Richardson, 1987) models individual competition for finite resources in a closed environment. Individuals similar to one another (according to some measure of similarity) mutually decrease each other's fitness by competing for the same resources. Even if originally considered less fit, isolated individuals are thus given a greater chance of reproducing, favouring diversification.

Finding a good trade-off description means achieving a diverse, if not uniform, sampling of the trade-off surface *in objective function space*. In the sharing scheme proposed here, share counts are computed based on individual distance in the objective domain, but only between individuals with the same rank. Sharing works by providing an additional selective pressure to that imposed by ranking, in order to counter the effects of genetic drift. Genetic drift becomes more important as more individuals in the population are assigned the same rank.

4.4.2.2 Setting the niche size

The sharing parameter σ_{share} establishes how far apart two individuals must be in order for them to decrease each other's fitness. The exact value which would allow a number of points to sample a trade-off surface whilst only tangentially interfering with one another depends on the area of such a surface. The following results assume that all objectives have the same, low priority, but can also be applied to a certain extent when there are multiple priority levels.

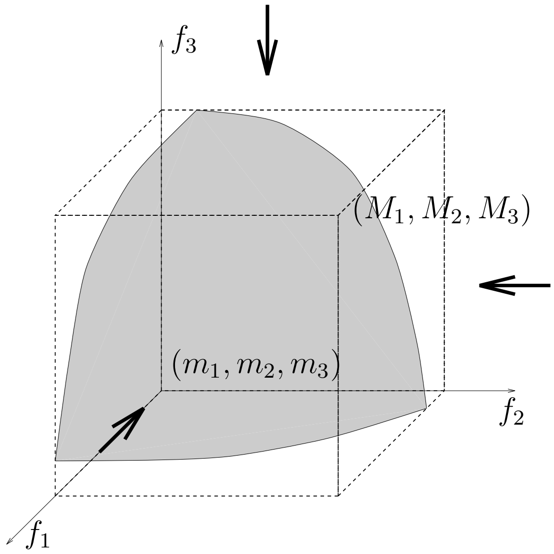


Figure 4.8: An example of a trade-off surface in 3-dimensional space.

When expressed in the objective value domain, and due to the definition of non-inferiority, an upper limit for the size of the trade-off surface can be calculated from the minimum and maximum values each objective assumes within that surface. Let S be the trade-off set in the decision variable domain, $\mathbf{f}(S)$ the trade-off set in the objective domain and $\mathbf{u} = (u_1, \ldots, u_n)$ any objective vector in $\mathbf{f}(S)$. Also, let

$$\begin{aligned}
\mathbf{m} &= (\min_{\mathbf{u}} y_1, \ldots, \min_{\mathbf{u}} y_n) = (m_1, \ldots, m_n) \\
\mathbf{M} &= (\max_{\mathbf{u}} y_1, \ldots, \max_{\mathbf{u}} y_n) = (M_1, \ldots, M_n)\,,
\end{aligned}$$

as illustrated in Figure 4.8.

The definition of non-dominance implies that any line parallel to any of the axes will have not more than one of its points in $\mathbf{f}(S)$, i.e., each objective is a single-valued function of the remaining objectives. Therefore, the true area of $\mathbf{f}(S)$, if it exists, will be less than the sum of the areas of its projections according to each of the axes. Since the maximum area of each projection will be at most the area of the corresponding face of the hyperparallelogram defined by $\mathbf{m}$ and

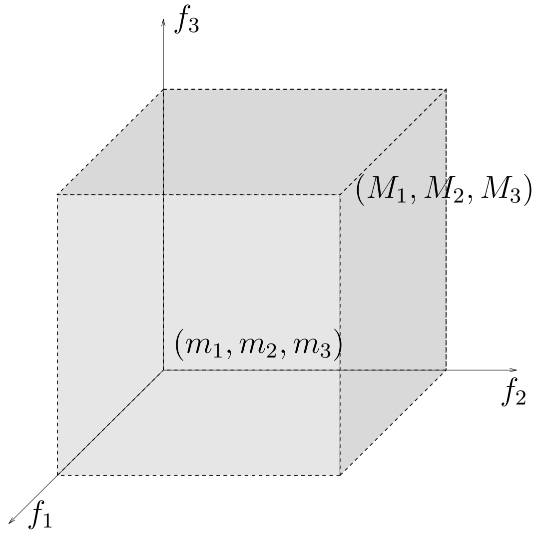


Figure 4.9: Upper bound for the area of a trade-off surface limited by the parallelogram defined by (m_1, m_2, m_3) and (M_1, M_2, M_3).

$\mathbf{M}$, the hyperarea of $\mathbf{f}(S)$ will be less than

$$A = \sum_{i=1}^{n} \prod_{\substack{j=1 \\ j \neq i}}^{n} \Delta_j \, ,$$

which is the sum of the areas of each different face of a hyperparallelogram of edges $\Delta_j = (M_j - m_j)$ (Figure 4.9).

The setting of σ_{share} also depends on how the distance between individuals is measured, and namely on how the objectives are scaled. In fact, the idea of sampling the trade-off surface *uniformly* implicitly refers to the scale in which objectives are expressed. The appropriate scaling of the objectives can often be determined as the aspect ratio which provides an *acceptable* visualization of the trade-off, or by normalizing objectives by the magnitude of the corresponding goal values. In particular, normalizing objectives by the best estimate of Δ_j available at each particular generation seems to yield good results (see the application examples in the next Chapters). This view is also expressed in a recent paper by Horn *et al.* (1994).

Assuming objectives are appropriately scaled, and using the ∞-norm as a

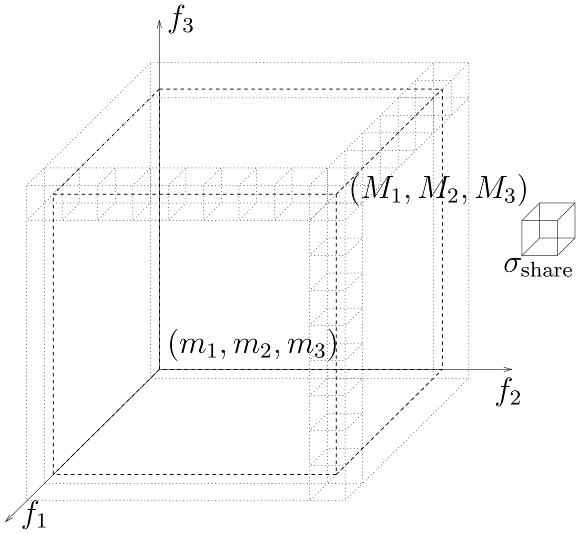


Figure 4.10: Sampling area A. Each point is σ_{share} apart from each of its neighbours (∞-norm).

measure of distance, the maximum number of points that can sample area A without interfering with each other can be computed as the number of hypercubes of volume σ_{share}^n that can be placed over the hyperparallelogram defined by A (Figure 4.10). This can be estimated from the difference in volume between two hyperparallelograms, one with edges $\Delta_i + \sigma_{\text{share}}$ and the other with edges Δ_i, by dividing it by the volume of a hypercube of edge σ_{share}, i.e.,

$$N = \frac{\prod_{i=1}^{n} (\Delta_i + \sigma_{\text{share}}) - \prod_{i=1}^{n} \Delta_i}{\sigma_{\text{share}}^n} .$$

Conversely, given a number of individuals (points), N, it is possible to estimate σ_{share} by solving the $(n-1)$-order polynomial equation

$$N\sigma_{\text{share}}^{n-1} - \frac{\prod_{i=1}^{n} (\Delta_i + \sigma_{\text{share}}) - \prod_{i=1}^{n} \Delta_i}{\sigma_{\text{share}}} = 0$$

for $\sigma_{\text{share}} > 0$.

When there are objectives with different priorities and there are known solutions which meet all goals with priority higher than 1, trade-offs will involve only priority 1 (low-priority) objectives. The sharing parameter can, therefore, be computed for these only, using the expression above. This should be the case towards the end of the GA run in a problem where high-priority objectives can be satisfied.

Similarly, if the highest level of priority, i, which the preferred solutions known at any given time violate is greater than 1, the trade-offs explored by the preferability relation will not involve objectives with priority higher than i. Again, sharing may be performed while taking into account priority i objectives only. It is a fact that objectives with priority lower than i *may* also become involved in the decision process, but this will only happen when comparing vectors with equal violating priority i components. If this is the case, and the DM decides to move on to consider objectives with priority $i-1$, then the relevant priority-i objectives should either see their associated goals changed, or be associated priority $i-1$ by the DM for sharing to occur as desired.

4.4.2.3 Mating restriction

Mating restriction can be implemented much in the same way as sharing, by specifying how close individuals should be in order to mate. The corresponding parameter, σ_{mate}, can also be defined in the objective domain. After selection, one individual in the population is chosen, and the population searched for a mate within a distance σ_{mate}. If such an individual can be found, then mating is performed. Otherwise, a random individual is chosen (Deb and Goldberg, 1989).

Mating restriction assumes that neighbouring fit individuals are genotypically similar, so that they can form stable niches. Extra attention must therefore be paid to the coding of the chromosomes. Gray codes, as opposed to standard binary, are known to be useful for their property of adjacency. However, encoding and manipulating decision variables independently of each other, as is usually the case, cannot be expected to consistently express any relationship between them.

On the other hand, the Pareto/preferred sets, when represented in the decision variable domain, are very likely to exhibit such dependencies, as is the case for the example shown earlier in Figure 4.3. As the size of the solution set in phenotypic space and/or the number of decision variables increase, an increasing number of individuals may be necessary in order to assure niche sizes small enough for the individuals in each niche to be sufficiently similar to each other. Only then can the formation of lethals be effectively reduced.

The difficulties that ridge-shaped fitness landscapes present to simple genetic search have also been noted by Wagner (1988). He has shown theoretically that, under biologically reasonable assumptions, the rate of progression of unconstrained phenotypes on certain types of ridge-shaped landscapes is bounded, in which case it decreases rapidly as the number of decision variables increases. This implies that genetic operators must be biased towards producing offspring which stay on the ridge. Mating restriction attempts to do this by keeping the level of exploration small, which may result in the predicted slow progression of the population across the trade-off surface. Achieving fast progression on such landscapes requires the concurrent search for a suitable genetic representation or, alternatively, some sort of self-adaptation mechanism of the type found in evolution strategies for mutation variances and correlated mutations. This is currently an open field for research.

In order to work around these difficulties, the DM can interactively reduce the size of the trade-off set by appropriately refining the current preferences. This requires the GA to be able to cope in some way with the corresponding change in the fitness landscape.

4.4.3 Progressive articulation of preferences

Setting definite aspiration levels in terms of goals and associated priorities is often difficult if done in the absence of any trade-off information. On the other hand, an accurate global description of the trade-off surface tends to be expensive, or even impossible, to produce, since the Pareto set may not be bounded. Interactively refining preferences has the potential advantage of reducing computational effort by concentrating optimization effort on the region from which compromise solutions are more likely to emerge, while simultaneously providing the DM with trade-off information on which preference refinement can be based.

From the optimizer's point of view, the main difficulty associated with progressive articulation of preferences is the changing environment on which it must work. Consequently, the action of the DM may have to be restricted to the tightening of initially loose requirements, as with the moving-boundaries process (Zakian and Al-Naib, 1970). In this case, though the overall optimization problem may change, the final solution must remain in the set of possible solutions which satisfy the current preferences at any given time.

When EA-based optimizers are used, the DM may gain more freedom and actually decide to explore regions of the trade-off surface not considered in the initial set of preferences. The continuous introduction of a small number of random immigrants in the current population (Grefenstette, 1992), for example, has been shown to improve the response of GAs to sudden changes in the objective function, while also potentially improving their performance as global optimizers.

Although there is no hard limit on how much the DM may wander away from preferences set originally, it must be noted that EAs will work on the utility function implicitly defined by the preference vectors the DM specifies. Any EA can only converge to a compromise solution if the DM comes to consistently prefer that solution to any others.

Giving the DM freedom to specify any preferences at any time also raises the question of what information should be stored during a run, so that no trade-off information acquired is lost. From Lemma 4.1, the non-dominated set of a particular problem contains at least one vector equivalent to any vector in the preferred set of the problem, defined by a given preference vector. Therefore, it may be acceptable to store only the non-inferior individuals evaluated during a run of the algorithm. A database of individuals currently non-dominated is also useful in setting the appropriate niche sizes for sharing and mating restriction, since it includes the relevant individuals from previous generations in the niche-size estimation process.

4.5 Simple examples

The examples given in this section are aimed at illustrating some of the points discussed earlier. They are based on the minimization of the two competing objectives

$$\begin{aligned} f_1(x_1,\ldots,x_k) &= 1-\exp\left(-\sum_{i=1}^{k}(x_i-1/\sqrt{k})^2\right) \\ f_2(x_1,\ldots,x_k) &= 1-\exp\left(-\sum_{i=1}^{k}(x_i+1/\sqrt{k})^2\right), \end{aligned}$$

which are not very different from the objective functions considered in subsection 4.3.3, but are defined for any number of decision variables k. The minimum of f_1 is located at $(x_1,\ldots,x_k)=(1/\sqrt{k},\ldots,1/\sqrt{k})$ for all k, and that of f_2 is located at $(x_1,\ldots,x_k)=(-1/\sqrt{k},\ldots,-1/\sqrt{k})$. Due to symmetry in the two functions, the Pareto-optimal set clearly corresponds to all points on the line defined by

$$x_1=x_2=\cdots=x_k \quad \wedge \quad -1/\sqrt{k}\le x_1\le 1/\sqrt{k}\,.$$

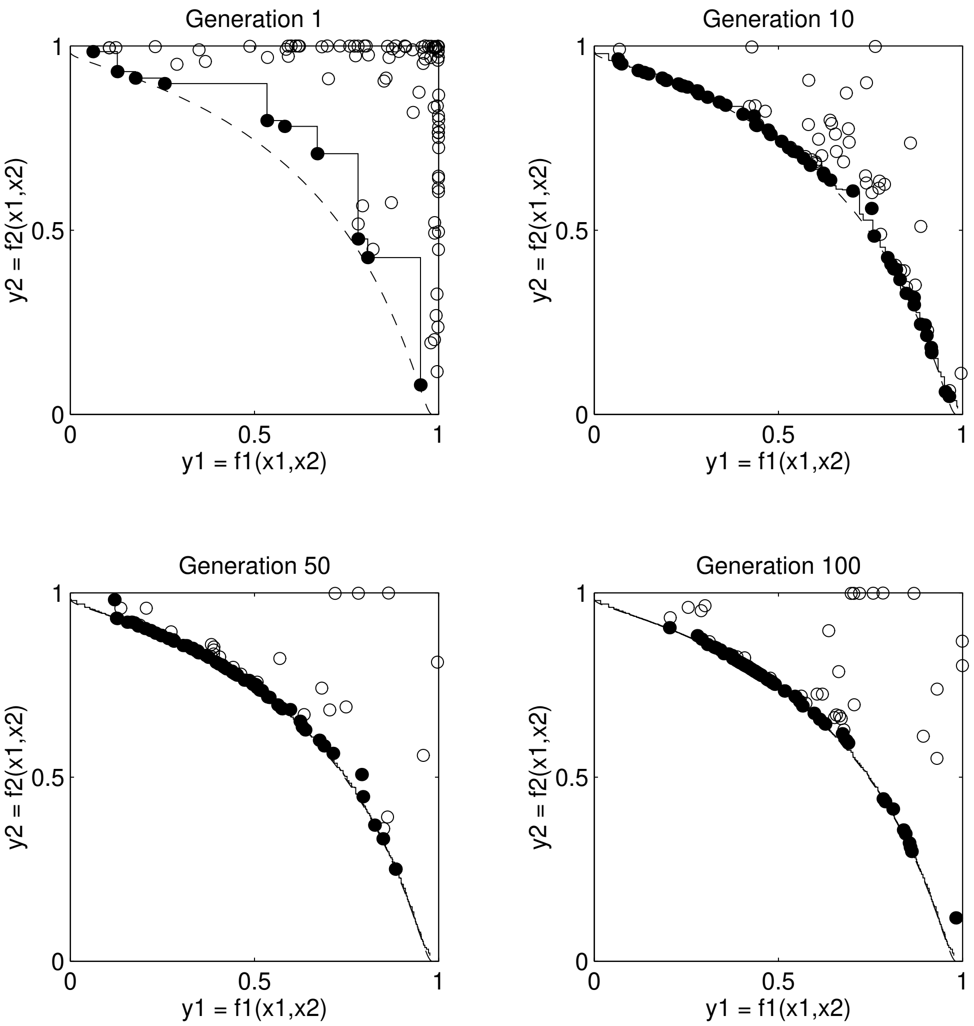
Since the Euclidean distance between the two optima remains constant for any number of decision variables k considered, the problem admits the same non-dominated set for all k.

A simple genetic algorithm with a population size of 100 individuals, binary chromosomes, reduced-surrogate shuffle crossover and binary mutation will be the basis for the GA runs that follow. Multiobjective ranking is performed as described earlier in this Chapter. Decision variables are Gray-encoded as 16-bit strings in the interval $[-2, 2)$ and concatenated to form the chromosomes.

4.5.1 Simple GA with Pareto-ranking

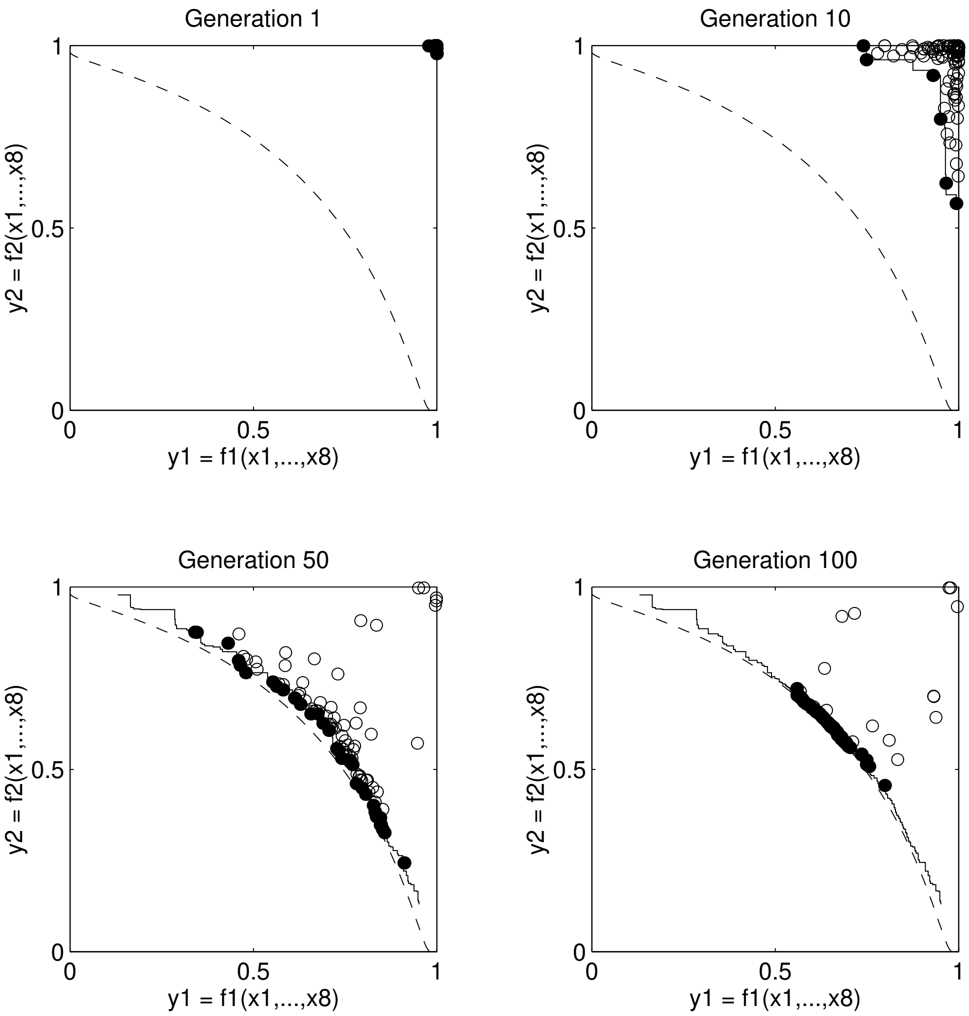
The evolution of the population of the simple GA with Pareto-ranking on the problem for $k = 2$ is illustrated in Figure 4.11. Non-dominated individuals are marked with filled circles ($\bullet$) and other individuals with hollow circles ($\circ$). Following the initial generation, repeated selection of non-dominated individuals and production of offspring soon produces a reasonable description of the trade-off surface of the problem (generation 10). However, letting the GA run longer, it can be seen that the population continues to converge towards an arbitrary region of that surface, due to genetic drift. Thereafter, it becomes very unlikely that the sampling of other regions of the trade-off will be improved at all.

This effect becomes more important as the number of variables increases while maintaining the population size fixed. Figure 4.12 shows the evolution of the trade-off surface for $k = 8$. Not surprisingly, the quality of the trade-off improves more slowly. In addition to the search space being much larger, points along the trade-off surface now differ from each other in many more bit positions, making the search more difficult and providing even more scope for genetic drift to occur. At generation 100, the population can be seen to cover a region of the trade-off surface smaller than in the previous example, while the best non-dominated points ever found are also farther from optimal.



• Non-dominated individuals ◦ Dominated individuals
— Best trade-off so far (cumulative) – – Actual Pareto set

Figure 4.11: Evolution of a trade-off surface (f_1 vs. f_2, 2 decision variables). Simple GA with Pareto-ranking.



• Non-dominated individuals ◦ Dominated individuals
— Best trade-off so far (cumulative) – – Actual Pareto set

Figure 4.12: Evolution of a trade-off surface (f_1 vs. f_2, 8 decision variables). Simple GA with Pareto-ranking.

4.5.2 Simple GA with Pareto-ranking and sharing

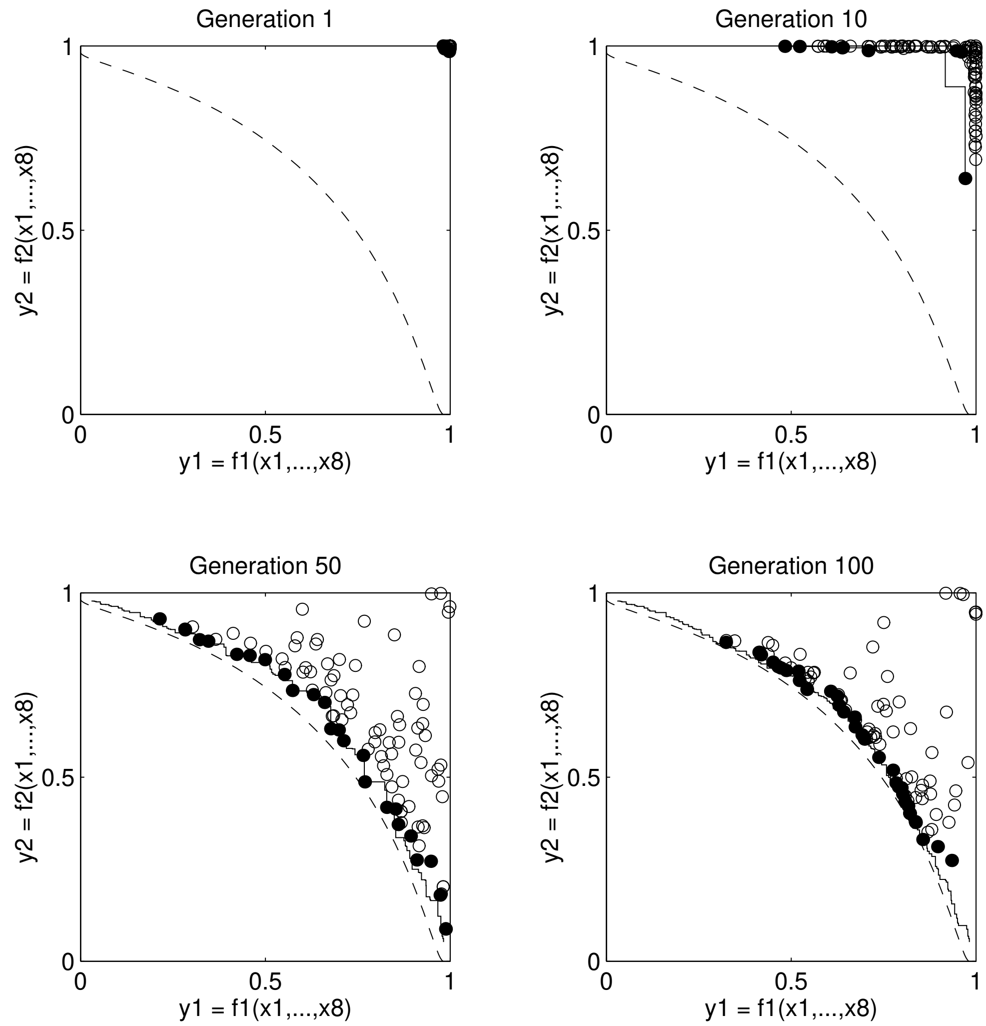
Sharing (Figure 4.13) can be seen to promote diversity among the population, and to maintain it to some extent as the search proceeds. However, the trade-off surface produced is still not as desired. In particular, a larger number of dominated individuals can be seen to exist at generations 50 and 100, and the population still does not cover the whole trade-off surface well. As predicted earlier, promoting diversity through sharing has caused more low-fitness individuals to be produced through recombination, making improvement of current non-dominated solutions slow.

4.5.3 Simple GA with Pareto-ranking, sharing and mating restriction

Introducing mating restriction can be seen to clearly improve the algorithm's long-term performance (Figure 4.14). The population is now able to cover virtually the whole trade-off surface, while remaining more or less uniformly distributed across its length. The best non-dominated points ever found now approach the theoretical trade-off better, and, as long as the population remains distributed across the whole trade-off surface, can be expected to improve as the algorithm runs.

4.5.4 Progressive articulation of preferences

If, during a GA run, the DM decides that some regions of the trade-off surface are of not of practical interest, it may choose to set the preference vector in order to reflect that. Considering a new run of the previous example, the population at generation 20 already indicates the trade-off between f_1 and f_2, though in a rough and sub-optimal manner (first plot in Figure 4.15). At this stage, new goals g_1 and g_2 may be set for the two objectives. Running the GA for another



• Non-dominated individuals ○ Dominated individuals
— Best trade-off so far (cumulative) – – Actual Pareto set

Figure 4.13: Evolution of a trade-off surface (f_1 vs. f_2, 8 decision variables). Simple GA with Pareto-ranking and sharing.

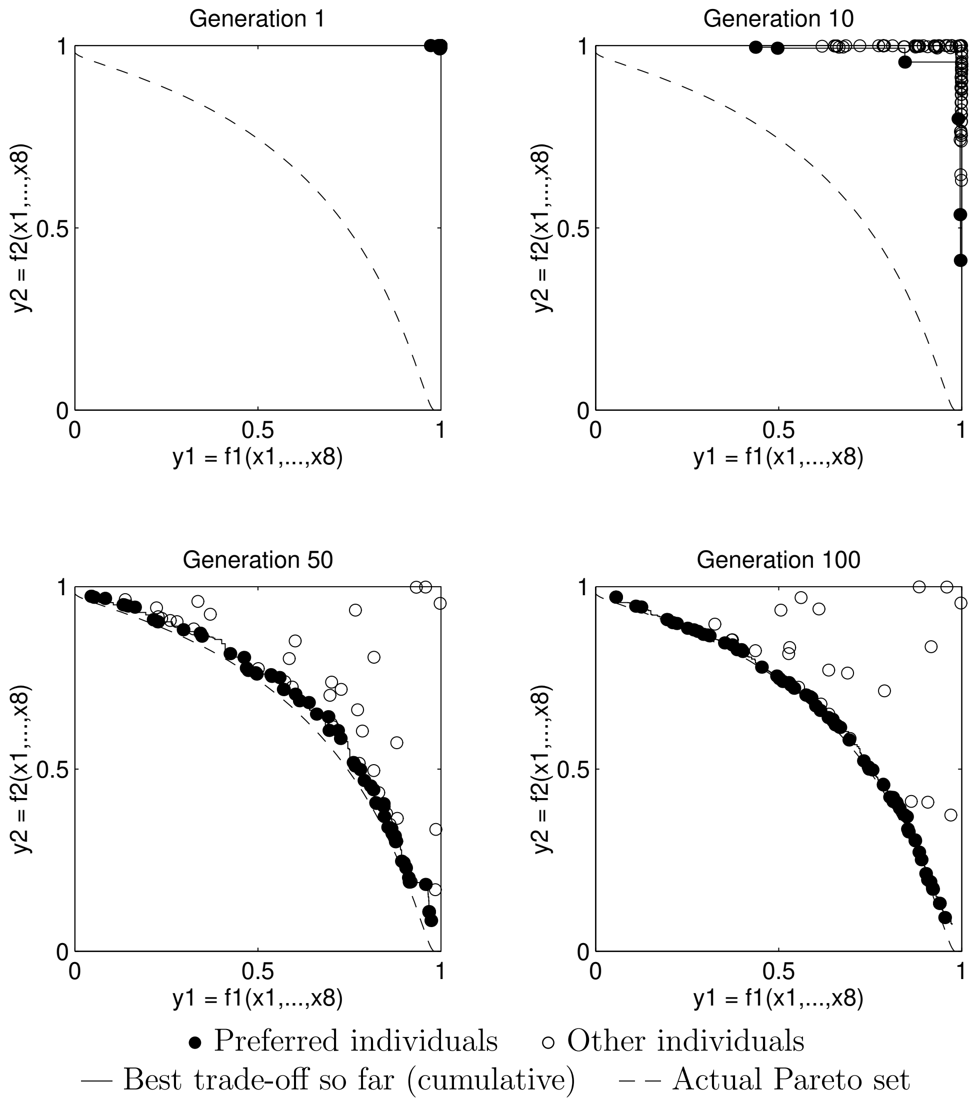
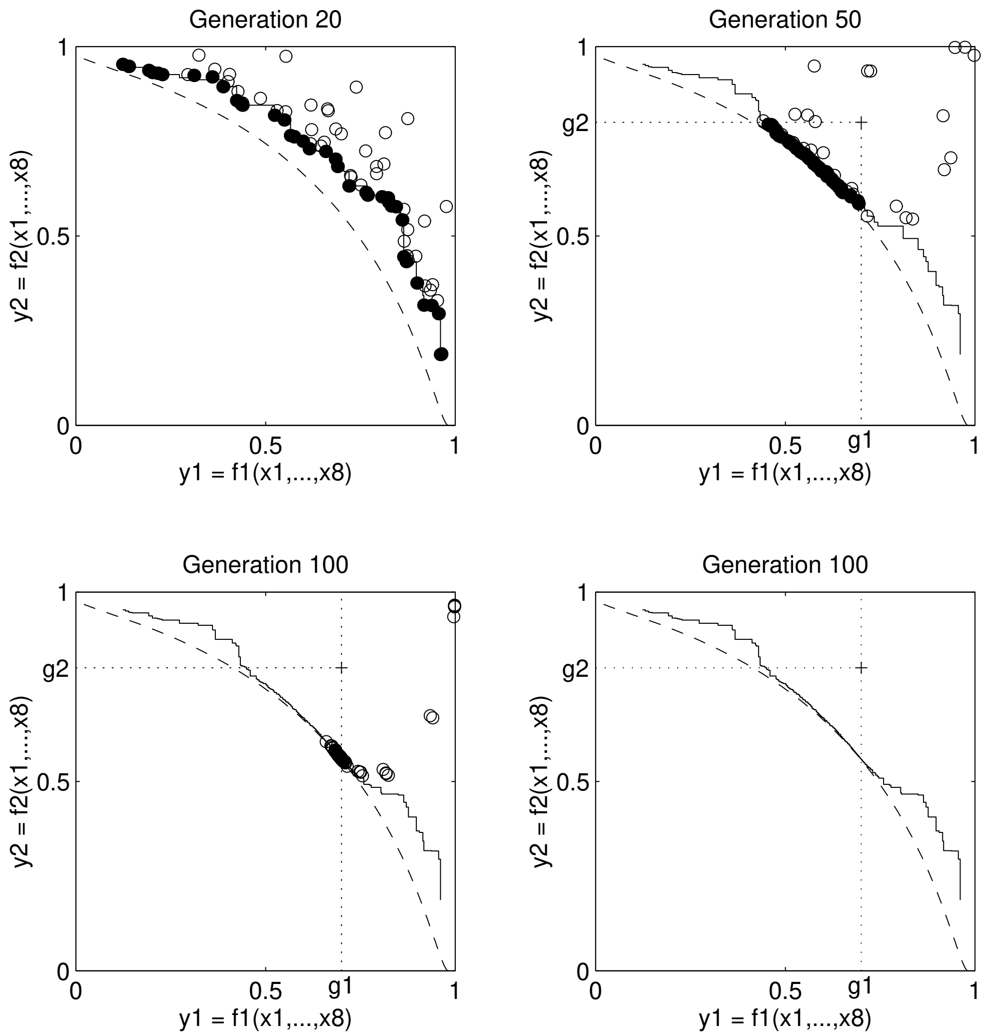


Figure 4.14: Evolution of a trade-off surface (f_1 vs. f_2, 8 decision variables). Simple GA with Pareto-ranking, sharing and mating restriction.



• Preferable individuals ○ Non-preferable individuals
— Best trade-off so far (cumulative) – – Actual Pareto set

Figure 4.15: Progressive articulation of preferences. Computational effort is concentrated on regions of the trade-off surface attractive to the DM.

30 generations (second plot), it can be seen that not only is the population now concentrated on the relevant portion of the trade-off curve, but also the set of all non-dominated individuals ever evaluated during the run is of better quality in this region. Increasing the priority of f_1 without changing the goals, and continuing to run the GA, the population is driven towards the point corresponding to $y_1 = g_1$ (third plot), further increasing the quality of the final trade-off approximation around this point.

The quality of the global trade-off approximation produced at the end of a run can be seen in the fourth plot in Figure 4.15 to vary locally according to how the DM articulated preferences during that run, and to reflect the concentration of search effort in the preferred regions. Reciprocally, the DM's settings for the preference vector will generally be influenced by the progress of the search.

4.6 Conclusions

This Chapter began with a critical overview of evolutionary approaches to constrained and multiobjective optimization to date. A generalized view of evolutionary multiobjective optimization was then conceived as the interaction between two complementary entities: a Decision Maker, responsible for the evaluation of the relative cost of candidate individuals, and a search algorithm, responsible for the production of new candidates.

In line with this generalization, the concept of preferability given a preference vector comprising goal and priority information was then proposed. It was formalized in terms of a transitive relational operator, leading to a strategy for the scale-independent, multiobjective ranking of a whole population. The concept encompasses several multiobjective and constrained formulations commonly used, including Pareto, lexicographic and goal programming approaches. The cost landscapes resulting from the ranking approach proposed can be interpreted in terms of the probability of improvement via random search, a notion on which recent theoretical results in the study of single-objective artificial evolution are based (Altenberg, 1994).

A multiobjective genetic algorithm has also been formulated, making use of the preferability relation, fitness sharing, and mating restriction techniques. The implementation of such techniques in the objective domain has been proposed, and expressions to estimate the corresponding niche sizes have been derived.

Sample genetic algorithm runs on a simple biobjective problem illustrated some of the difficulties posed by Pareto and preferability-based assessment of solutions, and how the techniques proposed address them.

The MOGA, as formulated in this Chapter, makes no assumption about the nature of the decision variables involved in the search process. Therefore, the considerations made and the techniques proposed are largely domain independent, being applicable to virtually every domain of application of GAs and other evolutionary approaches. In the next two Chapters, two parameter optimization problems and a combinatorial (subset selection) problem, all drawn from the control engineering field, are approached in a multiobjective way by means of genetic algorithms. The very different nature of the two sets of problems is almost irrelevant to the multiobjective nature of the GAs used.

Chapter 5

Multiobjective Controller Parameter Optimization

5.1 Introduction

Optimal controller design has been approached from many perspectives, ranging from the finding of analytical expressions for the optimal controller (e.g., LQG design and $\mathcal{H}_\infty$ design (Kwakernaak, 1986)), to the formulation of the design problem as a non-linear programming problem and the consequent application of numerical optimizers to produce optimal or near-optimal solutions (Hancock, 1992). Somewhere in between these two extremes lie intermediate approaches where, in certain cases, controller design can be cast as a linear, or at least convex, optimization problem, and thus more efficiently solved (Boyd *et al.*, 1988).

In this Chapter, the application of a multiobjective genetic algorithm to two controller design problems extends the non-linear optimization approach by direct handling of meaningful objectives. In fact, rather than trying to guarantee favourable properties of the design objectives prior to optimization, such as continuity or smoothness, objectives are stated in a form which simply reflects the natural way in which the designer perceives the problem. The exploration of the trade-offs between the various design objectives is left to the GA, at times confirming, and quantifying, the designer's unformalized knowledge (or intuition), but also possibly revealing unexpected aspects of the problem under study. The "wild" nature of genetic search imposes a strong demand on the proper formalization of the design objectives, a lot more so than on the mathematical properties that they may or may not possess.

In the first case-study, a controller is to be found for a non-minimum phase, linear time-invariant plant. The controller should be optimal or near optimal with respect to both noise sensitivity and robustness criteria, while remaining open-loop stable and maintaining a given complexity. Low complexity is one of those objectives which, although present in almost all practical applications, can make mathematical analysis difficult. For this reason, it is often addressed, a posteriori, by means of model reduction techniques (Hutton and Friedland, 1975).

The second case-study is concerned with the low-pressure spool speed governor in a non-linear model of a Pegasus gas turbine engine. The parameters of the controller are to be set so as to minimize several performance measures in both the time and the frequency domains, some of which require the actual simulation of the model.

Since the two application examples reduce to parameter optimization problems, both make use of exactly the same genetic algorithm. The implementation of the GA will, therefore be described first, in the next Section. Each application example is then addressed in detail, separately. The Chapter concludes with a discussion of the results.

5.2 Implementation of the genetic algorithm

All of the functions needed to set up a simple genetic algorithm, such as single-objective ranking, sampling, and various types of crossover and mutation, were implemented as MATLAB (The MathWorks, 1992a) M-files. This core of GA functions formed a prototype Genetic Algorithm Toolbox for use with MATLAB which has meanwhile been considerably extended (Chipperfield *et al.*, 1994), and is currently used in over 100 sites worldwide.

Multiobjective ranking, sharing and mating restriction, as described in Chapter 4, were implemented in C and compiled into MEX (Matlab EXecutable) files. Interpreted MATLAB was either too slow for these tasks (for-loop implementation) or too memory-hungry (vectorized implementation), especially as the number of GA individuals involved increased.

A graphical user interface (GUI) was written as an M-file, making use of the graphical capabilities introduced in Version 4 of the MATLAB package. The resulting genetic optimization environment was thus fully integrated with the many toolboxes available for MATLAB, most importantly the Control Systems Toolbox and SIMULINK (The MathWorks, 1992b), which were extensively used in the implementation of the various objective functions.

The MOGA consisted of a fairly standard generational GA, only modified to incorporate multiobjective ranking, sharing and mating restriction as described in Chapter 4. Also, in order to make the GA more responsive to on-line preference changes (subsection 4.4.3), a small proportion ($\delta = 0.1$) of random immigrants was inserted in the population at each generation.

5.2.1 Chromosome coding

In both case-studies, decision variables were encoded as binary strings, using Gray coding, and then concatenated to form the chromosomes. This allowed most of the GA to be set up in a fairly standard way, and effort to be concentrated on its multiobjective aspects. Details such as the precision to which decision variables were encoded will be given in the sections corresponding to each case-study.

5.2.2 Individual evaluation

Chromosomes were decoded and the various objective functions evaluated at the points obtained. After the first generation, only those offspring which were actually changed by the genetic operators were decoded and re-evaluated, as suggested by Oliveira *et al.* (1991). This simple "caching" mechanism reduced the number of objective function evaluations by between 20% and 25%.

The necessary scalarization of the objective vectors was performed by ranking the population according to the concept of preferability, as described in Section 4.3.

5.2.3 Fitness assignment

Once the multiobjective ranks were calculated, individuals were sorted by rank value, for fitness assignment. In a purely generational GA with a constant population size N, the average number of offspring per individual is 1 and, therefore, the number of offspring expected by each individual equals its relative fitness. However, in an implementation where δN individuals consist of random immigrants and only $(1-\delta)N$ individuals in each new generation are the result of selection, the relative fitness of the best individual must increase with δ for its expected number of offspring to remain constant. For example, in order for the best individual to produce on average μ offspring ($\mu = 2$ is a typical setting), its relative fitness should be $s = \mu/(1-\delta)$. Having decided on the value of s, the rest of the population was assigned fitness exponentially, as has been explained in subsection 2.3.2. When multiple individuals had the same multiobjective rank, their fitness was averaged so that all of them were assigned the same raw fitness prior to sharing, as shown previously in Figure 4.7.

Sharing was performed as follows. The *niche count* of each individual was initially set to zero and then incremented by a certain amount for every individual in the population with the same multiobjective rank, including itself. The contribution of an individual to another's niche count was dictated by a *sharing function*, which, in this case, was a triangular function of their mutual distance in objective space (Goldberg and Richardson, 1987). The original fitness values were then weighted by the inverse of the niche counts and subsequently normalized by the average of the weights within each rank, before selection.

The sharing parameter σ_{share} was estimated at each generation given the number of preferable individuals in the population and, for each objective, the range those individuals defined in normalized objective space. Two possibilities were investigated for the normalization of the objectives.

The first possibility consisted of normalizing each objective by the corresponding range defined in objective space by the current set of preferable individuals. This is equivalent to computing σ_{share} assuming $\Delta_j = 1$ for all j and then denormalizing the σ_{share} obtained. This seemed to encourage the population to expand and cover more of the trade-off surface, by penalizing middling individuals slightly more than those at the extremes. However, in certain circumstances (non-attainable goals, for example), this sometimes led to the insufficient production of middling individuals.

The second possibility was to normalize the objectives by the ranges defined by the set of preferred non-dominated individuals accumulated up to and including the current generation. These values are less affected by variations in the current population, and the GA seemed to behave better in the circumstances mentioned above. A combination of both methods, where each objective was normalized by the smaller of the two (current and cumulative) ranges, appeared to perform best.

After sharing, SUS (Baker, 1987) was used to select $(1-\delta)N$ individuals from the old population, which were then recombined and mutated to produce $(1-\delta)N$ offspring. The remaining δN individuals were simply generated at random and added to the population of offspring to form a new generation. Immigrants did not take part in recombination or mutation at this stage.

5.2.4 Recombination

Once the $(1-\delta)N$ parents of the next generation were selected from the old population, they were paired up and recombined with high probability (0.7, also a typical value). Mating restriction was implemented by forming pairs of individuals within a distance $\sigma_{\mathrm{mate}} = \sigma_{\mathrm{share}}$ of each other in objective space, where possible (Deb and Goldberg, 1989). Reduced-surrogate shuffle crossover was used for recombination (see subsection 2.3.4).

5.2.5 Mutation

After recombination, each bit in the chromosomes was mutated independently with a given probability, which was set according to the considerations made in Chapter 2 concerning the survival of a single best individual in the population.

The probability of a length ℓ chromosome not being modified by independent bit mutation is

$$P_{\mathrm{s}} = (1 - p_{\mathrm{m}})^{\ell} ,$$

where p_{m} represents the bit mutation probability. In the absence of crossover, the probability of survival P_{s} should be no less than the inverse of the expected number of offspring of the best individual. It follows that

$$p_{\mathrm{m}} \le 1 - \mu^{-1/\ell} .$$

In the presence of crossover, the actual p_{m} should be somewhere below the limit. For $\mu = 2$, setting

$$p_{\mathrm{m}} = 1 - (\alpha\mu)^{-1/\ell} ,$$

where $\alpha = 0.9$, was found to perform well. Note that in the presence of mating restriction, crossover tends not to be too destructive. No fine tuning of these parameters was attempted.

5.3 Mixed $\mathcal{H}_2/\mathcal{H}_\infty$ reduced-order controller design

Theoretical results (Boyd *et al.*, 1988) show that, in the linear time-invariant (LTI) case, objectives such as rise time, settling time, overshoot, asymptotic tracking, decoupling and regulation, gain and phase margins, small disturbance response and bounds on frequency response magnitudes are convex with respect to the closed-loop transfer function. Essentially, this means that if two closed-loop systems satisfy certain goals for these objectives, the system which is the average of those two closed-loop systems will also satisfy those goals.

The same results also show that, once a stabilizing controller is found for a given plant, the design problem can be formulated as a multiobjective optimization problem convex in Q, a transfer function design parameter which is a mapping of all stabilizing controllers for that plant. However, when addressed in the controller parameter domain, the same design problem is no longer necessarily convex and, in general, it may be difficult to solve.

Other objectives, such as open-loop controller stability and low controller complexity, may not result in convex optimization problems and, therefore, are not included in the results mentioned above. Non-linear systems pose similar difficulties, as their mathematical treatment is generally more complex.

Barratt and Boyd (1989) have proposed a simple LTI controller design problem for which they computed exact trade-offs involving noise sensitivity and robustness measures. These were *global* trade-offs, defined over the space of all stabilizing controllers, and potentially achievable only with controllers of arbitrary complexity. Such trade-offs can serve as maximum performance bounds against which controllers of any complexity, obtained by any design method, can be compared. Although generally complex, the corresponding controllers might, in certain cases, be directly implementable. Alternatively, sufficiently simple con-

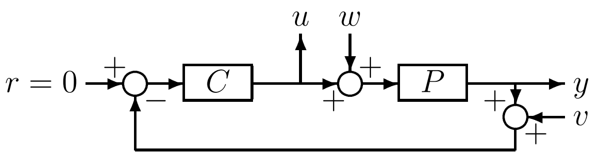


Figure 5.1: Closed-loop system.

trollers could be derived from them through model reduction techniques.

In this section, an instance of the same problem, but where controller complexity is fixed, is considered. The trade-offs found depend directly on the predefined complexity of the controllers sought. The outcome are families of *final*, reduced-order controllers whose performance is well known as a consequence of the optimization process.

5.3.1 The design problem

The problem (Barratt and Boyd, 1989) consists of designing a discrete-time regulator for the single-input single-output plant

$$P(s) = \frac{1}{s^2} \cdot \frac{4-s}{4+s} \,. \tag{5.1}$$

Such a double integrator plant with excess phase provides a simple but realistic basis for illustrating design trade-offs. The discretization of the plant using a zero-order hold at 10 Hz gives the transfer function

$$P(z) = \frac{-0.00379\,(z-1.492)\,(z+0.7679)}{(z-1)^2\,(z-0.6703)} \,. \tag{5.2}$$

The complete regulator system is presented in Figure 5.1. The discrete-time outputs u and y are, respectively, the actuator and the plant outputs. The discrete-time inputs w and v represent actuator and sensor noise, and are considered to be driven by zero-mean, independent, white noise with root-mean-square (rms) values of 10 and 1, respectively.

5.3.2 Design objectives

A number of objectives including noise sensitivity and robustness measures is considered here. A controller of given complexity is sought which represents a compromise between some or all of these objectives.

Closed-loop stability This is probably the most basic objective to be satisfied. If no stabilizing controller is known for a particular plant, non-stabilizing controllers can still be ranked according to how far from being stabilizing they are, as indicated by, for example, the maximum of the magnitudes of the poles of the corresponding closed-loop system. This value can be computed easily in the MATLAB environment.

Stabilizing controllers can be found by minimizing the degree of instability of the closed-loop system.[1] Once stability is achieved, the remaining objectives become active. Since closed-loop stability is an absolute design requirement, the maximum pole magnitude of the closed-loop system is set up as a high priority objective, or constraint, with an associated goal of 1.

Output and actuator variance The trade-off between steady-state output and actuator variances due to the presence of process (actuator) and measurement (sensor) noise, if considered in isolation from other objectives, can be computed analytically from LQG theory. The conventional approach is to find the minimum of the linear combination of the two variances

$$J = \lim_{k \to +\infty} E\{y_k^2 + \rho u_k^2\} \tag{5.3}$$

for various settings of the parameter ρ until suitable values of output and actuator variance are found.

[1]Barratt and Boyd (1989) designed a simple lead-lag regulator for this particular plant, in order to provide the initial stabilizing controller required by their approach.

In MATLAB, the variances $\lim_{k\to+\infty} Ey_k^2$ and $\lim_{k\to+\infty} Eu_k^2$ for a given discrete-time controller-plant setup can be computed as two separate objectives by using the function `dcovar`, available in the Control Systems Toolbox.

The order of the regulators found analytically is generally dictated by the order of the plant. The need for simpler controllers has prompted much interest and work in the area of model reduction. In fact, no analytical solution is known for controllers with arbitrarily fixed complexity. Their design requires the use of optimization techniques.

Sensitivity to additive loop perturbations The M-circle radius, defined as the minimum distance from the Nyquist plot of the loop gain $PC(\exp j\omega T)$ to the critical point -1, is a measure of robustness which combines both gain and phase margins. It relates to the maximum sensitivity of the system, defined as

$$\|S\|_\infty = \left\| \frac{1}{1+PC} \right\|_\infty \tag{5.4}$$

in the following way:

$$M = 1/\|S\|_\infty . \tag{5.5}$$

Therefore, minimizing the maximum sensitivity $\|S\|_\infty$ corresponds to maximizing the M-circle radius, and thus the stability margin of the system. Variations in the loop gain may appear, for example, as a consequence of variations in the parameters of the system.

A function to compute the $\mathcal{H}_\infty$ norm of discrete-time LTI systems was written in MATLAB. The algorithm implemented was the discrete-time equivalent of the fast, iterative algorithm proposed by Bruinsma and Steinbuch (1990) for continuous-time systems.

Sensitivity to additive plant perturbations This second measure of robustness expresses the ability of the regulator to maintain closed-loop stability in the presence of stable additive plant perturbations. Characterizing plant perturbations ΔP in terms of the maximum magnitude of their frequency response, the smallest stable perturbation D which will destabilize the closed-loop system is known to be the inverse of the maximum magnitude of the closed-loop transfer function from r to u, i.e.,

$$1/D = \left\| \frac{C}{1+PC} \right\|_\infty . \tag{5.6}$$

Additive plant perturbations may arise from inaccurate modelling of the plant, either due to tolerances in the parameters or ignored plant dynamics.

Open-loop controller stability It is often required that controllers be open-loop stable. This constraint can be implemented simply by requiring that the maximum pole magnitude of the controller be less than unity.

5.3.3 Parameter encoding

Controllers were parameterized in terms of their roots (poles and zeros) and their gain. The fact that a large number of different systems, represented by possibly very different pole-zero patterns, can exhibit the same input-output behaviour within arbitrary finite accuracy (Hippe and Stahl, 1987) makes the pole-zero domain very rich in approximate solutions for the GA to explore.

Zeros and poles were defined through pairs of Gray encoded real parameters, respectively the average, α, and the deviation from the average, β, of each pair of roots, as suggested by Kristinsson and Dumont (1992). A positive deviation indicates real roots and negative deviation indicates complex conjugate roots.

A pair of zeros and a pair of poles were encoded as two α and β pairs in the interval between -1 and 1. Most controllers defined in this way are open-loop stable and minimum phase. The gain was also Gray encoded, in the interval between 0 and 100. 16 bits were used for each parameter, which lead to 80-bit

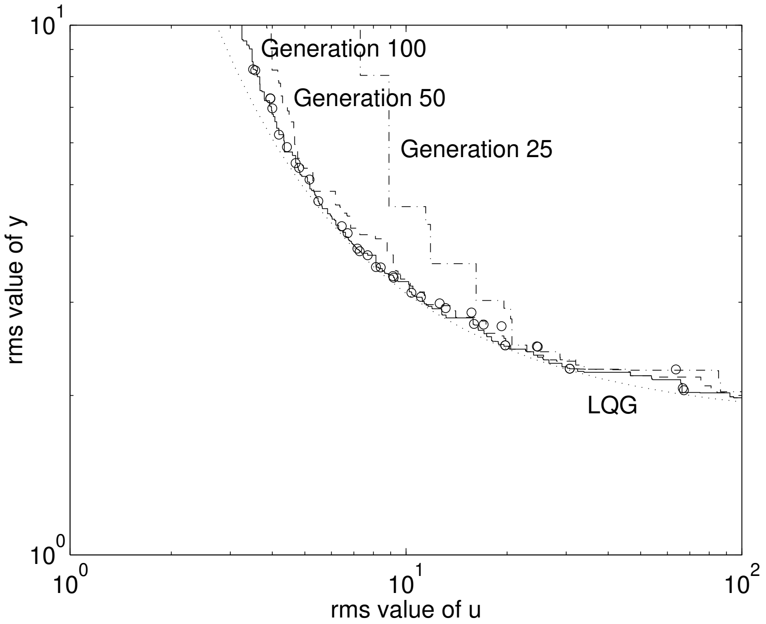


Figure 5.2: Evolution of the noise sensitivity trade-off (single run).

long chromosomes.

5.3.4 Noise sensitivity trade-off

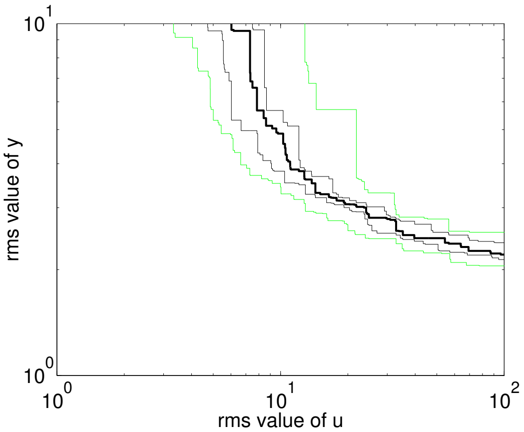
Figure 5.2 shows how the GA evolved a family of second order controllers for a first instance of this problem. In addition to the closed-loop stability constraint, only the two objectives relating to noise sensitivity, i.e., the actuator and output variances, were included. Since the theoretical LQG trade-off surface, known from Barratt and Boyd's work and easily reproducible with MATLAB, was best visualized on logarithmic axes, the logarithm of the two objectives was actually used for the purposes of sharing and mating restriction. The region of the trade-off curve to be evolved was delimited by setting the goal vector to $u_{\rm rms} = 100$ and $y_{\rm rms} = 10$.

In a sample run, the set of preferred individuals accumulated over the first 25 generations already provided a rough description of the trade-off surface (the dash-dotted line), which was improved as the search progressed. The Figure also shows the non-dominated individuals present in the hundredth generation (marked $\circ$). Note how the population is more or less uniformly distributed along the trade-off surface, as desired. It is also worth noting that, in this example, second order controllers come very close to the best (dotted line) that can be achieved with any stabilizing controller (LQG regulators for this plant are third order).

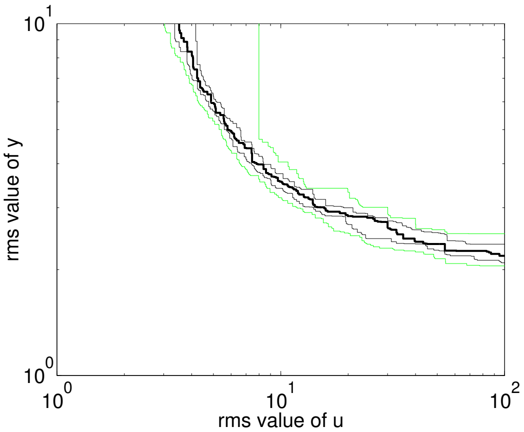
Results obtained from 13 independent runs of the algorithm on this instance of the problem provide a clear estimate of how likely to be attained in a run of the GA the various regions of the objective space were. The plots, which were produced from the sets of non-dominated individuals found up to and including generations 25, 50, 75 and 100 as discussed earlier in subsection 3.4.3, are presented in Figure 5.3.

In all plots, the lower grey line indicates the best trade-off approximation known as a consequence of all the runs, while the upper grey line delimits the set of goal vectors attained in all the runs. The thick black line provides an estimate of the family of possible goal vectors (or requirements) which, each on its own, would have a 50% chance of being attained. The thin black lines provide analogous estimates, but for the 25% and 75% chance levels.

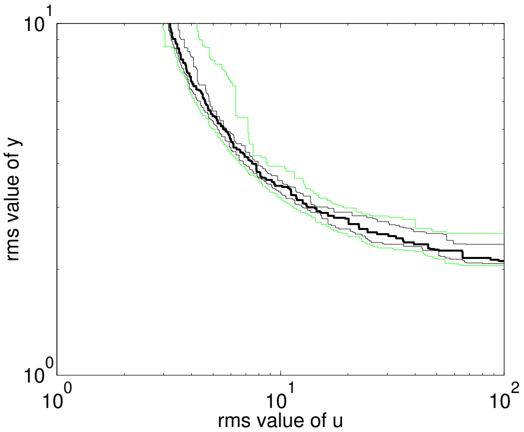
As the number of generations increased, the area delimited by the two grey lines can be seen to have become progressively narrower. This shows both that the quality of the non-dominated solutions produced by the GA did not vary much between runs, *and* that a good coverage of the trade-off surface was more or less consistently achieved. In addition to that, the 50%-line can be seen to be generally closer to the lower grey and black lines than to the corresponding upper lines, especially in the lower-right region of the trade-off surface. This skewness is an indication that most runs approximated most of the trade-off surface well.



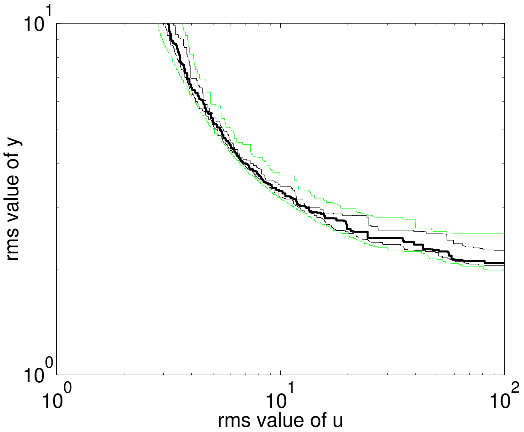
After 25 generations



After 50 generations



After 75 generations



After 100 generations

Figure 5.3: Evolution of the noise sensitivity trade-off (13 runs).

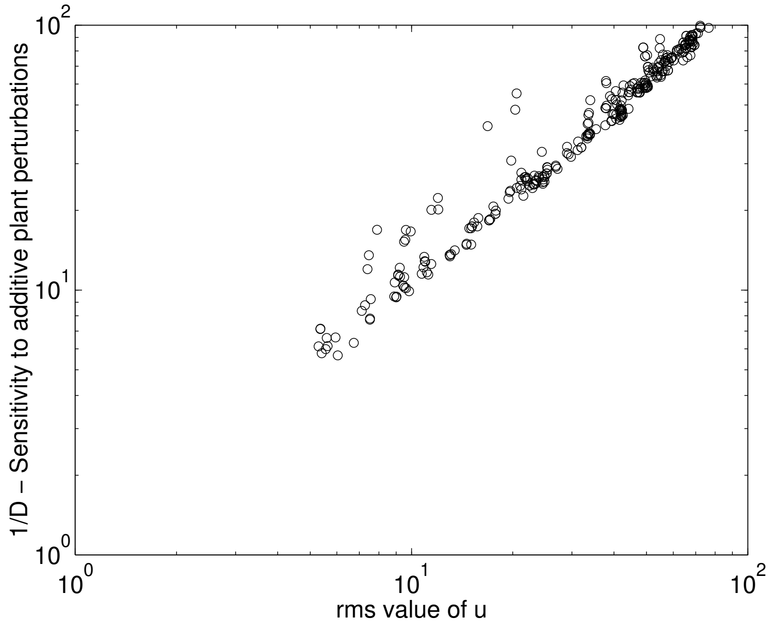


Figure 5.4: Two largely non-competing objectives.

5.3.5 Trade-offs involving noise sensitivity and robustness measures

Noise sensitivity can also be traded off against the sensitivity to additive loop and plant perturbations simply by including these robustness measures in the ranking of the population. In Figure 5.4, actuator variance is plotted against sensitivity to additive plant perturbations for the set of preferable individuals found up to and including the 60th generation. The strong direct relationship shown between these two objectives confirms the general intuition that increasing the control action to achieve better control may make the closed-loop system less tolerant to variations in the dynamics of the plant. Understanding that these two objectives are largely non-competing is important for the designer, as it conceptually reduces the complexity of the problem.

Suppose the designer could decide upon a maximum sensitivity to plant perturbations of $1/D = 10$. Also, suppose that a corresponding actuator rms value of 10, which can be expected from the graph in Figure 5.4, is acceptable but

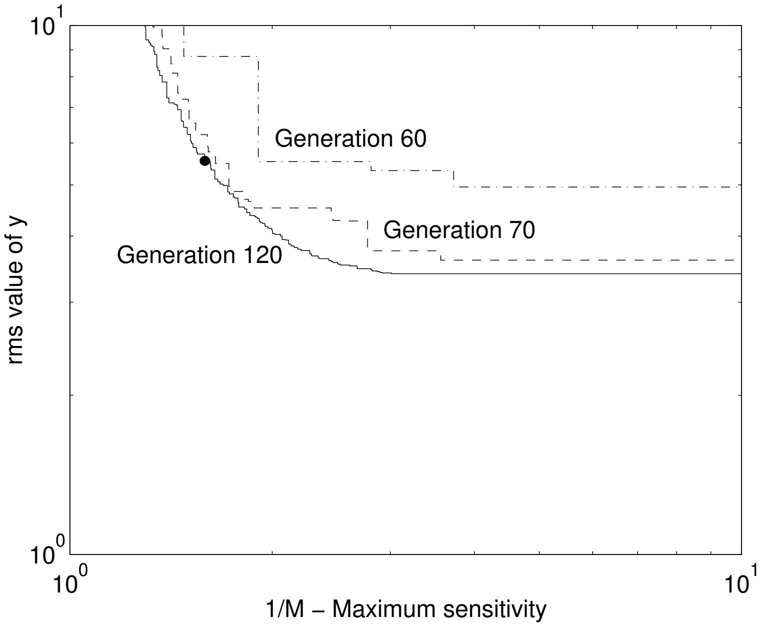


Figure 5.5: Maximum sensitivity versus output variance ($u_{\mathrm{rms}} \leq 10$ and $1/D \leq 10$).

should not be exceeded. Setting these goals and increasing the priority of the corresponding objectives converts the problem into one involving two objectives (output variance and maximum sensitivity) and three constraints (closed-loop stability, maximum actuator variance and sensitivity to additive plant perturbations).

Some information about the trade-off between maximum sensitivity and output variance under the constraints is already available in, and can be extracted from, the set of all non-dominated individuals evaluated, leading to the dash-dotted line in the plot in Figure 5.5.

Continuing to run the GA causes the population to abandon the regions of the search space which violate the new constraints and concentrate on the trade-off between the two soft objectives. The dashed and solid lines in the plot in Figure 5.5 show the improvement achieved in the next 10 and 60 generations, respectively.

The point marked with a filled circle (•) indicates a possible compromise solution. The corresponding controller is characterized by the following zeros (z), poles (p) and gain (k):

$$z_1 = 0.9659 \quad p_1 = 0.5685 + 0.4127i \quad k = 6.770$$
$$z_2 = 0.5879 \quad p_2 = 0.5685 - 0.4127i$$

and can be seen to satisfy the constraints imposed earlier:

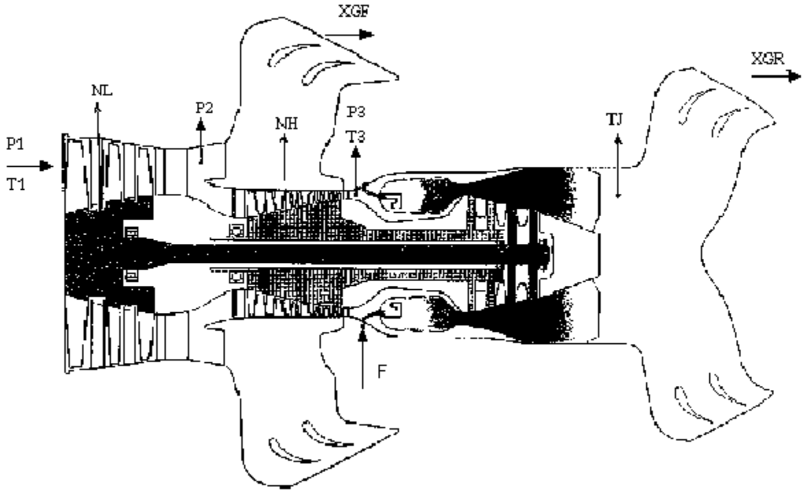
$$\begin{aligned} |p_{\text{c.l.}}|_{\max} &= 0.9481 \leq 1 \\ y_{\text{rms}} &= 5.5495 \\ u_{\text{rms}} &= 8.6030 \leq 10 \\ 1/M &= 1.5865 \\ 1/D &= 9.4689 \leq 10 \\ (|p_{\text{o.l.}}|_{\max} &= 0.7025 \leq 1) \end{aligned}$$

All preferable controllers found were open-loop stable.

5.4 Pegasus GTE controller optimization

The second case-study was based on a non-linear model of a Rolls-Royce Pegasus gas turbine engine (Hancock, 1992; Swansea, 1991). The engine layout and the physical meaning of some of the associated variables are shown in Figure 5.6. The engine model reproduces dynamic characteristic variations for the fuel thrust range over the complete flight envelope.

The model of the full system also includes a fuel system model, stepper motor input and digital fan speed controller, and is shown in Figure 5.7. The demand in low-pressure spool speed (NLDEM) is the only control input of the controller. The other inputs allow external perturbations to be applied to the system, but are not used in this work. NLDEM is compared with the actual speed of the low-pressure spool (NL) in order to produce a demand in high-pressure spool speed



F	Fuel flow
NH	High-pressure spool speed
NL	Low-pressure spool speed
P3	High-pressure delivery pressure
P2	Low-pressure delivery pressure
T3	High-pressure delivery temperature
TJ	Jet pipe temperature
XGF	Front nozzle thrust
XGR	Rear nozzle thrust

Figure 5.6: The Pegasus turbofan engine layout.

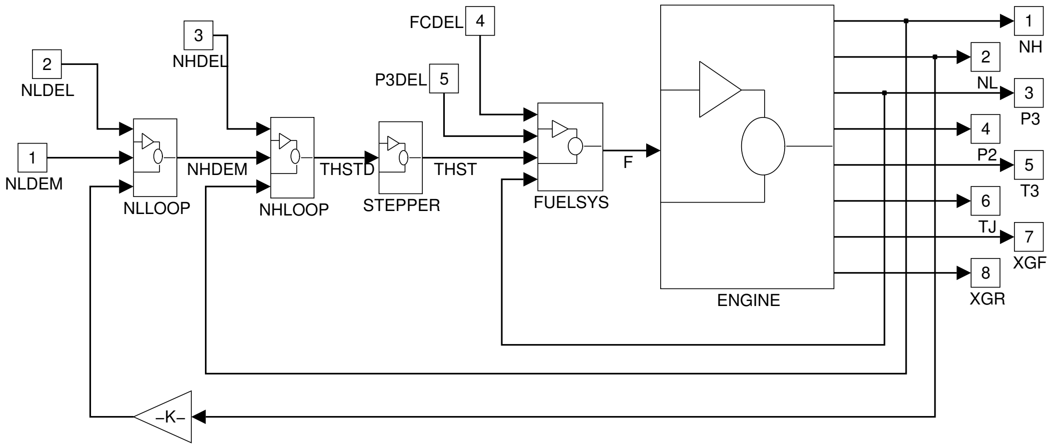


Figure 5.7: The SIMULINK model of the system.

(NHDEM). The desired position of the stepper motor THSTD is then computed from NHDEM and the actual high-pressure spool speed (NH).

The structure of the existing controller was originally determined by Rolls-Royce (Shutler, 1991) on the basis of requirements imposed by its integration with other systems on the aircraft. Both NH and NL are states of the engine model. The remaining outputs are simply non-linear functions of the two states.

5.4.1 The design problem

Each block of the controller, NLLOOP and NHLOOP, is characterized by four parameters, one of which is a real gain value. The other three parameters are time constants. The appropriate setting of such parameters, or design variables, is cast as the problem of optimizing a number of closed-loop performance measures.

In the existing controller, the non-linearity of the plant (i.e., the stepper motor, the fuel system and the engine) is handled by scheduling different controller parameters on-line against the values of NLDEM and NH. Here, a single operating point of the plant is considered, and the design procedure described gives but a single step towards the design of a controller for the full range of operation of the plant. Nevertheless, it provides another good example of how a genetic algorithm can explore the concept of preferability.

5.4.2 Design objectives

The design objectives were extracted from the original stability and response requirements provided by the manufacturers, consisting of a selection of both time and frequency-domain objectives. Frequency-domain measures were computed from the model linearized around the operating point defined by NL = 70%. The time-domain characteristics were derived from the response of the full non-linear model to a small step input (ΔNL = 3%), obtained through simulation.

Closed-loop stability This objective was computed exactly as for the first case-study. For efficiency, the plant model was first linearized around its operating point (NL=70%), prior to any GA run. Controllers to be evaluated were applied to this linear model and the maximum pole magnitude of the resulting system computed. However, this objective was not assigned higher priority than the others. Since a fast time response was also required, the maximum pole magnitude of the closed-loop system should continue to be minimized below unity, which is equivalent to minimizing the largest time-constant of the system.

Gain and phase margins The gain and phase margins are measures of robustness to multiplicative loop perturbations, and are generally independent from the maximum pole magnitude of the closed-loop system. The gain margin for the Pegasus engine is required to be at least 1.5 (3.53 dB), and the phase margin is required to be at least 50 degrees. Both margins are measured by opening the loop after the controller output.[2] The maximization of both margins is implemented as the minimization of their additive inverses.

Rise time The time taken by the closed-loop system to reach, and stay above, 70% of the final output change demanded by a step-input. It is required to be no longer than 0.7 of the time taken by the open-loop engine to reach 70% of a similar output change, following a step in fuel flow. This and the following objectives were computed through simulation of the full model of the system.

Settling time The time taken by the closed-loop system to stay within $\pm 10\%$ of the final output change demanded by a step-input. It is required to be

[2]This is consistent with modelling the uncertainty associated with the fuel system in terms of maximum gain and phase deviations, although it does not directly address the behaviour of the system in the presence of that uncertainty.

no longer than the time taken by the open-loop engine to reach 90% of a similar output change, following a step in fuel flow.

Overshoot The maximum value reached by the output as a consequence of a step input should not exceed the final output change by more than 10%.

Output error Given the presence of a pure digital integrator in the control loop, the steady-state output error of the system is guaranteed to be zero. Consequently, and provided the system is fast enough, the output error measured at the end of the simulation should be close to zero. The associated goal is set to 0.1% of the desired output value.[3]

5.4.3 Parameter encoding

The controller parameters were encoded as 15-digit binary strings each, using Gray coding, and then concatenated to form the chromosomes. Gain parameters in the interval $[10^{-4}, 10^{3})$, and time constants in the interval $[10^{-6}, 10)$ (s), were logarithmically mapped onto the set of possible chromosome values, $\{0, \ldots, 2^{15} - 1\}$, providing a resolution of about 4 significant digits in each case.

5.4.4 A two-variable example

Consider the problem of minimizing the seven objectives by varying only the controller gains, KNL and KNH. Since there are only two variables, the cost surfaces associated with each of the objectives can be produced through gridding and visualized, although this is a computationally intensive task.

Figures 5.8 to 5.11 depict four of the seven single-objective cost surfaces involved in this problem. Due to the bad scaling of some of these objectives, the points on the grid were ranked after evaluation, as discussed in subsection 4.3.3.

[3]This goal was taken from the steady state stability requirement, which states that speed fluctuations should not exceed $\pm 0.1\%$NL.

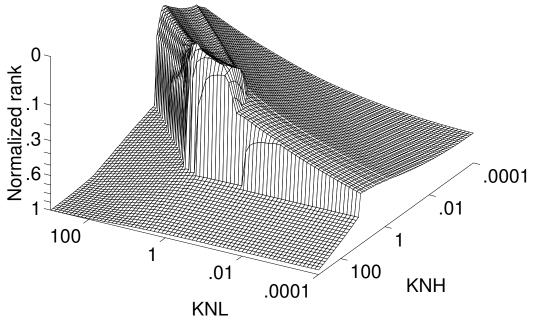


Figure 5.8: The cost surface associated with the maximum pole magnitude of the closed-loop system.

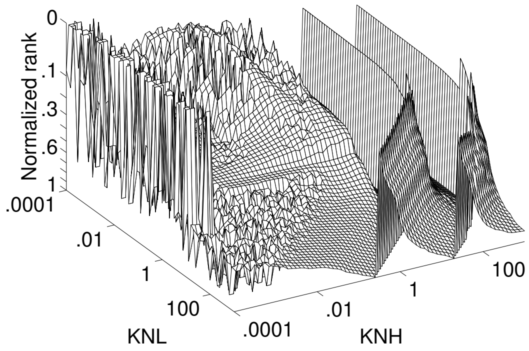


Figure 5.9: The cost surface associated with the phase margin of the closed-loop system.

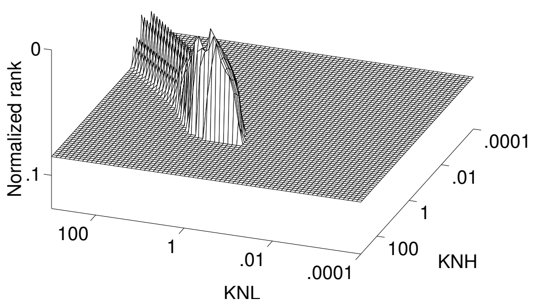


Figure 5.10: The cost surface associated with the settling time of the closed-loop step response.

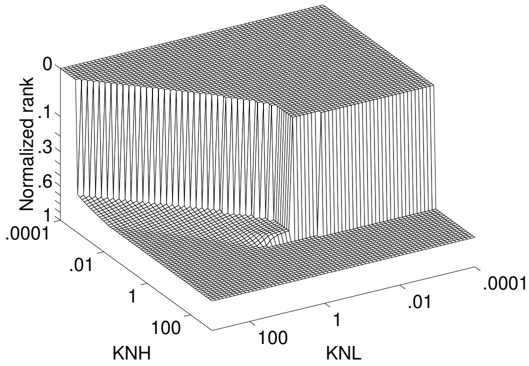


Figure 5.11: The cost surface associated with the overshoot of the closed-loop step response.

Whilst being closer to how a GA with ranking would see them, the resulting surfaces are much easier to visualize than the original ones. In each case, the axes have been rotated so as to highlight the features of each surface.

It is possible to see how some objectives, notably those defined in the time domain, tend to display large flat regions corresponding either to lowest performance, as with settling time (Figure 5.10) and rise-time (not shown), or even to optimal performance in that objective's sense, as with overshoot (Figure 5.11). However, zero (optimal) overshoot means very little on its own: for example, it

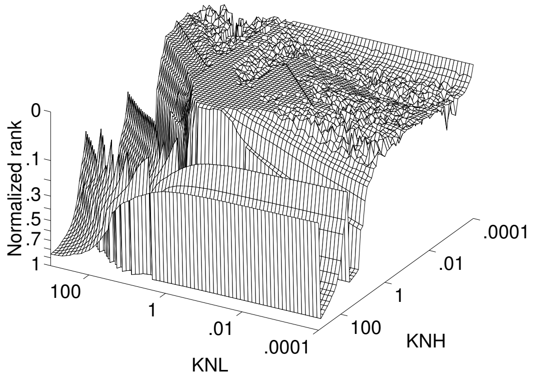


Figure 5.12: A representation of the cost surface associated with the Pareto-based formulation in the absence of preference information.

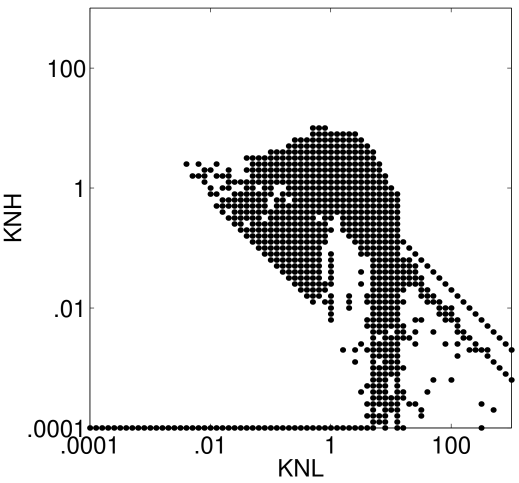


Figure 5.13: An alternative view of the non-dominated set estimated through gridding.

may imply too long a rise-time.

In these regions of the search space, the optimizer must rely on other objectives, such as the maximum closed-loop pole magnitude (Figure 5.8). The closed forms used to compute the gain and phase margins, though resulting in fairly fast numerical algorithms, proved not to be sufficiently numerically robust across the whole search space. This is apparent in Figure 5.9 for the phase margin. Nevertheless, the algorithms did produce accurate results with "sensible" gain settings, and were used instead of more stable, but brute-force, approaches.[4]

One can also estimate and plot a representation of the cost surface associated with the Pareto formulation of the problem when no goals are specified (Figure 5.12). Here, the most striking features are, perhaps, the large area of the search space corresponding to non-dominated solutions and the irregular shape of that region, also shown in Figure 5.13.

Specifying goals for the objectives defines a much smaller area of preferable

[4]A discussion of genetic search with approximate function evaluations can be found in (Grefenstette and Fitzpatrick, 1985)

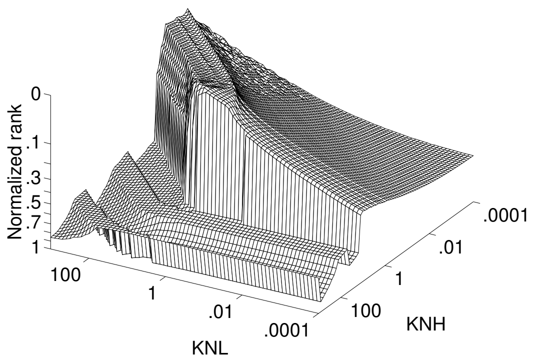


Figure 5.14: A representation of the cost surface associated with the formulation including goal information.

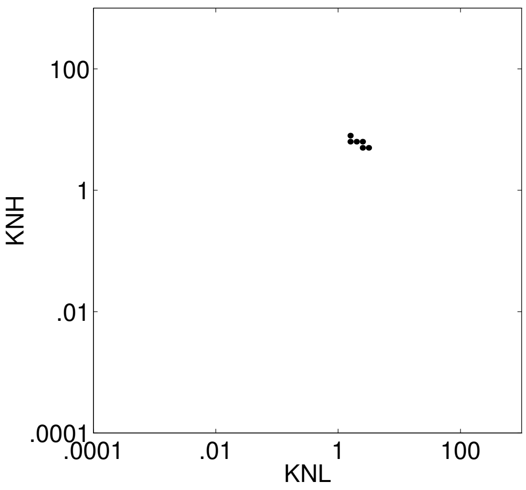


Figure 5.15: An alternative view of the preferable set estimated through gridding.

solutions (Figures 5.14 and 5.15), whose dimensions approach the resolution of the grid. As it turns out, none of these points is, in fact, satisficing. Actual preferred solutions, whether satisficing or not, will probably be confined to an even smaller region of the search space.

5.4.4.1 Genetic algorithm results

A genetic algorithm with a population size of 100 individuals was run on this problem five times, each time for 50 generations. Out of a total of about 4000 points evaluated in each run, around a half were non-dominated. Of these, from 10 to 200 points were satisficing. Runs in which the population could locate itself earlier in the satisficing region rapidly produced variations of the first satisficing individuals found, which explains why some runs were apparently much more successful than others.

In this example, some degree of preference information proves to be absolutely necessary to guide the GA through a very large non-dominated set towards solutions of practical interest. Figure 5.16 illustrates the solutions so-far-preferable at three different stages of an algorithm run. The preferable solutions found at the end of the run reveal trade-offs between the several objectives within the bounds imposed by the goals associated with them.

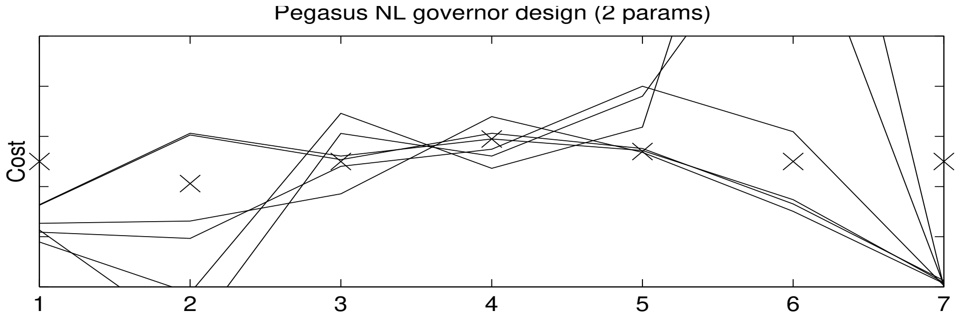
In these plots, objectives are represented by their integer index i on the x-axis, and performance in each objective dimension is plotted on the y-axis. As explained in Chapter 3, each candidate solution $\mathbf{x}$ is represented by a line connecting the points $(i, f_i^*(\mathbf{x}))$. A trade-off between adjacent objectives results in the crossing of lines between them, whereas non-concurrent lines indicate that those objectives are non-competing. For example, rise-time (objective 4) and settling-time (objective 5) do not seem to compete heavily. In turn, there is a clear trade-off between these two objectives and objective 6 (overshoot). The gain and phase margins (objectives 2 and 3) seem to be competing objectives in part of the trade-off only.

5.4.5 Full controller parameter optimization

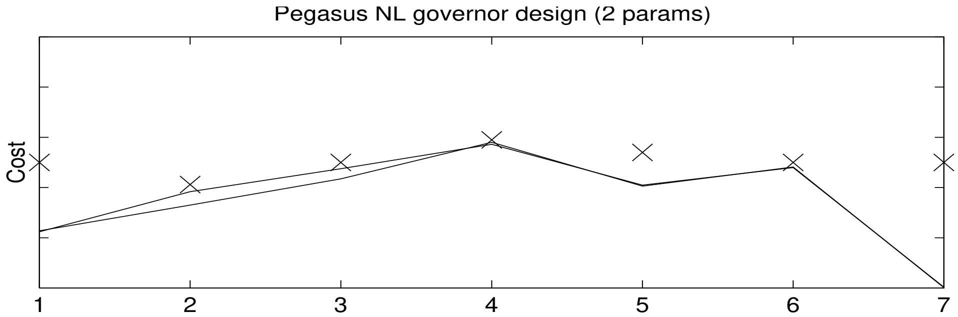
The optimization of the full controller assumes no a priori knowledge of the controller parameter values apart from that implicitly used in setting the parameter ranges. Individuals are now 120-bit long, and the population is increased to 150 individuals.

Running the GA reveals a much larger set of satisficing solutions. The preferable individuals found up to and including the 40th generation are shown in Figure 5.17(a). When compared with the previous example, the solutions found have all greater maximum pole-magnitudes, that is, longer time-constants. On the other hand, they generally exhibit better stability margins and settling time. Thus, not only do the solutions found in this second example not dominate those found previously, but also they represent a significantly different region of the preferable set.

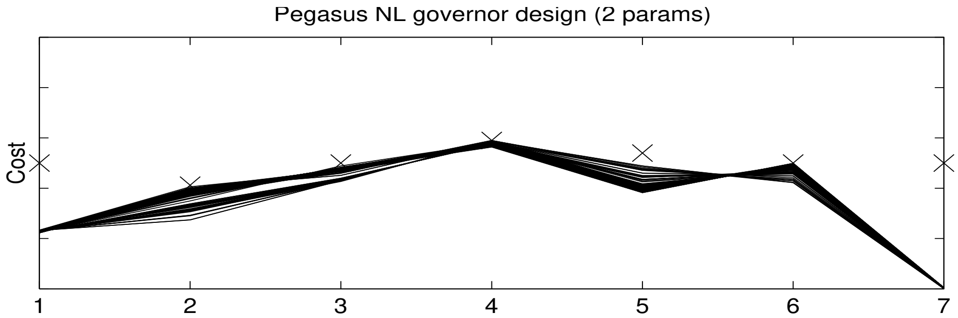
The fact that the 8-variable GA could not find solutions similar to, or better than, those found for the 2-variable example shows how sampling the whole



After 10 generations.



After 30 generations.



After 50 generations.

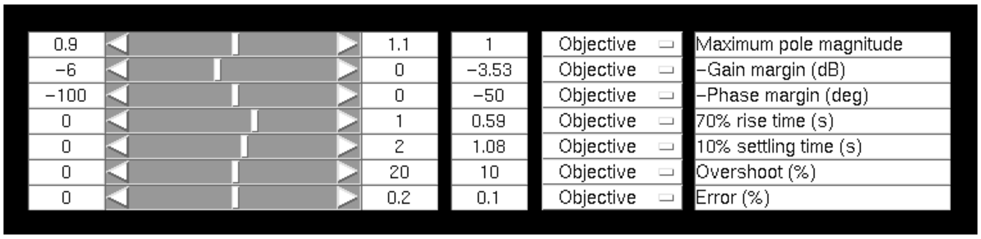
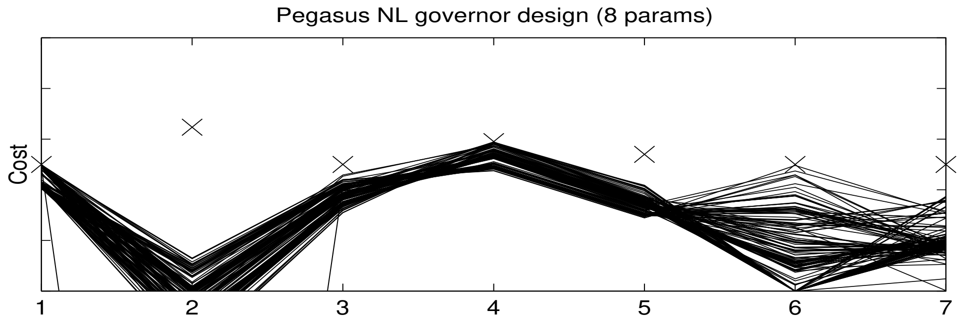
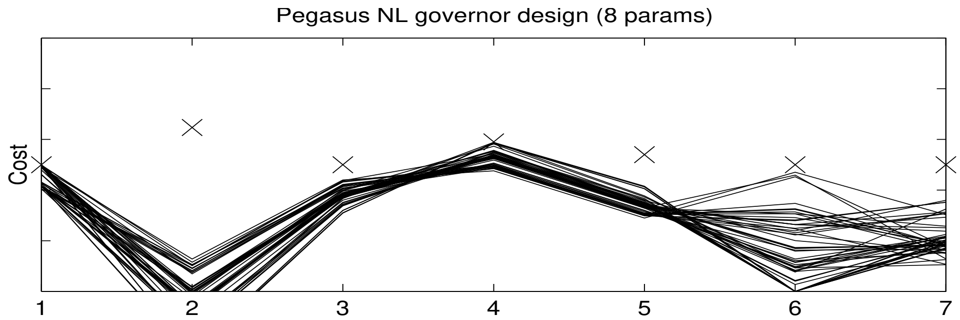


Figure 5.16: Sample trade-off graphs from a run on the 2-variable example, showing the preferable solutions found after 10, 30 and 50 generations. The vertical axis ranges are the same as those associated with the control-panel sliders. The goal values are given in the middle column of the panel and are plotted as “×”.



(a) Cumulative trade-off graph after 40 generations.



(b) Preferable individuals in the population at generation 40.

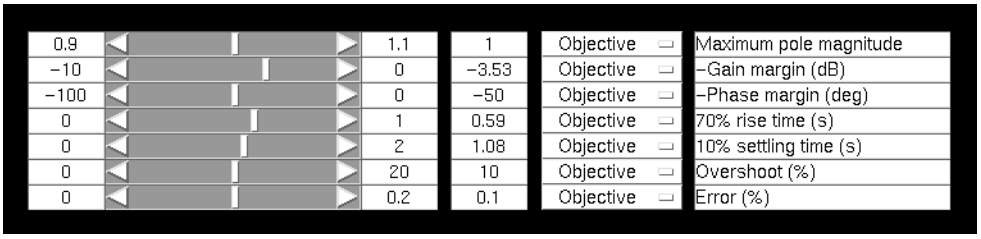


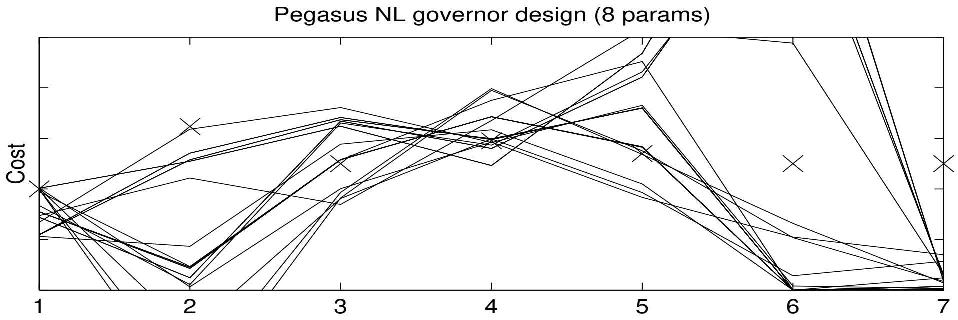
Figure 5.17: Sample trade-off graphs from a run on the 8-variable example, showing preferable solutions found after 40 generations. The vertical axis ranges are the same as those associated with the control-panel sliders. The goal values are given in the middle column of the panel and are plotted as a “×”.

preferred region with a population of a given size may not be always feasible, even though niche induction mechanisms are used. The diversity among the preferable individuals in the population at generation 40, shown by the corresponding trade-off graph in Figure 5.17(b), is an indication that these techniques are indeed at work, at least to some extent. Nevertheless, their effectiveness in causing the population to spread across the whole, actual preferred set also depends on the genetic algorithm's own ability to capture the regularities in the fitness landscape, and generate offspring accordingly. Unfortunately, as discussed in subsection 4.4.2, binary coded GAs cannot be expected to be able to deal well with strong variable dependencies.

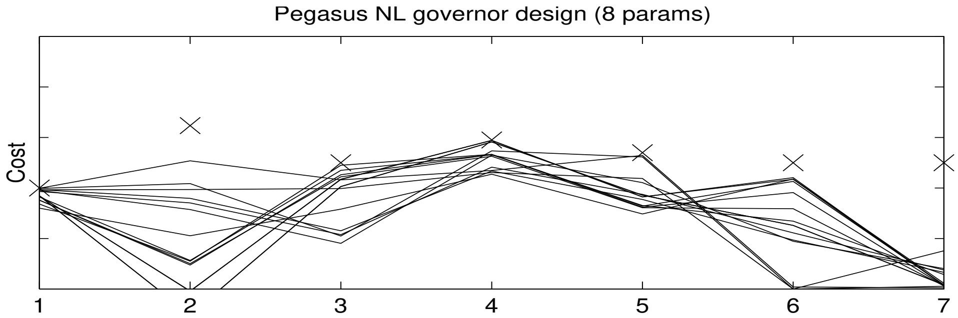
Appropriate interaction with the decision maker can help address this difficulty. The algorithm has found solutions which do, in principle, meet their goals, and which it sees all as equally fit. Other solutions non-preferable to these may exist, but they are probably more difficult to find given the encoding structure used for the chromosomes and the associated genetic operators. If the DM finds the candidate solutions produced by the GA unacceptable, then preferences should be refined so as to stimulate the GA to move on to a different region of the non-dominated set.

As an example, consider again the trade-off graph in Figure 5.17(a). The step-response error at the end of the simulations is not particularly low, reflecting the long time constants associated with maximum pole magnitudes close to unity. If faster responses are desired, the DM can tighten the goals associated with either, or both, of these two objectives while continuing to run the algorithm.

Suppose the DM chooses to lower the goal associated with the maximum pole magnitude to 0.98. Solutions capable of meeting this new requirement turn out not to have yet been found, and a set of relatively non-dominated solutions close to being satisficing is presented to the DM (Figure 5.18(a)). If the DM *believes* this new set of goals can in fact be met (for example, because some goals in the previous set have clearly been over-attained), there is no algorithmic reason



(a) After 40 generations (new preferences)



(b) After 60 generations

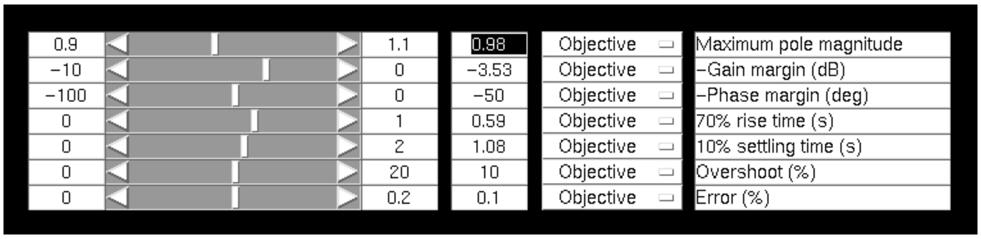


Figure 5.18: Sample trade-off graphs showing the effect of tightening one goal's value. The population is driven towards achieving that goal at the expense of degradation with respect to others.

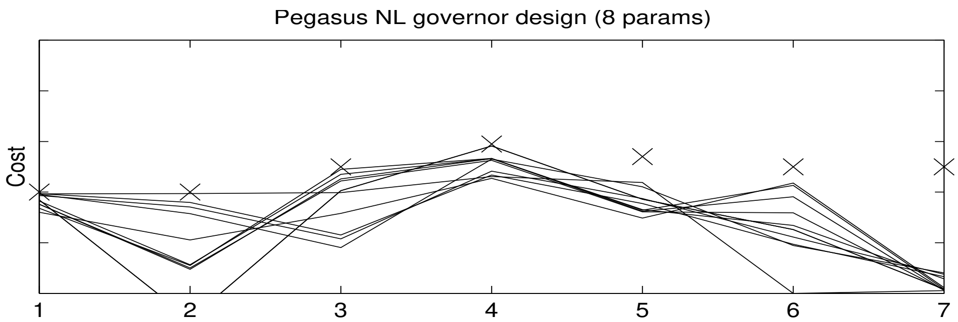
not to specify it. If the GA cannot find satisficing solutions, it will at least produce a set of relatively non-dominated approximations to the new goals on which further preference changes can be based. As the population continues to evolve, reductions in maximum pole magnitude are accompanied by reductions in the step-response error measured at the end of the simulation, and satisficing solutions are eventually found at the expense of some degradation in gain margin and other objectives (Figure 5.18(b)).

Further refinements could involve specifying hard, instead of soft, bounds for some of the objectives. Suppose the DM decided that a gain margin of 6dB seemed possible (in the presence of the trade-off graph), and that better gain and phase margins would not be required. Raising the priority of these objectives, and changing the gain margin goal, converts them into hard constraints as desired. Doing so reduces the preferred set further (Figure 5.19(a)), and the process continues. The family of step responses corresponding to the trade-off graph in Figure 5.19(b) is shown in Figure 5.20.

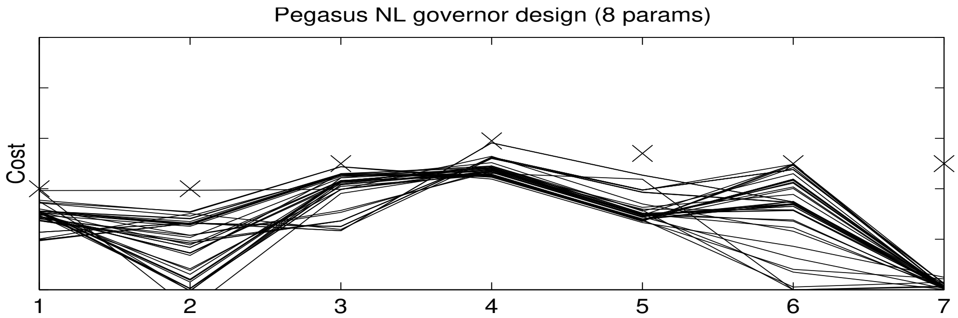
5.5 Discussion

In the search for a reduced, yet illustrative, set of realistic design objectives for the Pegasus engine, the ability of the GA to exploit omissions in the design requirements soon became apparent. The set up of the closed-loop maximum pole magnitude as a soft objective instead of a hard one was a direct consequence of the solutions produced by early runs of the algorithm being clearly unacceptable: the GA was sometimes able to exploit the non-linearity of the plant in such a way that the step-response characteristics achieved were only valid for the particular amplitude of the step used during simulation. Applying slightly larger or smaller steps to the controlled plant immediately revealed the slow dynamics in the closed loop, deeming the controller unacceptable.

The objective functions and respective goals used in the examples given reflect



(a) After 60 generations (hard stability margins)



(b) After 80 generations

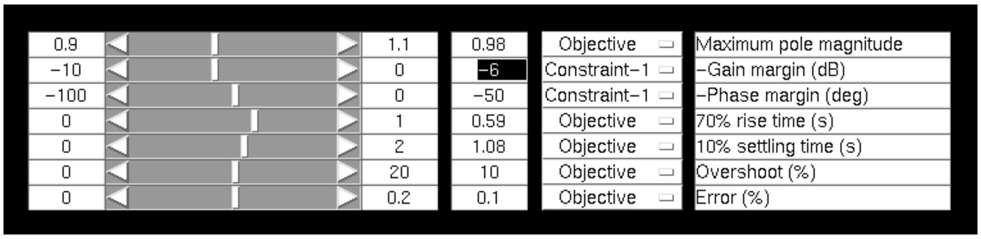


Figure 5.19: Sample trade-off graphs showing the effect of converting two objectives into hard constraints, while also changing the goal associated with of one of them.

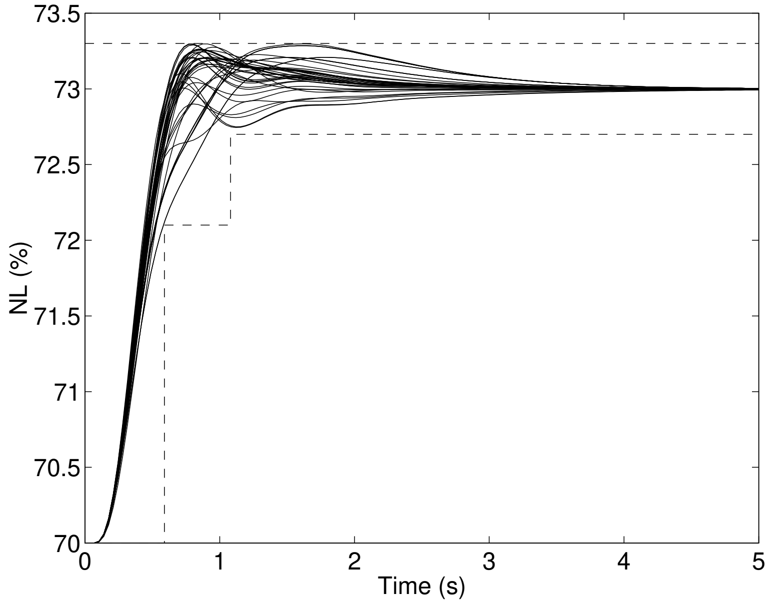


Figure 5.20: Family of preferable step-responses found after 80 generations.

design concerns in a direct form. The problems were formulated from the point of view of the designer, not that of the optimizer. The associated cost landscapes do not all meet common requirements such as continuity, smoothness, and absence of flat regions, and neither necessarily do the resulting multiobjective cost landscapes. The practical usefulness of the multiobjective formulation used is, therefore, tied to the ability to search and optimize cost surfaces in a class much broader than usual, as provided by the GA.

In the original work of Hancock (1992) on the Pegasus model, the same designer-oriented approach to controller optimization was adopted via the goal-attainment method, as implemented in the Optimization Toolbox for MATLAB (Grace, 1990). A sequential approach to optimization was proposed which he called *design by evolution*. Given a starting point, optimization was performed only on, say, two of the decision variables, while the others remained fixed. This provided the starting point for the next stage, which now included another variable. The process continued until all decision variables took part in the optimization.

The need for such a sequential approach clearly illustrates the shortcomings of gradient-based optimizers when applied directly to problems involving less than well-behaved objective functions. On the other hand, and since the notion of a "well-behaved" function is dependent on the optimizer, a better understanding of what classes of functions are suited to which genetic optimizers is still needed.

The Pegasus case-study is also a good example of a problem where pure Pareto-optimization would be impractical due to the size of the Pareto-optimal set. Specifying realistic goals for the objectives dramatically reduced the size of the region of the search space to be sampled accurately by the GA. This is certainly true for the two-variable example, and it should apply to an even greater extent for the full, eight-variable problem. In this last problem, even the original preferred set appeared to be too large for the GA. Tightening some of the goals during the GA run rapidly caused the population to move on towards more promising solutions.

On the other hand, when objective functions are expensive to evaluate, the interactive use of the MOGA does become less attractive, due to the long time spent at the function evaluation stage of the GA. Again, this was especially true for the Pegasus example. For even more time-consuming problems, it should be possible to address this difficulty by implementing the GA on a parallel architecture.

5.6 Conclusion

The application of a multiobjective GA based on the concept of preferability to the optimization of controller parameters for two different plants has been described. The problems involved a mixture of soft and hard objectives, some defined in the frequency domain, and some defined in the time domain. In both case-studies, a priori preference information was supplied to the GA by assigning goals and priorities to the objectives. These preferences were later refined according to the trade-off information produced as the GA ran.

The quality of the approximate trade-off surfaces produced by the MOGA will generally depend on the combined effect of objective functions, genetic representation, and operators used, and thus cannot be guaranteed. However, and despite its stochastic nature, the MOGA tended to perform more or less consistently when run several times on the same problem, especially in the long run. This has been explicitly shown for the first example presented which, involving only two objectives, was simpler to visualize. For more computationally intensive problems, such as the Pegasus example, this kind of study would require large amounts of computing effort. Again, parallel processing would be an attractive option, especially if combined with a more efficient implementation of the model than what can be achieved with SIMULINK.

The next Chapter reports on the application of the MOGA to a combinatoric problem derived from non-linear system identification.

Chapter 6

Nonlinear System Identification

6.1 Introduction

The identification of single-input single-output (SISO) systems from a set of, possibly noisy, input-output data (the training data set) is a classical problem in control engineering. If the system to be identified is known to be linear or approximately linear, then the problem reduces to determining the order and estimating the parameters of a transfer function which, in some sense and to the necessary extent, approximately describes the input-output relation of the system. The general, non-linear case is potentially much more complex.

System identification is traditionally performed in two stages. In a first stage, a model family (e.g., linear up to a given order, polynomial, etc.) is chosen a priori, and a candidate model is produced which minimizes some error criterion, here referred to as the *identification criterion*, over that family. Least-squares estimation of the coefficients of a linear model, for example, corresponds to minimizing the mean square one-step-ahead prediction error of that model on the training set, over the space of possible model parameters. In a second stage, the identified model is validated by verifying that it satisfies any assumptions made prior to identification, such as noise independence, for example, and that it can explain new data as well.

The choice of the identification criterion usually depends on a combination of the purpose which the identified model should serve and of how easily such a criterion can be optimized over the space of models on which it is defined. The traditional preference for least-squares methods, for example, is certainly due in part to their computational efficiency. Less tractable objectives are typically accommodated at the validation stage, in the hope that the identified model satisfies them. When this is not the case, alternative models must be produced by optimizing the identification criterion over a different model family (linear with a different order, involving different polynomial terms, etc.).

Evolutionary approaches to system identification have made the choice of the identification criterion more flexible. Measures of long-term prediction error, for example, have been used in combination with GAs by Kristinsson and Dumont (1992) for on-line parameter identification, as an alternative to recursive least-squares methods. Additionally, genetic programming techniques make it possible to search very broad classes of non-linear models (Oakley, 1994).

Multiobjective genetic algorithms enhance the appeal of evolutionary approaches to system identification further, by allowing validation criteria to influence the model production stage, and bias it towards models more likely to be valid. In this Chapter, the evolution of non-linear, polynomial models for a pilot scale liquid-level system will be considered. After formulating the identification problem as a subset selection problem in Section 6.2, a genetic approach to the evolution of subsets is reviewed in Section 6.3, including representation and operators. Experimental results for both a single-objective GA and a multiobjective GA are presented and discussed in Section 6.4.

(Part of this work was carried out in collaboration with a fellow PhD student, Eduardo Mendes, and was published as Fonseca *et al.* (1993).)

6.2 The identification problem

The NARMAX (Non-linear Auto-Regressive Moving Average model with eXogenous inputs) model (Leontaritis and Billings, 1987) can represent, and has been used to identify, a wide class of non-linear systems. For single-input single-output systems, the NARMAX model is:

$$\begin{aligned} y(t) \;=\; & \mathcal{F}(y(t-1), \ldots, y(t-n_y), u(t-1), \ldots, u(t-n_u), \\ & \qquad e(t-1), \ldots, e(t-n_e)) + e(t)\,, \end{aligned} \tag{6.1}$$

where $y(t)$, $u(t)$ and $e(t)$ represent output, input and noise at time t, respectively. The positive integers n_y, n_u and n_e are the corresponding maximum lags, and $\{e(t)\}$ is a zero mean independent sequence. $\mathcal{F}$ is some non-linear function, the form of which is usually unknown.

Because $\mathcal{F}$ can assume a variety of forms, the identification of NARMAX models is a much more difficult task than its linear counter-part, where the major difficulty is usually to determine the system order. In practice, the non-linear function $\mathcal{F}$ can be approximated, for example, by a polynomial expansion (Chen and Billings, 1989) of a given degree ℓ. Doing so, one obtains the representation

$$\begin{aligned} y(t) \;=\; & \theta_0 + \sum_{i_1=1}^{n} \theta_{i_1}\, x_{i_1}(t) + \sum_{i_1=1}^{n} \sum_{i_2=i_1}^{n} \theta_{i_1 i_2}\, x_{i_1}(t) x_{i_2}(t) + \cdots \\ & \cdots + \sum_{i_1=1}^{n} \cdots \sum_{i_\ell=i_{\ell-1}}^{n} \theta_{i_1 \ldots i_\ell}\, x_{i_1}(t) \cdots x_{i_\ell}(t) + e(t)\,, \end{aligned} \tag{6.2}$$

where $n = n_y + n_u + n_e$, all θ represent scalar coefficients, and all $x(t)$ represent lagged terms in y, u or e.

Equation (6.2) clearly belongs to the linear regression model

$$y(t) = \sum_{i=1}^{M} \theta_i\, p_i(t) + \xi(t), \qquad t = 1, \ldots, N\,, \tag{6.3}$$

where N is the data length, $p_i(t)$ are monomials of $x_1(t)$ to $x_n(t)$ up to degree ℓ and M is the number of distinct such monomials. $p_1(t) \equiv 1$ corresponds to a constant term, $\xi(t)$ is some modelling (residual) error and θ_i are unknown parameters to be estimated. Equation (6.3) can also be written in the matrix form

$$\begin{aligned} \mathbf{y} &= [\mathbf{p}_1 \; \dots \; \mathbf{p}_M]\, \boldsymbol{\theta} + \boldsymbol{\xi} \\ &= \mathbf{P}\, \boldsymbol{\theta} + \boldsymbol{\xi}\,. \end{aligned} \tag{6.4}$$

The parameter vector $\boldsymbol{\theta}$ is usually found by minimizing the Euclidean norm $\|\mathbf{y} - \mathbf{P}\boldsymbol{\theta}\|_2$, that is, by solving a Least Squares Problem. In Golub and Loan (1989) several methods for solving such a problem are given.

6.2.1 Term selection

The explosive growth of the number of possible monomial terms with the degree of non-linearity and the order (or maximum lag) assumed for the model implies that even relatively small values of degree (say $\ell \leq 3$) and lag (say $n_u, n_y \leq 5$ and $n_e = 0$) would result in a model too complex to be useful in practice ($\binom{5+5+3}{3} =$ 286 terms). More importantly, least-squares estimation of so many parameters would require impractically large amounts of data. Therefore, simpler models are usually considered which include only a reduced number of non-linear terms with up to a given order and degree. Although this reduces the number of coefficients to be estimated, it introduces a new problem: selecting the appropriate non-linear terms.

A current, heuristic approach to term selection (Chen *et al.*, 1989) is based on orthogonal least squares regression. The first term to be selected is that which leads to a single-term model with the lowest residual variance. The remaining terms are then successively incorporated according to their contribution to improvement in the model, until the residual variance ceases to decrease significantly as more terms are added. This approach has the important advantage of being computationally efficient, while giving some indication of the contribution of the various terms to the identified model. However, models obtained in this (sequential) way do not necessarily exhibit the lowest residual variance which could be achieved with models of similar complexity. A true optimization approach to term selection has the potential to produce simpler models.

6.3 Genetic algorithms and term selection

The search for a suitable model structure, as described above, can be seen as a subset selection problem. Given a parent set of distinct non-linear terms, a model composed of fewer terms is sought which, following parameter estimation, minimizes a given criterion.

In the following, the choice of a suitable genetic representation and associated operators is discussed.

6.3.1 Representation

Terms such as $u(t-1)y(t-4)$ and $u(t-2)y(t-4)$ may, at first glance, appear to be similar. However, their contribution to the final model can be drastically different, e.g., in the (common) case where the input signal u is uncorrelated. This suggests that different non-linear terms should be encoded in a form which implies no similarity between them. Seeing models as subsets of terms is consistent with this, and with the fact that the order of the terms in the model is irrelevant.

In an application of GAs to the knapsack problem (also a subset selection problem), Goldberg and Smith (1987) encoded subsets as binary strings of length equal to the number of elements in the parent set. In each string, a '1' indicated the presence of the corresponding element in the subset, and a '0' its absence.

index	term
1	1
2	$y(t-1)$
3	$y(t-2)$
$\vdots$	$\vdots$
i	$p_i(t)$
$\vdots$	$\vdots$
N	$u(t-n_u)^\ell$

$$\begin{array}{|c|c|c|c|}\hline i_1 & i_2 & \dots & i_k \\ \hline \end{array}$$

$$\downarrow$$

$$\begin{array}{|c|c|c|c|}\hline p_{i_1}(t) & p_{i_2}(t) & \dots & p_{i_k}(t) \\ \hline \end{array}$$

Figure 6.1: Chromosome coding: Models are represented as subsets with k elements.

When the parent set is very large, as may be the case in term selection problems, this leads to very large strings, even if only "small" subsets are known to be of interest.

An alternative representation for subsets, proposed by Lucasius and Kateman (1992), is better suited to searching for subsets of small size. The parent set is laid out in a look-up table format, so that elements (in this case, model terms) can be referred to by their (arbitrary) position in such a table (see Figure 6.1). Subset individuals, or candidate models, can thus be represented as strings composed of a number of all-distinct integers.

Although this is not a requirement of this representation, a fixed subset size will be used in the examples that follow, for simplicity.

6.3.2 Genetic operators

Genetic operators for the term selection problem should guarantee the generation of valid models at all times, preferably without the introduction of unnecessary redundancy. Redundancy exists when there are several *different* representations for the same individual. The operators used in this work satisfy these two requisites and are described next.

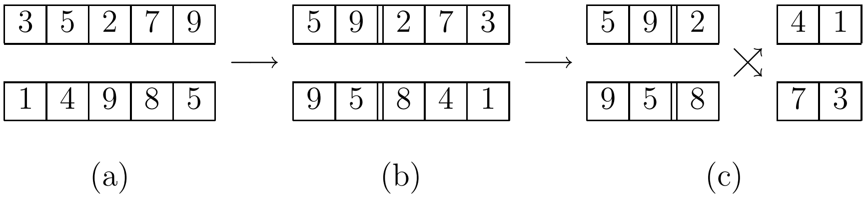


Figure 6.2: Subset crossover: (a) Parents. (b) Keep the common elements and shuffle the remaining. (c) Cross.

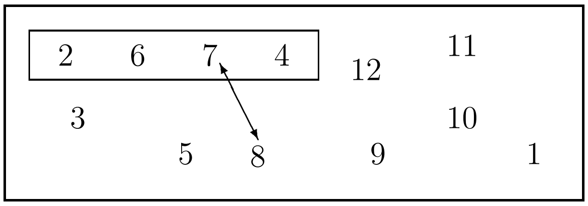


Figure 6.3: Trade mutation.

6.3.2.1 Full identity-preserving subset crossover

This crossover operator is a variant of the recombination operators proposed by Lucasius and Kateman (1992). Offspring subsets are produced from two parent subsets so that their intersection is preserved. The remaining elements in each parent are shuffled and exchanged to produce two offspring, so that the identity of all elements is preserved, i.e., no genetic information is lost in the process, and the order in which terms appear in each parent is ignored. In this way, order redundancy is consistently eliminated from the representation (see Figure 6.2).

6.3.2.2 Trade mutation

The trade mutation operator tests each element of the subset in turn, and replaces it with a randomly selected element of the current complement of that subset, with given probability, in an import-export fashion (see Figure 6.3). Since the complementary subset reflects the contents of the individual at all times, the operator is still defined in the case of subsets with more than half of the elements of the parent set.

6.4 Experimental results

A genetic algorithm was used to study data generated by a large pilot scale liquid-level system excited by a zero mean Gaussian input signal (Voon, 1984). From a total of 1000 input-output pairs, the first 500 points were used in the identification, while the remaining 500 points were divided into two sets and used for validation.

6.4.1 First runs

A first series of runs was aimed at verifying the ability of a GA based on the above representation and operators to minimize residual variance over a space of subset models. Models were assumed to have 7 terms of degree up to 3 and maximum lags $n_y = n_u = 2$. Note that, although the number of elements in the full model set is only 35, the number of possible 7-term models amounts to $\binom{35}{7} = 6,724,520$.

A genetic algorithm with population size 50 was run for 100 generations, using residual variance as the single objective. This GA was essentially a particular (single-objective) case of that used in the previous Chapter, except for the representation and genetic operators, which were as described above. Also, sharing and mating restriction were disabled.

Figure 6.4 shows how the residual variance (VAR) of the best model of each run tended to evolve as the number of generations increased. The median (thick solid line), the first and third quartiles (thin solid lines), and the maximum and minimum (grey lines) of 13 runs are shown. Despite different initial conditions, all runs were able to produce near minimum-residual variance models after around

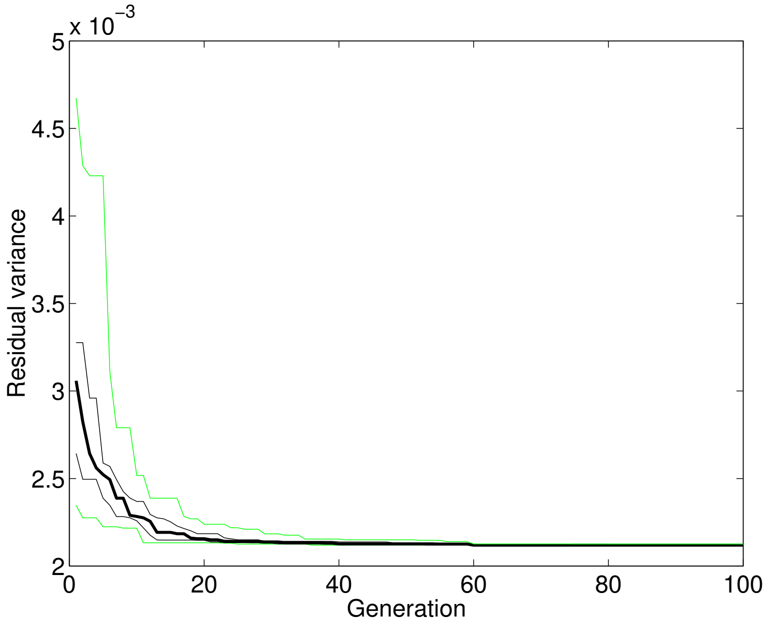


Figure 6.4: Evolution of low-residual variance models.

60 generations.

The model with the lowest residual variance was found in all but one of the 13 runs. It can be written as

$$\begin{aligned} y(t) &= a_1 y(t-1) + a_2 u(t-1) + a_3 y(t-2) + a_4 u(t-2) + \\ &\quad + a_5 y(t-1)u(t-1) + a_6 y(t-2)^2 u(t-2) + a_7 u(t-1)u(t-2)^2 \,. \end{aligned}$$

In order to (simplistically) validate this model, its mean-square long-term prediction error (LTPE) was measured on the same set of data used to estimate the coefficients. Also, the mean-square one-step-ahead prediction error (OSAPE) and the LTPE were computed for the two validation sets. The results are given in Table 6.1. The corresponding long-term prediction graphs are presented in Figure 6.5. It can be seen that the model does approximate the measured output of the system to a reasonable extent, although not so well in some particular data

Table 6.1: Minimum-residual variance model.

Estimation data (500 points)		New data 1 (250 points)		New data 2 (251 points)	
VAR	LTPE	OSAPE	LTPE	OSAPE	LTPE
2.119	53.78	2.012	13.71	2.815	39.67
$\times 10^{-3}$					

segments.

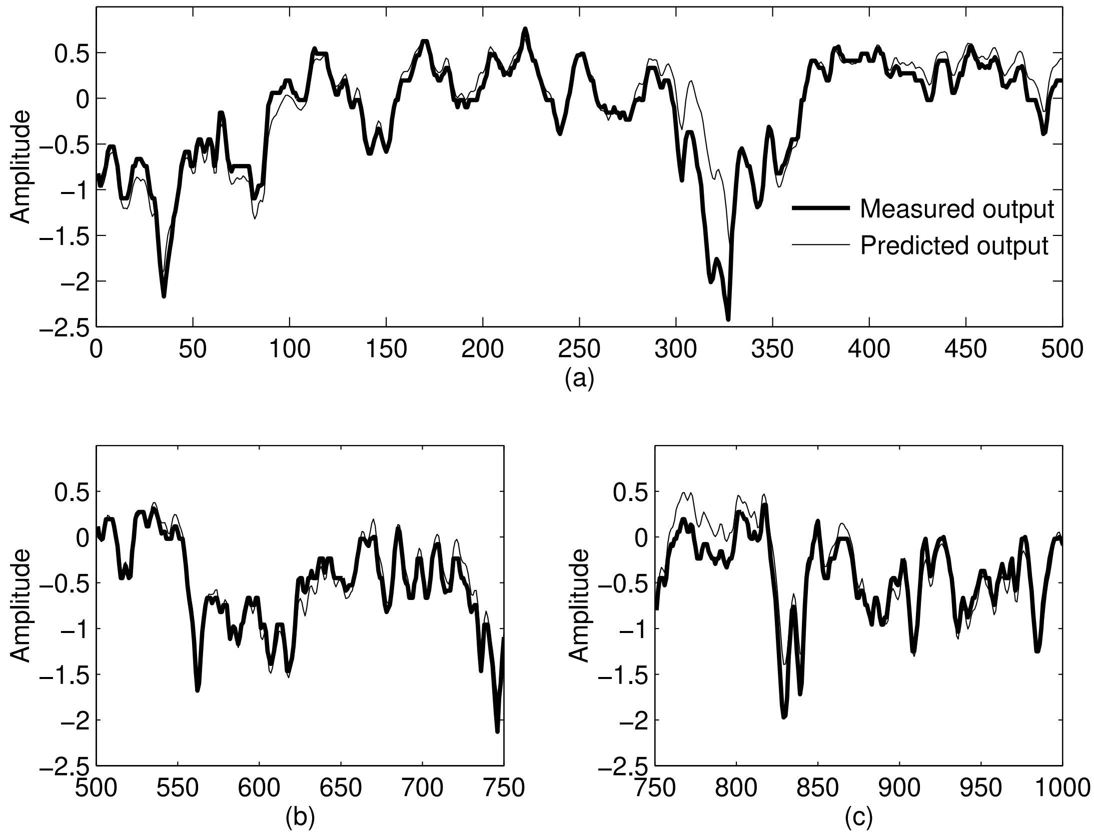
Since many other models were actually generated and evaluated by the GA, it would be possible to study those as well. However, if the ability of the model to predict in the long term is in fact relevant, long-term prediction should really become another identification objective.

6.4.2 Multiobjective identification

In a second series of runs, a multiobjective genetic algorithm included both residual variance and mean-square long-term prediction error as objectives, both relative to the training set. Goals of zero were set for both objectives, actually implementing a pure Pareto GA. Sharing and mating restriction were performed in the objective domain. Once more, 13 runs of the GA were performed, this time for up to 200 generations.

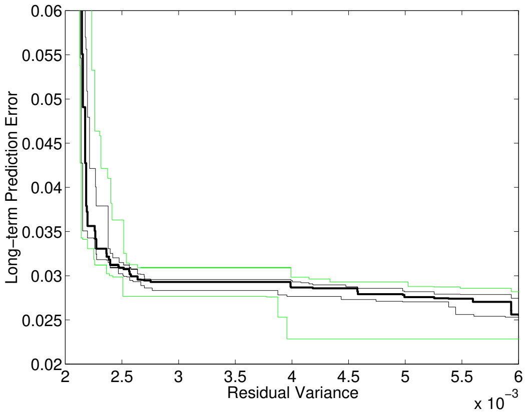
Results are presented in Figure 6.6. The plots, produced from the sets of non-dominated individuals found up to and including generations 50, 100, 150 and 200, show how the GA evolved models in the various regions of the objective space. As before, thick solid lines indicate performance levels which, individually, were attained in 50% of the runs, while the upper and lower thin lines correspond to performance levels attained in 75% and 25% of the cases, respectively. The grey lines correspond to performance levels attained only in a single run (lower), or in all runs (upper).

These results also illustrate, and quantify, the fact that long-term prediction

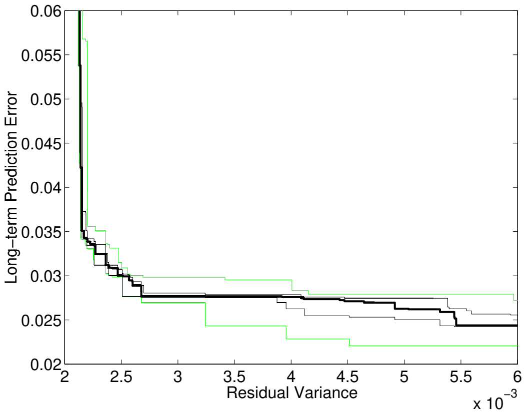


$$\begin{aligned}\hat{y}(t) &= a_1\hat{y}(t-1) + a_2u(t-1) + a_3\hat{y}(t-2) + a_4u(t-2) + \\ &\quad + a_5\hat{y}(t-1)u(t-1) + a_6\hat{y}(t-2)^2u(t-2) + a_7u(t-1)u(t-2)^2\end{aligned}$$

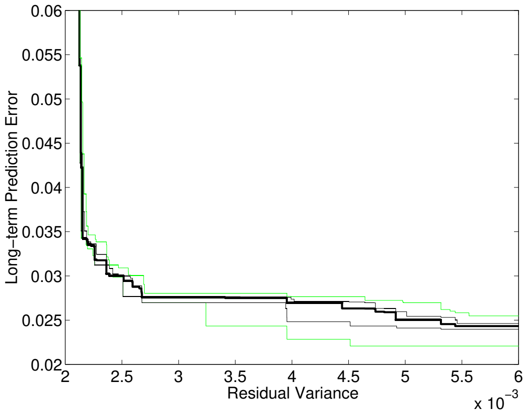
Figure 6.5: Minimum-residual variance model predicted output. (a) Estimation set. (b) New data 1. (c) New data 2.



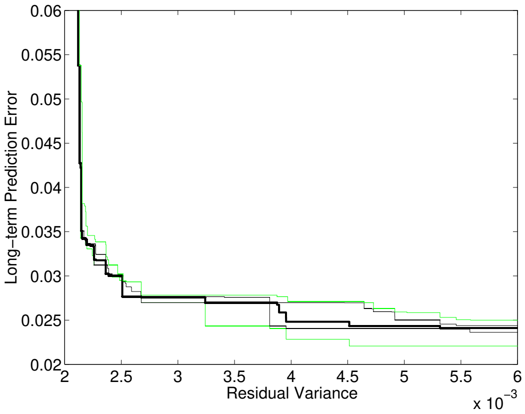
After 50 generations



After 100 generations



After 150 generations



After 200 generations

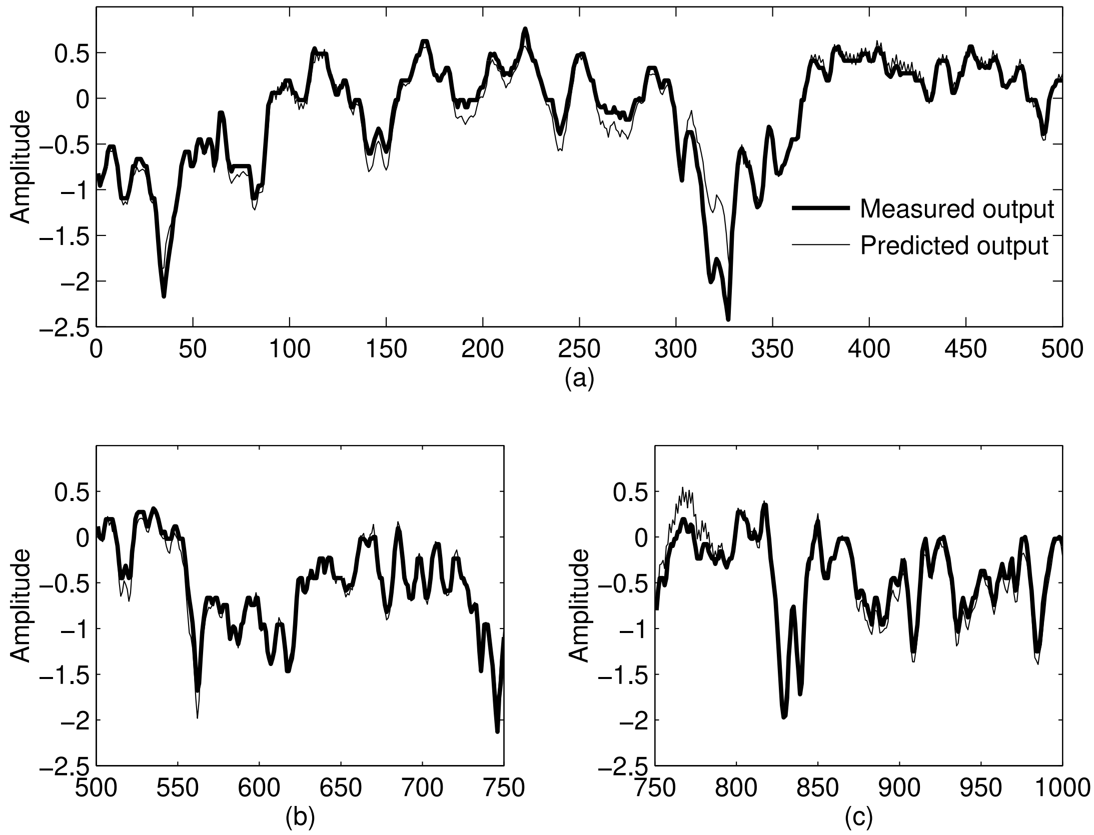
Figure 6.6: Evolution of low-residual variance and low long-term prediction error models (13 runs).

Table 6.2: Performance of some candidate models

	Estimation data (500 points)		New data 1 (250 points)		New data 2 (251 points)	
Model no.	VAR	LTPE	OSAPE	LTPE	OSAPE	LTPE
1	2.119	53.78	2.012	13.71	2.815	39.67
2	2.252	31.82	1.988	8.27	2.763	32.38
3	3.242	24.33	2.264	9.26	4.469	21.83
4	4.514	22.07	3.254	22.57	6.749	40.01
	$\times 10^{-3}$					

error can usually be significantly reduced by allowing some (minimal) degradation in residual variance. Perhaps more interesting is the distinctive knee of the trade-off curve, which indicates a clear conflict between the ability of models in the class considered to predict the next step (as indicated by the residual variance), and their long-term prediction performance. Most surprising of all, however, is to see that long-term prediction can apparently be improved even further at the expense of a considerably poorer one-step-ahead prediction performance. Another interesting detail is that the best long-term predictors seemed to consistently exclude the term $y(t-1)$, which could justify worse one-step-ahead prediction.

Table 6.2 presents validation results for four models selected from those evaluated in the various runs of the MOGA. Model 1 is the same as the minimum residual variance model presented earlier, while model 4 is the model with the lowest LTPE (on the training set) found by the GA. The two intermediate models can be seen to generally perform better on new data than the extreme ones: model 3 tends to predict better than model 2 in the long-term, but model 2 exhibits better one-step-ahead prediction. Long-term prediction graphs for model 3 are given in Figure 6.7. Note how the predicted output, although generally closer to the measured output than in the case of model 1, is locally oscillatory. This behaviour explains the model's higher one-step-ahead prediction error.



$$
\begin{aligned}
\hat{y}(t) &= a_1 + a_2 u(t-1) + a_3 \hat{y}(t-2) + a_4 + \hat{y}(t-1)u(t-1) \\
&\quad + a_5 \hat{y}(t-2)^2 + a_6 \hat{y}(t-1)^3 + a_7 u(t-2)\hat{y}(t-2)^2
\end{aligned}
$$

Figure 6.7: Model predicted output for model 3. (a) Estimation set. (b) New data 1. (c) New data 2.

Given the clear trade-off between the two identification objectives, it may be the case that the model class considered cannot simultaneously capture all aspects of the dynamics of the system. Therefore, the choice of a final model will be essentially subjective, and dependent on the purpose which the model should serve.

6.5 Conclusion

A genetic algorithm for term selection was described, and applied in the identification of a pilot-scale, but real system. Due to being a generate-and-test approach, the genetic algorithm produces not only one, but a family of low-variance models, which can be assessed according to different criteria before the final model is chosen. The search, being carried out in model space rather than term space, has the potential for being more effective than other methods available. In addition, it can easily accommodate multiple identification objectives via multiobjective selection, as proposed in Chapter 4. It is therefore a much more general approach than orthogonal least-squares methods, although also more computationally intensive.

The subset GA, as proposed here, will only generate models with a fixed number of terms. It should not be too difficult to modify the operators to accommodate all models with a number of terms up to a given limit. The number of terms of each model could then be made into an additional objective to be minimized. Any other quantities which either measure model complexity such as the maximum lag, or improve the likelihood of obtaining a valid model, e.g., correlation measures on the residuals, could also be incorporated as objectives, and optimized. However, validation would still require data unknown to the GA.

A multiobjective approach to system identification such as the one presented here, in addition to generating not one, but a family of models from which the final model can be chosen, may also produce valuable insight into the quality of the data, the appropriateness of the model space chosen, and possibly also the complexity required for the model.

Chapter 7

Conclusions

7.1 Genetic algorithms

Genetic algorithms, and other evolutionary techniques, have considerably broadened the scope of optimization in engineering. The interest and effort devoted to GAs by the control engineering community has been representative of that of the engineering community at large, which has culminated recently in the realization of a well-attended International Conference dedicated to Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA, 1995). Consisting essentially of structured, albeit stochastic, generate-and-test approaches, GAs find application in many areas which other optimizers do not reach. In particular, problems involving any combination of numeric (real, integer, etc.) and non-numeric decision variables (permutations, subsets, graphs, and others) can be handled, as long as the search space can be represented and *suitable* genetic operators can be defined over that space.

Since it remains unclear how to design suitable genetic operators for given problems, or classes of problems, what has become known as the "representation problem" can be seen as a major limitation of GAs. In engineering, however, there is usually a great deal of domain knowledge which can be successfully embedded in the GA at this level, although it may require some experimentation.

Applications in control engineering range from standard parameter tuning, usually to side-step difficulties imposed by discontinuities, non-smoothness and other unfavourable properties of the cost surface of the problem, to the optimization of controller structures, neural-network topologies, sensor and actuator placement, and path-planning, among others. Both on-line and off-line applications typically rely on the ability to model the system under study sufficiently accurately, or, at least, on some means of ensuring that a particular individual evaluation (on-line) will not have catastrophic consequences.

Another important class of problems which can be addressed with GAs is that where the quality of a solution cannot be reduced to a single numerical value, but rather, can only be obtained by comparison with another solution in terms of "better", "comparable", or "worse". This is because GAs only require that a (broad) ranking of the population be performed at each generation. The evolution of game-playing programs, for example, falls in this category, and so does Pareto optimization.

7.2 Multiobjective evolutionary optimization

Research on multiobjective optimization with evolutionary algorithms has been prompted mainly by the clear potential of a population-based search to concurrently evolve and maintain multiple non-dominated solutions, so that design decisions may be delayed until sufficient trade-off information is available. Since the quality of a solution must be represented by a scalar fitness value, selection has been the main aspect of EAs to receive attention. Although several multiobjective selection strategies have been proposed, more emphasis has been put on detailing the selection procedure itself than on analyzing its impact on the cost surface actually searched by the EA. As discussed in Chapter 4, some of these approaches are actually, although implicitly, aggregating approaches. Only some of the more recent approaches are directly based on the concept of Pareto-optimality.

In order to unify the various approaches discussed, multiobjective selection was described more generally as a decision-making exercise involving the individuals in the EA population. This made it clear that the problem of multiobjective "selection" (in fact, utility or cost assignment) is neither unique to EAs nor new. The handling of multiple objectives with EAs given goal and priority information could thus be addressed from a decision-making perspective, independently from the genetic algorithm used later. This conferred on the approach a reasonable degree of generality, facilitating both the development of the actual strategy (by addressing pairwise comparison first and only then population ranking) and its characterization and interpretation. The result was a cost-assignment strategy encompassing many common multiobjective and constrained formulations, where the cost assigned to candidate solutions can be interpreted as an estimate of the proportion of the search space which improves on those solutions, and thus, related to the original problem.

The subsequent development of a multiobjective genetic algorithm (MOGA) focused on three main aspects. Firstly, a simple extension of traditional rank-based fitness assignment was introduced in order to guarantee that equal fitness was assigned to all individuals with the same cost. Then, sharing and mating restriction were formulated in the objective domain in order to promote and maintain diversity in the population, while allowing the appropriate niche sizes to be estimated from the data accumulated during the GA run. Finally, the introduction of a small percentage of random immigrants into the population to improve responsiveness to on-line preference changes completed the modifications made to an otherwise typical generational genetic algorithm.

7.3 Applications in control

Multiple objectives and constraints arise continuously in control and systems engineering. In some cases, objectives are sufficiently well-behaved for approaches based on deterministic hill-climbing, or even gradients, to be applicable, making it possible to obtain trade-off information fairly efficiently by simple iteration. In the more general case, however, discontinuities, plateaus, and noise can make those approaches fail even for a single objective.

Control is, and has been for some time now, an important area of application for GAs and related techniques, but multiple objective and constraint handling has, with a few exceptions, invariably resorted to the aggregation of non-commensurable quantities. The tuning of various coefficients, which may vary even between problems in the same class, has been typically performed off-line and by trial and error.

In Chapters 5 and 6, the MOGA proposed in Chapter 4 was applied to three different sets of problems. The first, a simple but not trivial example in linear time-invariant optimal controller design, provides a good illustration of how preferable solutions are discovered as the GA runs. The second set of problems, based on the full non-linear model of a Pegasus gas turbine engine, provided an even more realistic example of the class of cost surfaces which may need to be handled in practice, illustrating well how some degree of preference articulation may be needed in practice to arrive at useful solutions. Examples of interaction with the GA also given at this stage showed how the decision maker can influence the progression of the search as it proceeds.

Finally, in Chapter 6, the identification of a non-linear plant from real data was considered. This example was aimed at demonstrating how multiobjective GAs are not constrained to parameter tuning problems, but rather may find application and bring innovation to other problem domains, especially if GAs have already been applied there.

7.4 Performance considerations

The question of how well a particular GA performs in comparison to other methods is generally a difficult one. For example, genetic algorithms are stochastic and normally start from a random population, whereas many other methods are deterministic and require that a starting point be supplied, which can make the comparison unfair. Also, measures of performance such as time to find a satisfactory solution (average, maximum, etc.) and quality of the best solution found in a given time (average, lowest, etc.) strictly apply only to the problem studied, and care must be taken when generalizing. Whereas other optimizers are usually aimed at well-defined classes of functions (linear, differentiable, etc.), the fact that the properties of functions amenable to genetic optimization is not yet well understood makes any generalization even more difficult.

In the general multiobjective case, the notion of performance becomes more complex, to the point where even purely describing the *multiobjective* performance of a single approach can be difficult in itself. For this reason, no direct comparison with other methods was attempted in this work.

The visualization technique proposed in Chapter 3 is a first step towards describing multiobjective algorithm performance in a way which relies solely on ordinal information (if an algorithm is insensitive to objective scaling so should any performance measure) and which recognizes that multiobjective performance is localized in objective space. For example, one algorithm may approximate better the middle-region of a trade-off surface than the extremes. Such a description of multiobjective performance should also enable the actual comparison of different multiobjective approaches in a statistically sound way in the future.

7.5 Future perspectives

While the original and main topic of this work was to devise and study novel applications of genetic algorithms in control engineering, it was always a primary concern to retain as much of their general applicability as possible in the new techniques developed. For this reason, an effort was made to separate the domain-independent aspects of multiobjective genetic algorithms (e.g., the ranking of a population given goal and priority information) from those aspects which are inherently problem dependent (the setting of goals and priorities, representation and genetic operators, etc.).

As a result, the contributions made during the course of this research have been recognized not only by the control engineering community (Fleming and Fonseca, 1993; Fonseca and Fleming, 1994; Fonseca *et al.*, 1993), but also by the genetic algorithm community (Fonseca and Fleming, 1993; Fonseca and Fleming, 1995d) and the systems engineering community at large (Fonseca and Fleming, 1995b; Fonseca and Fleming, 1995c). In addition, two research proposals, the first on "Multiobjective Genetic Algorithms" and the second on "Genetic Algorithms for Subset Selection in Identification and Control", have emerged from the work reported here and successfully been funded by EPSRC, totalling 5 man-years of funded research at post-doctoral level.

7.5.1 Evolutionary algorithms

The preferability relation, as defined, can only handle goal and priority information. This is appropriate for many applications in control system design, because goals and priorities are usually easier to set than numerical weights. However, the proposed separation between decision making and search in multiobjective evolutionary optimization opens the way for alternative decision strategies to be either developed or simply imported from decision theory. For example, a ranking strategy based on stochastic paired comparisons and possibly applicable in the evolution of game-playing programs is described by Uppuluri (1989).

The tendency that Pareto and Pareto-like ranking schemes seem to have to establish strong dependence relations between decision variables should be a matter of concern when implementing the genetic operators for a given representation. Tightly coupled decision variables are already known to pose difficulties to GAs and other evolutionary approaches, but this seems to gain particular importance in the multiobjective case. Mating restriction and on-line preference articulation somehow side-step this problem, the former by constraining the action of crossover, and the latter by allowing the Decision Maker to effectively reduce the length of the ridge which the population must cover.

The implementation of fitness sharing may also be improved. Performing fitness sharing in decision variable space (Srinivas and Deb, 1994) instead of objective space would provide a fitness assignment mechanism truly independent from objective scaling. This would imply abandoning the notion of a uniform sampling of the trade-off surface, which may or may not be desirable, but the resulting GA would then be guaranteed to perform equally well on any monotonic transformation of a given function. Unfortunately, the setting of niche sizes in the phenotypic domain is still a difficult issue. Results from statistics (Silverman, 1986) seem to find application in this area, but currently their application is restricted to Euclidean search spaces (Fonseca and Fleming, 1995a). The emergence of niches in structured populations (Davidor, 1991b) may also prove to be relevant in the multiobjective case.

Progressive articulation of preferences with GAs, while very appealing, will only be so as long as the usually large number of function evaluations that must be performed per GA generation can be computed sufficiently quickly. Here, the use of parallel processing would be particularly appropriate.

Finally, and despite growing interest, multiple objective handling has not yet found its way into the standard genetic algorithm software packages, being left mostly to the user to implement. Work is currently in progress to address this issue and integrate the multiobjective GA routines developed as part of this work with the Genetic Algorithm Toolbox for MATLAB (Chipperfield *et al.*, 1994).

7.5.2 Control systems engineering

Fully exploiting the potential of genetic algorithms in control engineering will require, at least in some cases, that problems themselves be formulated in an innovative way, since GAs are considerably different from other optimizers. For example, the formulation of cost functions for multiobjective optimization problems has been traditionally constrained by the very notion of cost function, where each individual is assigned a *well-defined* cost value. In the case of genetic algorithms, where only a relative performance measure is needed, this is unnecessary.

Naturally, it will always constitute good practice to approach scientific development in small steps which can be explained and understood easily. This is why a positive effort was made in the development of the MOGA to confine any modifications to well-defined aspects of the "simple GA". Similarly, the evolutionary approach to non-linear system identification adopted in Chapter 6 can be considered rather conservative when compared with current work in genetic programming.

While evolutionary techniques often stand accused of purely generating solutions and not contributing to the actual understanding of problems, this work has shown that genetic algorithms can indeed be used as exploration and decision-support tools. In fact, the sets of non-dominated individuals found by a GA convey information concerning both what may characterize good solutions (typically relations between decision variables) and what the performance trade-offs inherent to a particular type of solution may be. In a design situation, where the initial specifications are often stated in linguistic terms, such information can help designers form an understanding of a satisfactory solution, and possibly also stimulate any subsequent analysis of the problem.

Specific areas which, in the future, may benefit from multiobjective evolutionary techniques are, among others:

Controller design, especially in what concerns controller structure. The trade-offs between performance and controller complexity are still generally difficult to establish. Encoding both the controller structure and the parameters at the chromosome level in a genetic programming fashion should make it possible to evolve such trade-offs.

System identification, by allowing different objectives to influence the production of candidate models. Currently, models are typically produced by minimizing a single, tractable, error criterion in the hope that they will subsequently pass the validation stage. Again, genetic programming should make it possible to evolve performance-complexity trade-offs for broad classes of identified models.

Scheduling. In production systems, it is often desirable to maximize throughput and resource utilization while minimizing latency, coping with a dynamic order list and with machine failures, and complying with statutory requirements relative to the work force. A multiobjective genetic scheduler could, in the case where not all orders can be completed on time, provide various alternatives for orders to be delayed or even sacrificed in favour of others, involving a human in the decision process.

Reliability testing. The search for environmental conditions which may cause a particular design to fail is an application of evolutionary algorithms which, although not being able to provide grounds for certification, can reveal unsuspected modes of failure in that design. Again, since systems may fail in many different ways, multiobjective EAs may be able to extract further information concerning how different modes of failure may be related, and what environmental conditions can trigger them.

Appendix A

Proofs

A.1 Proof of Lemma 4.1

It suffices to show that

$$\mathbf{u} \;{}_{\mathrm{p}}\!< \mathbf{v} \;\underset{\mathbf{g}}{\Longrightarrow}\; (\mathbf{u}_{1,\ldots,i} \underset{\mathbf{g}_{1,\ldots,i}}{\prec} \mathbf{v}_{1,\ldots,i}) \vee (\mathbf{u}_{1,\ldots,i} \underset{\mathbf{g}_{1,\ldots,i}}{\equiv} \mathbf{v}_{1,\ldots,i})$$

for all $i = 1, \ldots, p$ and all $p \in \mathbb{N}$, which can be done by induction over i. The proof of the lemma is obtained by setting $i = p$.

Base clause ($i = 1$)

$$\mathbf{u} \;{}_{\mathrm{p}}\!< \mathbf{v} \;\underset{\mathbf{g}}{\Longrightarrow}\; (\mathbf{u}_1 \underset{\mathbf{g}_1}{\prec} \mathbf{v}_1) \;\vee\; (\mathbf{u}_1 \underset{\mathbf{g}_1}{\equiv} \mathbf{v}_1)$$

Proof

From Definition 3.1 (Pareto dominance), if an n-dimensional vector $\mathbf{v}$ is dominated by another vector $\mathbf{u}$, then any component u_k of $\mathbf{u}$ will be less than or equal to the corresponding component v_k of $\mathbf{v}$, with $k = 1, \ldots, n$. This also implies that any subvector of $\mathbf{u}$ will either dominate or be equal to the corresponding subvector of $\mathbf{v}$. In particular, for $\mathbf{u}_1^{\overset{\mathbf{u}}{\frown}}$ and $\mathbf{v}_1^{\overset{\mathbf{u}}{\frown}}$,

$$\mathbf{u} \;{}_{\mathrm{P}}{<}\; \mathbf{v} \;\underset{\mathbf{g}}{\Longrightarrow}\; (\mathbf{u}_1^{\overset{\mathbf{u}}{\frown}} \;{}_{\mathrm{P}}{<}\; \mathbf{v}_1^{\overset{\mathbf{u}}{\frown}}) \vee (\mathbf{u}_1^{\overset{\mathbf{u}}{\frown}} = \mathbf{v}_1^{\overset{\mathbf{u}}{\frown}})$$

If $\mathbf{u}_1^{\overset{\mathbf{u}}{\frown}} \;{}_{\mathrm{P}}{<}\; \mathbf{v}_1^{\overset{\mathbf{u}}{\frown}}$, then, by Definition 4.1, $\mathbf{u}_1$ is preferable to $\mathbf{v}_1$. Otherwise, $\mathbf{u}_1^{\overset{\mathbf{u}}{\frown}} = \mathbf{v}_1^{\overset{\mathbf{u}}{\frown}}$, and, similarly, one can write:

$$\mathbf{u} \;{}_{\mathrm{P}}{<}\; \mathbf{v} \;\underset{\mathbf{g}}{\Longrightarrow}\; (\mathbf{u}_1^{\overset{\mathbf{u}}{\smile}} \;{}_{\mathrm{P}}{<}\; \mathbf{v}_1^{\overset{\mathbf{u}}{\smile}}) \vee (\mathbf{u}_1^{\overset{\mathbf{u}}{\smile}} = \mathbf{v}_1^{\overset{\mathbf{u}}{\smile}})$$

Again by Definition 4.1, if $\mathbf{u}_1^{\overset{\mathbf{u}}{\smile}} \;{}_{\mathrm{P}}{<}\; \mathbf{v}_1^{\overset{\mathbf{u}}{\smile}}$, then $\mathbf{u}_1 \underset{\mathbf{g}_{1,\dots,p}}{\prec} \mathbf{v}_1$. Otherwise, $\mathbf{u}_1^{\overset{\mathbf{u}}{\smile}}$ is equal to $\mathbf{v}_1^{\overset{\mathbf{u}}{\smile}}$ and, by Definition 4.2, $\mathbf{u}_1$ is equivalent to $\mathbf{v}_1$.

Recursion clause $(1 < i \leq p)$

If

$$\mathbf{u} \;{}_{\mathrm{P}}{<}\; \mathbf{v} \;\underset{\mathbf{g}}{\Longrightarrow}\; (\mathbf{u}_{1,\dots,i-1} \underset{\mathbf{g}_{1,\dots,i-1}}{\prec} \mathbf{v}_{1,\dots,i-1}) \;\vee\; (\mathbf{u}_{1,\dots,i-1} \underset{\mathbf{g}_{1,\dots,i-1}}{\equiv} \mathbf{v}_{1,\dots,i-1})$$

then

$$\mathbf{u} \;{}_{\mathrm{P}}{<}\; \mathbf{v} \;\underset{\mathbf{g}}{\Longrightarrow}\; (\mathbf{u}_{1,\dots,i} \underset{\mathbf{g}_{1,\dots,i}}{\prec} \mathbf{v}_{1,\dots,i}) \;\vee\; (\mathbf{u}_{1,\dots,i} \underset{\mathbf{g}_{1,\dots,i}}{\equiv} \mathbf{v}_{1,\dots,i})$$

Proof

As before, one can write:

$$\mathbf{u} \;{}_{\mathrm{P}}{<}\; \mathbf{v} \;\underset{\mathbf{g}}{\Longrightarrow}\; (\mathbf{u}_i^{\overset{\mathbf{u}}{\frown}} \;{}_{\mathrm{P}}{<}\; \mathbf{v}_i^{\overset{\mathbf{u}}{\frown}}) \vee (\mathbf{u}_i^{\overset{\mathbf{u}}{\frown}} = \mathbf{v}_i^{\overset{\mathbf{u}}{\frown}})$$

If $\mathbf{u}_i^{\overset{\mathbf{u}}{\frown}} \;{}_{\mathrm{P}}{<}\; \mathbf{v}_i^{\overset{\mathbf{u}}{\frown}}$, then, by Definition 4.1, $\mathbf{u}_{1,\dots,i}$ is preferable to $\mathbf{v}_{1,\dots,i}$. If, on the other hand, $\mathbf{u}_i^{\overset{\mathbf{u}}{\frown}} = \mathbf{v}_i^{\overset{\mathbf{u}}{\frown}}$, then either $\mathbf{v}_i^{\overset{\mathbf{u}}{\smile}} \not\leq \mathbf{g}_i^{\overset{\mathbf{u}}{\smile}}$, in which case $\mathbf{u}_{1,\dots,i} \underset{\mathbf{g}_{1,\dots,i}}{\prec} \mathbf{v}_{1,\dots,i}$, or $\mathbf{v}_i^{\overset{\mathbf{u}}{\smile}} \leq \mathbf{g}_i^{\overset{\mathbf{u}}{\smile}}$.

In the latter case, and if the first alternative of the hypothesis is true, then $\mathbf{u}_{1,\dots,i}$ is preferable to $\mathbf{v}_{1,\dots,i}$. Otherwise, the second alternative of the hypothesis is that $\mathbf{u}_{1,\dots,i-1}$ is equivalent to $\mathbf{v}_{1,\dots,i-1}$. Since $\mathbf{u}_i^{\overset{\mathbf{u}}{\frown}} = \mathbf{v}_i^{\overset{\mathbf{u}}{\frown}}$ and $\mathbf{v}_i^{\overset{\mathbf{u}}{\smile}} \leq \mathbf{g}_i^{\overset{\mathbf{u}}{\smile}}$, the equivalence between $\mathbf{u}_{1,\dots,i}$ and $\mathbf{v}_{1,\dots,i}$ follows from Definition 4.2. $\square$

A.2 Proof of Lemma 4.2

The transitivity of the preferability relation will be proved by induction over p. The proof will be divided into three parts, the first two of which apply to both the base clause ($p = 1$) and the recursion clause ($p > 1$). In the third part, the appropriate distinction between the two clauses is made.

Base clause ($p = 1$)

$$\mathbf{u}_{1,\dots,p} \underset{\mathbf{g}_{1,\dots,p}}{\prec} \mathbf{v}_{1,\dots,p} \underset{\mathbf{g}_{1,\dots,p}}{\prec} \mathbf{w}_{1,\dots,p} \underset{\mathbf{g}_{1,\dots,p}}{\Longrightarrow} \mathbf{u}_{1,\dots,p} \underset{\mathbf{g}_{1,\dots,p}}{\prec} \mathbf{w}_{1,\dots,p}$$

Recursion clause ($p > 1$)

If

$$\mathbf{u}_{1,\dots,p-1} \underset{\mathbf{g}_{1,\dots,p-1}}{\prec} \mathbf{v}_{1,\dots,p-1} \underset{\mathbf{g}_{1,\dots,p-1}}{\prec} \mathbf{w}_{1,\dots,p-1} \underset{\mathbf{g}_{1,\dots,p-1}}{\Longrightarrow} \mathbf{u}_{1,\dots,p-1} \underset{\mathbf{g}_{1,\dots,p-1}}{\prec} \mathbf{w}_{1,\dots,p-1}$$

then

$$\mathbf{u}_{1,\dots,p} \underset{\mathbf{g}_{1,\dots,p}}{\prec} \mathbf{v}_{1,\dots,p} \underset{\mathbf{g}_{1,\dots,p}}{\prec} \mathbf{w}_{1,\dots,p} \underset{\mathbf{g}_{1,\dots,p}}{\Longrightarrow} \mathbf{u}_{1,\dots,p} \underset{\mathbf{g}_{1,\dots,p}}{\prec} \mathbf{w}_{1,\dots,p}$$

Proof

From Definition 4.1 (preferability),

$$\mathbf{u}_{1,\dots,p} \underset{\mathbf{g}_{1,\dots,p}}{\prec} \mathbf{v}_{1,\dots,p} \;\Rightarrow\; \mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} \leq \mathbf{v}_p^{\overset{\mathbf{u}}{\frown}}$$

$$\mathbf{v}_{1,\dots,p} \underset{\mathbf{g}_{1,\dots,p}}{\prec} \mathbf{w}_{1,\dots,p} \;\;\Rightarrow\;\; \mathbf{v}_p^{\overset{\mathbf{v}}{\frown}} \leq \mathbf{w}_p^{\overset{\mathbf{v}}{\frown}}$$

for all $p \geq 1$. On the other hand, since $\mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} > \mathbf{g}_p^{\overset{\mathbf{u}}{\frown}}$,

$$\mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} \leq \mathbf{v}_p^{\overset{\mathbf{u}}{\frown}} \;\;\Rightarrow\;\; \mathbf{v}_p^{\overset{\mathbf{u}}{\frown}} > \mathbf{g}_p^{\overset{\mathbf{u}}{\frown}}$$

which means that the components indicated by $\overset{\mathbf{u}}{\frown}$ are a subset of those indicated by $\overset{\mathbf{v}}{\frown}$. Therefore,

$$\mathbf{v}_p^{\overset{\mathbf{v}}{\frown}} \leq \mathbf{w}_p^{\overset{\mathbf{v}}{\frown}} \;\;\Rightarrow\;\; \mathbf{v}_p^{\overset{\mathbf{u}}{\frown}} \leq \mathbf{w}_p^{\overset{\mathbf{u}}{\frown}}\,.$$

Case I: $\mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} \;{}_{\mathrm{p}}\!< \mathbf{v}_p^{\overset{\mathbf{u}}{\frown}}$

$$(\mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} \;{}_{\mathrm{p}}\!< \mathbf{v}_p^{\overset{\mathbf{u}}{\frown}}) \wedge (\mathbf{v}_p^{\overset{\mathbf{u}}{\frown}} \leq \mathbf{w}_p^{\overset{\mathbf{u}}{\frown}}) \;\;\Rightarrow\;\; \mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} \;{}_{\mathrm{p}}\!< \mathbf{w}_p^{\overset{\mathbf{u}}{\frown}}$$

which implies $\mathbf{u}_{1,\dots,p} \underset{\mathbf{g}_{1,\dots,p}}{\prec} \mathbf{w}_{1,\dots,p}$, for all $p \geq 1$.

Case II: $(\mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} = \mathbf{v}_p^{\overset{\mathbf{u}}{\frown}}) \wedge (\mathbf{v}_p^{\overset{\mathbf{u}}{\smile}} \not\leq \mathbf{g}_p^{\overset{\mathbf{u}}{\smile}})$

$$(\mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} = \mathbf{v}_p^{\overset{\mathbf{u}}{\frown}}) \wedge (\mathbf{v}_p^{\overset{\mathbf{u}}{\frown}} \leq \mathbf{w}_p^{\overset{\mathbf{u}}{\frown}}) \;\;\Rightarrow\;\; \mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} \leq \mathbf{w}_p^{\overset{\mathbf{u}}{\frown}}$$

If $\mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} \;{}_{\mathrm{p}}\!< \mathbf{w}_p^{\overset{\mathbf{u}}{\frown}}$, then $\mathbf{u} \underset{\mathbf{g}}{\prec} \mathbf{w}$.

If $\mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} = \mathbf{w}_p^{\overset{\mathbf{u}}{\frown}}$, one must also note that $\mathbf{v}_p^{\overset{\mathbf{u}}{\smile}} \not\leq \mathbf{g}_p^{\overset{\mathbf{u}}{\smile}}$ implies that there are at least some components $\overset{\mathbf{v}}{\frown}$ in the set of components indicated by $\overset{\mathbf{u}}{\smile}$. Consequently,

$$(\mathbf{v}_p^{\overset{\mathbf{u}}{\smile}} \not\leq \mathbf{g}_p^{\overset{\mathbf{u}}{\smile}}) \wedge (\mathbf{v}_p^{\overset{\mathbf{v}}{\frown}} \leq \mathbf{w}_p^{\overset{\mathbf{v}}{\frown}}) \;\;\Rightarrow\;\; \mathbf{w}_p^{\overset{\mathbf{u}}{\smile}} \not\leq \mathbf{g}_p^{\overset{\mathbf{u}}{\smile}}$$

The preferability of $\mathbf{u}_{1,\dots,p}$ over $\mathbf{w}_{1,\dots,p}$ follows from

$$(\mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} = \mathbf{w}_p^{\overset{\mathbf{u}}{\frown}}) \wedge (\mathbf{w}_p^{\overset{\mathbf{u}}{\smile}} \not\leq \mathbf{g}_p^{\overset{\mathbf{u}}{\smile}}) \tag{A.1}$$

for all $p \geq 1$.

Case III: $(\mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} = \mathbf{v}_p^{\overset{\mathbf{u}}{\frown}}) \wedge (\mathbf{v}_p^{\overset{\mathbf{u}}{\smile}} \leq \mathbf{g}_p^{\overset{\mathbf{u}}{\smile}})$

In this case, $\overset{\mathbf{u}}{\frown}$ and $\overset{\mathbf{u}}{\smile}$ designate exactly the same sets of components as $\overset{\mathbf{v}}{\frown}$ and $\overset{\mathbf{v}}{\smile}$, respectively. In the case where $\mathbf{v}_p^{\overset{\mathbf{v}}{\frown}} \mathrel{{}_{\mathrm{p}}{<}} \mathbf{w}_p^{\overset{\mathbf{v}}{\frown}}$, one can write:

$$(\mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} = \mathbf{v}_p^{\overset{\mathbf{u}}{\frown}}) \wedge (\mathbf{v}_p^{\overset{\mathbf{u}}{\frown}} \mathrel{{}_{\mathrm{p}}{<}} \mathbf{w}_p^{\overset{\mathbf{u}}{\frown}}) \;\;\Rightarrow\;\; \mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} \mathrel{{}_{\mathrm{p}}{<}} \mathbf{w}_p^{\overset{\mathbf{u}}{\frown}}$$

which implies $\mathbf{u}_{1,\dots,p} \underset{\mathbf{g}_{1,\dots,p}}{\prec} \mathbf{w}_{1,\dots,p}$, for all $p \geq 1$.

If $\mathbf{v}_p^{\overset{\mathbf{v}}{\frown}} = \mathbf{w}_p^{\overset{\mathbf{v}}{\frown}}$, then also $\mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} = \mathbf{w}_p^{\overset{\mathbf{u}}{\frown}}$. If, in addition to that, $\mathbf{w}_p^{\overset{\mathbf{v}}{\smile}} \not\leq \mathbf{g}_p^{\overset{\mathbf{v}}{\smile}}$, one can write

$$(\mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} = \mathbf{w}_p^{\overset{\mathbf{u}}{\frown}}) \wedge (\mathbf{w}_p^{\overset{\mathbf{u}}{\smile}} \not\leq \mathbf{g}_p^{\overset{\mathbf{u}}{\smile}})$$

which implies that $\mathbf{u}_{1,\dots,p}$ is preferable to $\mathbf{w}_{1,\dots,p}$ given $\mathbf{g}_{1,\dots,p}$, for all $p \geq 1$.

If $\mathbf{w}_p^{\overset{\mathbf{v}}{\smile}} \leq \mathbf{g}_p^{\overset{\mathbf{v}}{\smile}}$, the base clause and the recursion clause must be considered separately.

Case III(a): $(p = 1)$

$$\begin{aligned}
(\mathbf{u}_1 \underset{\mathbf{g}_1}{\prec} \mathbf{v}_1) \wedge (\mathbf{u}_1^{\overset{\mathbf{u}}{\frown}} = \mathbf{v}_1^{\overset{\mathbf{u}}{\frown}}) \wedge (\mathbf{v}_1^{\overset{\mathbf{u}}{\smile}} \leq \mathbf{g}_1^{\overset{\mathbf{u}}{\smile}}) \;\;&\Rightarrow\;\; (\mathbf{u}_1^{\overset{\mathbf{u}}{\smile}} \mathrel{{}_{\mathrm{p}}{<}} \mathbf{v}_1^{\overset{\mathbf{u}}{\smile}}) \\
(\mathbf{v}_1 \underset{\mathbf{g}_1}{\prec} \mathbf{w}_1) \wedge (\mathbf{v}_1^{\overset{\mathbf{v}}{\frown}} = \mathbf{w}_1^{\overset{\mathbf{v}}{\frown}}) \wedge (\mathbf{v}_1^{\overset{\mathbf{v}}{\smile}} \leq \mathbf{g}_1^{\overset{\mathbf{v}}{\smile}}) \;\;&\Rightarrow\;\; (\mathbf{v}_1^{\overset{\mathbf{v}}{\smile}} \mathrel{{}_{\mathrm{p}}{<}} \mathbf{w}_1^{\overset{\mathbf{v}}{\smile}}) \\
&\Rightarrow\;\; (\mathbf{v}_1^{\overset{\mathbf{u}}{\smile}} \mathrel{{}_{\mathrm{p}}{<}} \mathbf{w}_1^{\overset{\mathbf{u}}{\smile}})
\end{aligned}$$

From the above, and given the transitivity of the inferiority relation, it follows that $\mathbf{u}_1^{\overset{\mathbf{u}}{\smile}} \mathrel{{}_{\mathrm{p}}{<}} \mathbf{w}_1^{\overset{\mathbf{u}}{\smile}}$, which implies that $\mathbf{u}_1$ is preferable to $\mathbf{w}_1$ given $\mathbf{g}_1$, and proves the base clause.

Case III(b): $(p > 1)$

$$
\begin{aligned}
(\mathbf{u}_{1,\ldots,p} \underset{\mathbf{g}_{1,\ldots,p}}{\prec} \mathbf{v}_{1,\ldots,p}) \wedge (\mathbf{u}_p^{\overset{\mathbf{u}}{\frown}} = \mathbf{v}_p^{\overset{\mathbf{u}}{\frown}}) \wedge (\mathbf{v}_p^{\overset{\mathbf{u}}{\smile}} \leq \mathbf{g}_p^{\overset{\mathbf{u}}{\smile}}) \quad &\Rightarrow \quad (\mathbf{u}_{1,\ldots,p-1} \underset{\mathbf{g}_{1,\ldots,p-1}}{\prec} \mathbf{v}_{1,\ldots,p-1}) \\
(\mathbf{v}_{1,\ldots,p} \underset{\mathbf{g}_{1,\ldots,p}}{\prec} \mathbf{w}_{1,\ldots,p}) \wedge (\mathbf{v}_p^{\overset{\mathbf{v}}{\frown}} = \mathbf{w}_p^{\overset{\mathbf{v}}{\frown}}) \wedge (\mathbf{v}_p^{\overset{\mathbf{v}}{\smile}} \leq \mathbf{g}_p^{\overset{\mathbf{v}}{\smile}}) \quad &\Rightarrow \quad (\mathbf{v}_{1,\ldots,p-1} \underset{\mathbf{g}_{1,\ldots,p-1}}{\prec} \mathbf{w}_{1,\ldots,p-1})
\end{aligned}
$$

From the above, and if the hypothesis is true, then $\mathbf{u}_{1,\ldots,p-1} \underset{\mathbf{g}_{1,\ldots,p-1}}{\prec} \mathbf{w}_{1,\ldots,p-1}$, which implies that $\mathbf{u}_{1,\ldots,p}$ is preferable to $\mathbf{w}_{1,\ldots,p}$ given $\mathbf{g}_{1,\ldots,p}$, and proves the recursion clause. $\square$

References

Altenberg, L. (1994). The evolution of evolvability in genetic programming. In Kinnear, Jr., K. E., editor, *Advances in Genetic Programming*, Complex Adaptive Systems, chapter 3, pages 47–74. MIT Press, Cambridge, Massachusetts.

Bäck, T. (1993). Optimal mutation rates in genetic algorithms. In (Forrest, 1993), pages 2–8.

Bäck, T. and Schwefel, H.-P. (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23.

Baker, J. E. (1987). Reducing bias and inefficiency in the selection algorithm. In (Grefenstette, 1987), pages 14–21.

Barratt, C. and Boyd, S. (1989). Example of exact trade-offs in linear control design. *IEEE Control Systems Magazine*, 9(1):46–52.

Belew, R. K. and Booker, L. B., editors (1991). *Genetic Algorithms: Proceedings of the Fourth International Conference*. Morgan Kaufmann, San Mateo, California.

Ben-Tal, A. (1980). Characterization of Pareto and lexicographic optimal solutions. In (Fandel and Gal, 1980), pages 1–11.

Booker, L. (1987). Improving search in genetic algorithms. In Davis, L., editor, *Genetic Algorithms and Simulated Annealing*, Research Notes in Artificial Intelligence, chapter 5, pages 61–73. Pitman, London.

Boyd, S. P., Balakrishnan, V., Barratt, C. H., Khraishi, N. M., Li, X., Meyer, D. G., and Norman, S. A. (1988). A new CAD method and associated architectures for linear controllers. *IEEE Transactions on Automatic Control*, 33(3):268–283.

Bruinsma, N. A. and Steinbuch, M. (1990). A fast algorithm to compute the $\mathcal{H}_\infty$-norm of a transfer function matrix. *Systems & Control Letters*, 14:287–293.

Caruana, R. A., Eshelman, L. J., and Schaffer, J. D. (1989). Representation and hidden bias II: Eliminating defining length bias in genetic search via shuffle crossover. In Sridharan, N. S., editor, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 750–755. Morgan Kaufmann.

Chen, S. and Billings, S. A. (1989). Representations of non-linear systems: the NARMAX model. *Int. J. Control*, 49(3):1013–1032.

Chen, S., Billings, S. A., and Luo, W. (1989). Orthogonal least squares methods and their application to non-linear system identification. *International Journal of Control*, 50(5):1873–1896.

Chipperfield, A., Fleming, P., Pohlheim, H., and Fonseca, C. (1994). Genetic algorithm toolbox user's guide. Research report 512, Dept. Automatic Control and Systems Eng., University of Sheffield, Sheffield, U.K.

Cieniawski, S. E. (1993). An investigation of the ability of genetic algorithms to generate the tradeoff curve of a multi-objective groundwater monitoring problem. Master's thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois.

Collins, R. J. and Jefferson, D. R. (1991). Selection in massively parallel genetic algorithms. In (Belew and Booker, 1991), pages 249–256.

Davidor, Y. (1989). Analogous crossover. In (Schaffer, 1989), pages 98–103.

Davidor, Y. (1991a). A genetic algorithm applied to robot trajectory generation. In (Davis, 1991), chapter 12, pages 144–165.

Davidor, Y. (1991b). A naturally occurring niche and species phenomenon: The model and first results. In (Belew and Booker, 1991), pages 257–263.

Davis, L., editor (1991). *Handbook of Genetic Algorithms*. van Nostrand Reinhold, New York.

Deb, K. and Goldberg, D. E. (1989). An investigation of niche and species formation in genetic function optimization. In (Schaffer, 1989), pages 42–50.

Dinkelbach, W. (1980). Multicriteria decision models with specified goal levels. In (Fandel and Gal, 1980), pages 52–59.

Fandel, G. and Gal, T., editors (1980). *Multiple Criteria Decision Making Theory and Application*, volume 177 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Berlin.

Fleming, P. J. and Fonseca, C. M. (1993). Genetic algorithms in control systems engineering. In *Proceedings of the 12th IFAC World Congress*, volume 2, pages 383–390, Sydney, Australia. Preprints.

Fleming, P. J. and Pashkevich, A. P. (1985). Computer aided control system design using a multiobjective optimization approach. In *Proc. IEE Control'85 Conference*, pages 174–179, Cambridge, U.K.

Fogarty, T. C. (1990). Adaptive rule-based optimization of combustion in multiple burner installations. In Gottlob, G. and Nejd, W., editors, *Expert Systems in Engineering: Principles and Applications*, pages 241–248. Springer-Verlag.

Fogarty, T. C., editor (1994). *Evolutionary Computing, AISB Workshop*, volume 865 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin.

Fogel, D. B. (1991). *System Identification Through Simulated Evolution: A machine learning approach to modelling*. Ginn Press, Needham, Massachusetts.

Fonseca, C. M. and Fleming, P. J. (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In (Forrest, 1993), pages 416–423.

Fonseca, C. M. and Fleming, P. J. (1994). Multiobjective optimal controller design with genetic algorithms. In *Proc. IEE Control'94 International Conference*, volume 1, pages 745–749, Warwick, U.K.

Fonseca, C. M. and Fleming, P. J. (1995a). Multiobjective genetic algorithms made easy: Selection, sharing and mating restriction. In (GALESIA, 1995), pages 45–52.

Fonseca, C. M. and Fleming, P. J. (1995b). Multiobjective optimization and multiple constraint handling with evolutionary algorithms I: A unified formulation. *IEEE Transactions on Systems, Man and Cybernetics*. To appear.

Fonseca, C. M. and Fleming, P. J. (1995c). Multiobjective optimization and multiple constraint handling with evolutionary algorithms II: Application example. *IEEE Transactions on Systems, Man and Cybernetics*. To appear.

Fonseca, C. M. and Fleming, P. J. (1995d). An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1).

Fonseca, C. M., Mendes, E. M., Fleming, P. J., and Billings, S. A. (1993). Nonlinear model term selection with genetic algorithms. In *IEE/IEEE Workshop on Natural Algorithms in Signal Processing*, volume 2, pages 27/1–27/8, Essex, U.K.

Forrest, S., editor (1993). *Genetic Algorithms: Proceedings of the Fifth International Conference*. Morgan Kaufmann, San Mateo, CA.

Fourman, M. P. (1985). Compaction of symbolic layout using genetic algorithms. In (Grefenstette, 1985), pages 141–153.

GALESIA (1995). *First IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, Sheffield, UK. The Institution of Electrical Engineers. Conference Publication No. 414.

Gembicki, F. W. (1974). *Vector Optimization for Control with Performance and Parameter Sensitivity Indices*. PhD thesis, Case Western Reserve University, Cleveland, Ohio, USA.

Goicoechea, A., Hansen, D. R., and Duckstein, L. (1982). *Multiobjective Decision Analysis with Engineering and Business Applications*. John Wiley and Sons, New York.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, Massachusetts.

Goldberg, D. E. (1991). The theory of virtual alphabets. In (Schwefel and Männer, 1991), pages 13–22.

Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In (Grefenstette, 1987), pages 41–49.

Goldberg, D. E. and Smith, R. E. (1987). Nonstationary function optimization using genetic algorithms with dominance and diploidy. In (Grefenstette, 1987), pages 59–68.

Golub, G. H. and Loan, C. F. V. (1989). *Matrix Computation*. The John Hopkins University Press, Baltimore and London, 2nd edition.

Grace, A. (1990). *Optimization Toolbox User's Guide*. The MathWorks, Inc.

Grefenstette, J. J., editor (1985). *Genetic Algorithms and Their Applications: Proceedings of the First International Conference on Genetic Algorithms*. Lawrence Erlbaum.

Grefenstette, J. J., editor (1987). *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Erlbaum.

Grefenstette, J. J. (1989). A system for learning control strategies with genetic algorithms. In (Schaffer, 1989), pages 183–190.

Grefenstette, J. J. (1992). Genetic algorithms for changing environments. In (Männer and Manderick, 1992), pages 137–144.

Grefenstette, J. J. and Fitzpatrick, J. M. (1985). Genetic search with approximate function evaluations. In (Grefenstette, 1985), pages 112–120.

Hajela, P. and Lin, C.-Y. (1992). Genetic search strategies in multicriterion optimal design. *Structural Optimization*, 4:99–107.

Hancock, P. J. B. (1994). An empirical comparison of selection methods in evolutionary algorithms. In (Fogarty, 1994), pages 80–94.

Hancock, S. D. (1992). *Gas Turbine Engine Controller Design Using Multi-Objective Optimization Techniques*. PhD thesis, University of Wales, Bangor, UK.

Hart, W. E. and Belew, R. K. (1991). Optimizing an arbitrary function is hard for the genetic algorithm. In (Belew and Booker, 1991), pages 190–195.

Harvey, I. (1992). Evolutionary robotics and saga: The case for hill crawling and tournament selection. Cognitive Science Research Paper 222, University of Sussex, Brighton, Brighton, U.K.

Hippe, P. and Stahl, H. (1987). Note on the relation between pole-zero and input-output behaviour of SISO systems and its influence in controller design. *International Journal of Control*, 45(4):1469–1477.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor.

Horn, J., Nafpliotis, N., and Goldberg, D. E. (1994). A niched Pareto genetic algorithm for multiobjective optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, volume 1, pages 82–87.

Hunt, K. J. (1992a). Controller synthesis with genetic algorithms: The evolutionary metaphor in the context of control system optimization. Report, Dept. Mechanical Engineering, University of Glasgow, Glasgow, U.K.

Hunt, K. J. (1992b). Optimal control system synthesis with genetic algorithms. In (Männer and Manderick, 1992), pages 381–389.

Hunt, K. J. (1992c). Polynomial LQG and $\mathcal{H}_\infty$ controller synthesis: A genetic algorithm solution. In *Proceedings of the IEEE Conference on Decision and Control*, Tucson, USA.

Hutton, M. F. and Friedland, B. (1975). Routh approximations for reducing order of linear, time invariant systems. *IEEE Transactions on Automatic Control*, 20(3):329–337.

Hwang, C.-L. and Masud, A. S. M. (1979). *Multiple Objective Decision Making – Methods and Applications*, volume 164 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Berlin.

Jakob, W., Gorges-Schleuter, M., and Blume, C. (1992). Application of genetic algorithms to task planning and learning. In (Männer and Manderick, 1992), pages 291–300.

Jones, G., Brown, R. D., Clark, D. E., Willet, P., and Glen, R. C. (1993). Searching databases of two-dimensional and three-dimensional chemical structures using genetic algorithms. In (Forrest, 1993), pages 597–602.

Karcz-Dulęba, I. (1994). Soft selection in D-optimal designs. In Davidor, Y., Schwefel, H.-P., and Männer, R., editors, *Parallel Problem Solving from Nature – PPSN III*, volume 866 of *Lecture Notes in Computer Science*, pages 608–616. Springer-Verlag, Berlin.

Karr, C. L. (1991). Design of an adaptive fuzzy logic controller using a genetic algorithm. In (Belew and Booker, 1991), pages 451–457.

Karr, C. L. (1992). An adaptive system for process control using genetic algorithms. In *International Symposium on Artificial Intelligence in Real-Time Control*, pages 585–590, Delft, the Netherlands. IFAC/IFIP/IMACS. Preprints.

Kristinsson, K. and Dumont, G. A. (1992). System identification and control using genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(5):1033–1046.

Kursawe, F. (1991). A variant of evolution strategies for vector optimization. In (Schwefel and Männer, 1991), pages 193–197.

Kwakernaak, H. (1986). A polynomial approach to minimax frequency domain optimization of multivariate feedback systems. *International Journal of Control*, 44(1):117–156.

Leontaritis, I. J. and Billings, S. A. (1987). Experimental design and identifiability for non-linear systems. *Int. J. Systems Sci.*, 18(1):189–202.

Linkens, D. A. and Nyongesa, H. O. (1992). A real-time genetic algorithm for fuzzy control. In *IEE Colloquium on Genetic Algorithms for Control Sys-*

tems Engineering, pages 9/1–9/4, London, U.K. The Institution of Electrical Engineers. Digest 1992/106.

Louis, S. J. and Rawlins, G. J. E. (1993). Pareto optimality, GA-easiness and deception. In (Forrest, 1993), pages 118–123.

Lucasius, C. B. and Kateman, G. (1992). Towards solving subset selection problems with the aid of the genetic algorithm. In (Männer and Manderick, 1992), pages 239–247.

Manderick, B., de Weger, M., and Spiessens, P. (1991). The genetic algorithm and the structure of the fitness landscape. In (Belew and Booker, 1991), pages 143–150.

Männer, R. and Manderick, B., editors (1992). *Parallel Problem Solving from Nature, 2*. North-Holland, Amsterdam.

Michalewicz, Z. (1993). A hierarchy of evolution programs: An experimental study. *Evolutionary Computation*, 1(1):51–76.

Michalewicz, Z. and Janikow, C. Z. (1991). Handling constraints in genetic algorithms. In (Belew and Booker, 1991), pages 151–157.

Miller, G. F. (1994). Exploiting mate choice in evolutionary computation: Sexual selection as a process of search, optimization and diversification. In (Fogarty, 1994), pages 65–79.

Mühlenbein, H. and Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm i. continuous parameter optimization. *Evolutionary Computation*, 1(1):25–49.

Murdock, T. M., Schmitendorf, W. E., and Forrest, S. (1991). Use of a genetic algorithm to analyze robust stability problems. In *Proceedings of the 1991*

American Control Conference, volume 1, pages 886–889, Boston, Massachussets. American Automatic Control Council.

Oakley, E. H. N. (1994). The application of genetic programming to the investigation of short, noisy, chaotic data series. In (Fogarty, 1994), pages 320–332.

Oliveira, P., Sequeira, J., and Sentieiro, J. (1991). Selection of controller parameters using genetic algorithms. In Tzafestas, S. G., editor, *Engineering Systems with Intelligence*, pages 431–438. Kluwer Academic, the Netherlands.

Powell, D. and Skolnick, M. M. (1993). Using genetic algorithms in engineering design optimization with non-linear constraints. In (Forrest, 1993), pages 424–431.

Rechenberg, I. (1973). *Evolutionsstrategie, Optimierung technischer Systeme nach Prinzipen der biologischen Evolution.* frommann-holzboog, Stuttgart. In German.

Rechenberg, I. (1994). *Evolutionsstrategie '94.* frommann-holzboog, Stuttgart. In German.

Reeves, C. R. (1994). Genetic algorithms and neighbourhood search. In (Fogarty, 1994), pages 115–130.

Richardson, J. T., Palmer, M. R., Liepins, G., and Hilliard, M. (1989). Some guidelines for genetic algorithms with penalty functions. In (Schaffer, 1989), pages 191–197.

Ritzel, B. J., Eheart, J. W., and Ranjithan, S. (1994). Using genetic algorithms to solve a multiple objective groundwater pollution containment problem. *Water Resources Research*, 30(5):1589–1603.

Schaffer, J. D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In (Grefenstette, 1985), pages 93–100.

Schaffer, J. D., editor (1989). *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann, San Mateo, CA.

Schaffer, J. D. (1993). Personal communication.

Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models.* Wiley, Chichester.

Schwefel, H.-P. and Männer, R., editors (1991). *Parallel Problem Solving from Nature, 1st Workshop, Proceedings*, volume 496 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin.

Shutler, A. G. (1991). A case study problem for the advanced controls technology club. Internal Report CSAN 2790, Rolls Royce plc, Bristol, U.K.

Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis*, volume 26 of *Monographs on Statistics and Applied Probability.* Chapman and Hall, London.

Solano, J. (1992). *Parallel Computation of Robot Motion Planning Algorithms.* PhD thesis, University of Wales, Bangor, UK.

Srinivas, N. and Deb, K. (1994). Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3). To appear.

Swansea (1991). Turbofan engine model. Technical report, Control and CAD Group, University College of Swansea, Swansea, U.K. Final draft.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In (Schaffer, 1989), pages 2–9.

Syswerda, G. (1991). A study of reproduction in generational and steady state genetic algorithms. In Rawlins, G. J. E., editor, *Foundations of Genetic Algorithms*, part 2, pages 94–101. Morgan Kaufmann, San Mateo, CA.

Syswerda, G. and Palmucci, J. (1991). The application of genetic algorithms to resource scheduling. In (Belew and Booker, 1991), pages 502–508.

The MathWorks (1992a). MATLAB *Reference Guide*. The MathWorks, Inc.

The MathWorks (1992b). SIMULINK *User's Guide*. The MathWorks, Inc.

Uppuluri, V. R. R. (1989). Prioritization techniques based on stochastic paired comparisons. In Karpak, B. and Zionts, S., editors, *Multiple Criteria Decision Making and Risk Analysis Using Microcomputers*, volume 56 of *NATO ASI Series F: Computer and Systems Sciences*, pages 293–303. Springer-Verlag, Berlin.

Voon, W. S. F. (1984). *Nonlinear Parameter Estimation Techniques*. PhD thesis, Dept. Automatic Control and Systems Eng., University of Sheffield, Sheffield, U.K.

Wagner, G. P. (1988). The influence of variation and of developmental constraints on the rate of multivariate phenotypic evolution. *Journal of Evolutionary Biology*, 1(1):45–66.

Walsh, G. R. (1975). *Methods of Optimization*. Wiley.

Whitley, D., Starkweather, T., and Shaner, D. (1991). The travelling salesman and sequence scheduling: Quality solutions using genetic edge recombination. In (Davis, 1991), chapter 22, pages 350–372.

Wienke, D., Lucasius, C., and Kateman, G. (1992). Multicriteria target vector optimization of analytical procedures using a genetic algorithm. Part I. Theory, numerical simulations and application to atomic emission spectroscopy. *Analytica Chimica Acta*, 265(2):211–225.

Wilson, P. B. and Macleod, M. D. (1993). Low implementation cost IIR digital filter design using genetic algorithms. In *IEE/IEEE Workshop on Natural Algorithms in Signal Processing*, volume 1, pages 4/1–4/8, Chelmsford, U.K.

Zakian, V. and Al-Naib, U. (1970). Design of dynamical and control systems by the method of inequalities. *Proceedings of the IEE*, 120(11):1421–1427.