Evolutionary Multi–Objective Decision Support Systems for Conceptual Design

by

Dragan Cvetković

A thesis submitted to the University of Plymouth

in partial fulfilment for the degree of

DOCTOR OF PHILOSOPHY

School of Computing

Faculty of Technology

University of Plymouth

In collaboration with

British Aerospace Systems, Warton

July 2000

Evolutionary Multi–Objective Decision Support Systems for Conceptual Design

Dragan Cvetković

Abstract

In this thesis the problem of conceptual engineering design and the possible use of adaptive search techniques and other machine based methods therein are explored. For the multi–objective optimisation (MOO) within conceptual design problem, genetic algorithms (GA) adapted to MOO are used and various techniques explored: weighted sums, lexicographic order, Pareto method with and without ranking, VEGA–like approaches etc. Large number of runs are performed for finding the optimal configuration and setting of the GA parameters. A novel method, *weighted Pareto method* is introduced and applied to a real–world optimisation problem.

Decision support methods within conceptual engineering design framework are discussed and a new preference method developed. The preference method for translating vague qualitative categories (such as "more important", "much less important" etc.) into quantitative values (numbers) is based on fuzzy preferences and graph theory methods. Several applications of preferences are presented and discussed:

- in weighted sum based optimisation methods;
- in weighted Pareto method;
- for ordering and manipulating constraints and scenarios;
- for a co-evolutionary, distributive GA–based MOO method;

The issue of complexity and sensitivity is addressed as well as potential generalisations of presented preference methods. Interactive dynamical constraints in the form of design scenarios are introduced. These are based on a propositional logic and a fairly rich mathematical language. They can be added, deleted and modified on–line during the design session without need for recompiling the code.

The use of machine–based agents in conceptual design process is investigated. They are classified into several different categories (e.g. interface agents, search agents, information agents). Several different categories of agents performing various specialised task are developed (mostly dealing with preferences, but also some filtering ones). They are integrated with the conceptual engineering design system to form a closed loop system that includes both computer and designer.

All these different aspects of conceptual engineering design are applied within Plymouth Engineering Design Centre / British Aerospace conceptual airframe design project.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank my supervisor Dr Ian Parmee and Plymouth Engineering Design Centre colleagues for their support during my research.

I would also like to thank to Dr Kalyanmoy Deb and to Dr Carlos A. Coello Coello for discussion and for ideas that emerged during those discussion.

Thanks also goes to British Aerospace for their support of the project and especially to Erik Webb and Richard Dell.

And last, but not least I would like to my wife Jana for her encouragement, support and bearing with me during this period.

Declaration

At no time during the registration for the degree Doctor of Philosophy has the author been registered for any other University award.

This study was financed by the EPSRC and British Aerospace plc.

Throughout the course of the research, close links were maintained with British Aerospace plc. The research has been presented at the following international conferences and in the following journals:

Cvetković, D. & Parmee, I. C. (1998), Evolutionary design and multi–objective optimisation, *in* '6th European Congress on Intelligent Techniques and Soft Computing EUFIT'98', Aachen, Germany, pp. 397–401.

Cvetković, D. & Parmee, I. C. (1999*a*), Genetic algorithm–based multi–objective optimisation and conceptual engineering design, *in* 'Proceedings of the 1999 Congress on Evolutionary Computation – CEC99', IEEE, Washington D.C., USA, pp. 29–36.

Cvetković, D. & Parmee, I. C. (1999*b*), Genetic algorithms based systems for conceptual engineering design, *in* U. Lindemann, H. Birkhofer, H. Meerkamm & S. Vajna, eds, 'Proceedings of the 12th International Conference on Engineering Design ICED'99', Vol. 2, TU München, München, Germany, pp. 1035–1038.

Cvetković, D. & Parmee, I. C. (1999*c*), Use of preferences for GA–based multi–objective optimisation, *in* W. Banzhaf & J. D. et al., eds, 'GECCO–99: Proceedings of the Genetic and Evolutionary Computation Conference', Morgan Kaufmann, Orlando, Florida, USA, pp. 1504–1509.

Cvetković, D. & Parmee, I. C. (2000*a*), Agentt–based support within an interactive evolutionary design system, Technical report, PEDC, University of Plymouth, Plymouth, UK. submitted to 'Research in Engineering Design' journal.

Cvetković, D. & Parmee, I. C. (2000*b*), The application of genetic algorithms and preferences in engineering design, Technical Report PEDC-01-2000, PEDC, University of Plymouth, Plymouth, UK.

Cvetković, D. & Parmee, I. C. (2000*c*), Designer's preferences and multi–objective preliminary design processes, *in* I. C. Parmee, ed., 'Evolutionary Design and Manufacture: Selected Papers from ACDM'00', Springer, London, pp. 249–260.

Cvetković, D. & Parmee, I. C. (2000*d*), Preferences and their application in multi–objective optimisation, Technical report, PEDC, University of Plymouth, Plymouth, UK. submitted to IEEE Transactions on Evolutionary Computation journal.

Cvetković, D., Parmee, I. C. & Webb, E. (1998), Multi–objective optimisation and preliminary airframe design, *in* I. C. Parmee, ed., 'Adaptive Computing in Design and Manufacture', Springer–Verlag, pp. 255–267.

Parmee, I. C., Cvetković, D., Bonham, C. R. & Mitchell, D. (2000), Towards interactive evolutionary design systems for the satisfaction of multiple and changing objectives, *in* 'European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2000)', Barcelona, Spain.

Parmee, I. C., Cvetković, D., Bonham, C. R. & Packham, I. S. (2000), Introducing prototype interactive evolutionary systems for ill–defined multi–objective design environments, Technical report, PEDC, University of Plymouth, Plymouth, UK. Submitted to 'Advances in Engineering Software'.

Parmee, I. C., Cvetković, D., Bonham, C. R. & Watson, A. H. (2000), Interactive evolutionary conceptual design systems, *in* J. S. Gero, ed., 'Artificial Intelligence in Design'00', Kluwer Academic Publishers, Dordrecht, pp. 249–268.

Parmee, I. C., Cvetković, D., Watson, A. H. & Bonham, C. R. (2000), 'Multi–objective satisfaction within an interactive evolutionary design environment', *Evolutionary Computation* **8**(2), pp. 197–222.

Signed: D. Cvetković

Date: 11/11/2000

CHAPTER 1

Introduction

In this thesis different adaptive and preference methods are explored and developed and applied to the conceptual engineering design process.

This chapter explains the basic problems in conceptual engineering design and the next chapter will explain the basic optimisation methods that are being used in this thesis.

This following quote (from "I, Robot" by Isaac Asimov, first published in 1950, (Asimov 1982, p. 697)) very much describes the problem of conceptual design and the fuzziness of that process:

> *"My dear Byerley, I see that you instinctively follow that great error – that the Machine knows all. Let me cite you a case from my personal experience. The cotton industry engages experienced buyers who purchase cotton. Their procedure is to pull a tuft of cotton out of a random bale of a lot. They will look at that tuft and feel it, tease it out, listen to the crackling perhaps as they do, touch it with their tongue, – and through this procedure they will determine the class of cotton the bales represent. There are about a dozen such classes. As a result of their decisions, purchases are made at certain prices, blends are made in certain proportions. — Now these buyers cannot yet be replaced by the Machine"*
>
> *"Why not? Surely the data involved is not too complicated for it?"*
>
> *"Probably not. But what data is this you refer to? No textile chemist knows exactly what it is that the buyers tests when he feels the tuft of cotton. Presumably there's the average length of the threads, their feel, the extent and nature of their slickness, the way they hang together and so on. — Several dozen items, subconsciously weighted, out of years of experience. But the* quantitative *nature of these tests is not known. So we have nothing to feed the Machine. Nor can the buyers explain their own judgement. They can only say, 'Well, look at it. Can't you* tell *it's class–such–and–such?"'*
>
> *"I see."*
>
> *"There are innumerable cases like that. The Machine is only a tool after all, which can help humanity progress faster by taking some of the burden of calculations and interpretations off his back. The task of the human brain remains what it has always been; that of discovering new data to be analysed, and of devising new concepts to be tested."*

1.1 Engineering design – A short introduction

Phadke (1989, p. 1) gives the following short definition of engineering design and its objective:

> *The objective of engineering design, a major part of research and development (R&D) is to produce drawings, specifications, and other relevant information needed to manufacture products that meet customer requirements.*

According to Pahl & Beitz (1996), "...the main task of engineers is to apply their scientific and engineering knowledge to the solution of the technical problems, and then to optimise those solutions within the requirements and constraints set by material, technological, economical, legal, environmental and human–related considerations. Problems become concrete tasks after the clarification and definition of the problems which engineers have to solve to create new technical products (artifacts). The mental creation of a new product is the task of design or development engineers, whereas its physical realisation is the responsibility of manufacturing engineers. [...] Designers contribute to finding solutions and developing products in a very specific way. They carry a heavy responsibility since their ideas, knowledge and skills determine in a decisive way the technical, economic and ecological properties of the product."

The activities of designers can be roughly classified into the following (Pahl & Beitz 1996):

1. **Conceptualising** i.e. searching for solution principles;
2. **Embodying** i.e. engineering a solution principle by determining the general arrangement and preliminary shapes and materials of all components;
3. **Detailing** i.e. finalising production and operational details;
4. **Computing** drawing and information collecting. These occur during all phases of the design process.

Accordingly, there are four stages of design process (Sen & Yang 1998, Pahl & Beitz 1996):

1. **Conceptual design** is that part of the design process in which, by the identification of the essential problems through abstraction, by the establishment of function structures and by the search for appropriate working principles and their combination, the basic solution path is laid down through the elaboration of a solution principle. Conceptual design *determines the principle* of a solution (Pahl & Beitz 1996, p. 139).
2. **Preliminary design.** Some authors do not consider preliminary design as a separate stage and consider it a part of conceptual design. According to Dym (1994, p. 33), the preliminary layout is obtained by refining the conceptual designs and ranking them against the design specifications, and choosing the best as the preliminary design.
3. **Embodiment design** is that part of design process in which, starting from the working structure or concept of a technical product, the design is developed, in accordance with technical and economic criteria and in the light of further information, to the point where subsequent detail design can lead directly to production (Pahl & Beitz 1996, p. 199).
4. **Detail design** is that part of the design process which completes the embodiment of technical products with final instructions about the layout, forms, dimensions and surface properties of all individual components, the definitive selection of materials and a final scrutiny of the production methods, operating procedures and costs (Pahl & Beitz 1996, p. 400).

Design process in its most general framework is presented in Figure 1.1. However, it is seldom so straight-forward, in most cases it corresponds more to a design spiral (Sen & Yang 1998, p. 4) where the requirements of design are met incrementally until some compromising design criteria have been met.

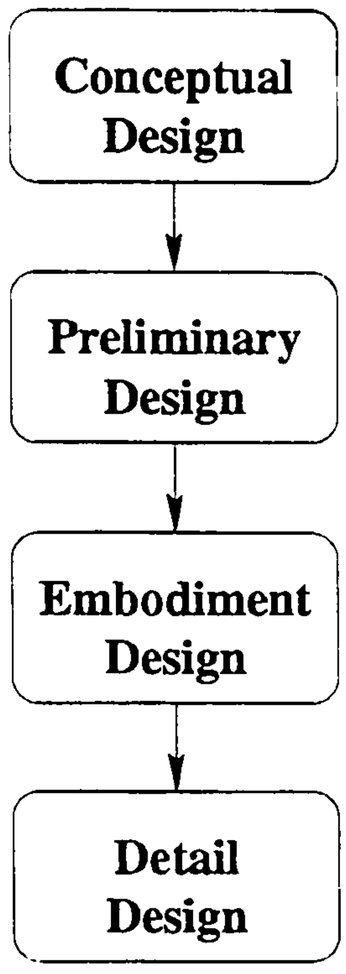


Figure 1.1. Simplified model of design process

The following classification of engineering design in 4 phases in not the unique one: some authors make distinction between embodiment and detailed design (Pahl & Beitz 1996), some others make difference between conceptual and preliminary design, but it all very much depends on the product developed.

On the more general level, design consists of a loop: product design ↔ manufacturing ↔ marketing ↔ improvement ↔ product design (Suh 1990) as presented in Figure 1.2.

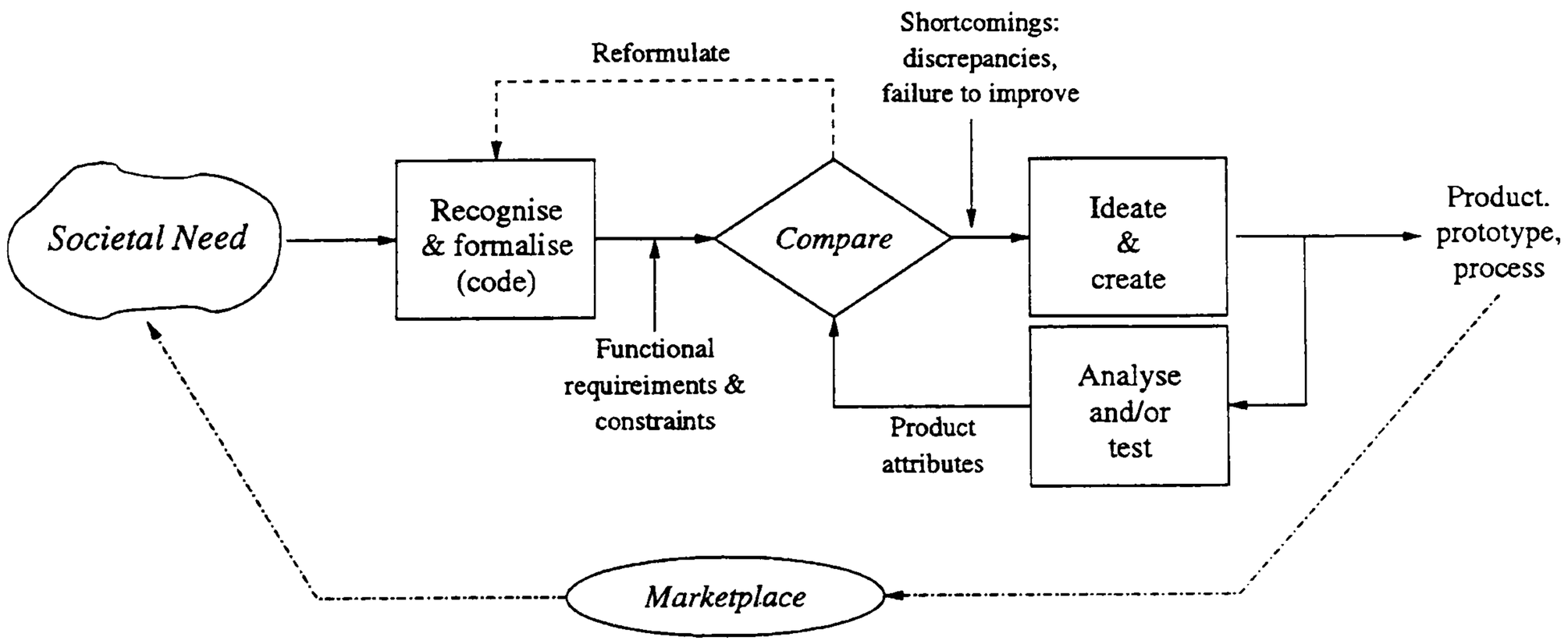


Figure 1.2. The design loop.

1.1.1 Creativity in design

Creativity is a very important component of some design phases. According to the level of creativity involved, design can be classified into the following 4 categories (Bahrami & Dagli 1994):

Creative Design: A priori plan for the solution of the problem does not exist. Design is an abstract decomposition of the problem into a set of levels that represents choices for the components of the problem. The key element in this design type is the transformation from the subconscious to conscious.

Innovative Design: The decomposition of the problem is known, but the alternatives for each of its subparts do not exist and must be synthesised. Design might be an original or unique combination of existing components. It can be argued that a certain amount of creativity comes into play in the innovative design process.

Redesign: An existing design is modified to meet required changes in the original functional requirements;

Routine Design: A priori plan of the solution exists. The subparts and alternatives are known in advance, perhaps as a result of either a creative or innovative design process. Routine design involves finding the appropriate alternatives for each subpart that satisfy the given constraints.

At the creative end of the spectrum, design is very fuzzy. As it moves to routine design, it gets precise, crisp and predetermined.

Gero (1990) gives the following classification of creativity in design:

- if the design variables and the ranges of values they can take remain fixed during design processing, then the process is routine design;
- if the design variables remain fixed but the ranges of value change, then it is innovative design;
- if design variables change too, then it is creative design

According to (Goel 1997):

> *...problem formulation and reformulation are integral parts of creative design. Designers' understanding of a problem typically evolves during creative design processing. This evolution of problem understanding may lead to (possible radical) changes in the problem and solution representations. [...] in creative design, knowledge needed to address a problem typically is not available in a form directly applicable to the problem. Instead, at least some of the needed knowledge has to be acquired from other knowledge sources, by analogical transfer from a different problem for example. [...] creativity in design lies on a continuum. That is, creativity in design may occur in degrees, where the degree of creativity may depend on the extent of problem and solution reformulation and the transfer of knowledge from different knowledge sources to the design problem.*

1.1.2 Design functions

The design function in engineering design is best described with a following quote:

> *Problem solving is common to all engineering work. The problem may involve quantitative or qualitative factors; it may be physical or economic; it may require abstract mathematics or common sense. Of great importance is the process of creative synthesis or design, putting ideas together to create a new and optimum solution.*

Although engineering problems vary in scope and complexity, the same general approach is applicable. First comes an analysis of the situation and a preliminary decision on a plan of attack. In line with this plan, the problem is reduced to a more categorical question that can be clearly stated. The stated question is then answered by deductive reasoning from known principles or by creative synthesis, as in a new design. The answer or design is always checked for accuracy and adequacy. Finally, the results for the simplified problem are interpreted in terms of the original problem and reported in an appropriate form.

In order of decreasing emphasis on science, the major functions of all engineering branches are the following:

Research *Using mathematical and scientific concepts, experimental techniques, and inductive reasoning, the research engineer seeks new principles and processes.*

Development *Development engineers apply the results of research to useful purposes. Creative application of new knowledge may result in a working model of a new electrical circuit, a chemical process, or an industrial machine.*

Design *In designing a structure or a product, the engineer selects methods, specifies materials, and determines shapes to satisfy technical requirements and to meet performance specifications.*

Construction *The construction engineer is responsible for preparing the site, determining procedures that will economically and safely yield the desired quality, directing the placement of materials, and organizing the personnel and equipment.*

Production *Plant layout and equipment selection are the responsibility of the production engineer, who chooses processes and tools, integrates the flow of materials and components, and provides for testing and inspection.*

Operation *The operating engineer controls machines, plants, and organizations providing power, transportation, and communication; determines procedures; and supervises personnel to obtain reliable and economic operation of complex equipment.*

Management and other functions *In some countries and industries, engineers analyze customers' requirements, recommend units to satisfy needs economically, and resolve related problems.*

With the integration of computers in everyday life, computers also play a major role in engineering design.

Since the mid-1960s, computer technology has been continually developed to the point at which aircraft and engine designs can be simulated and tested in myriad variations under a full spectrum of environmental conditions prior to construction. As a result, practical consideration may be given to a series of aircraft configurations, which, while occasionally and usually unsuccessfully attempted in the past, can now be used in production aircraft. These include forward swept wings, canard surfaces, blended body and wings, and the refinement of specialized airfoils (wing, propeller, and turbine blade). With this goes a far more comprehensive understanding of structural requirements, so that adequate strength can be maintained even as reductions are made in weight.

1.1.3 Basic problems in conceptual design

Some of the basic problems of conceptual design are briefly explained below. Those are the requirements arising in discussion with industrial partners (Parmee & Purchase 1997, Parmee 1998*a*, Parmee & Bonham 1998, Cvetković, Parmee & Webb 1998):

- There are objectives and there are constraints. The difference between them is very fuzzy and some of them will move from objectives to constraints or vice versa. Some constraints are hard, some not; some will change or disappear whilst others may be introduced as the problem knowledge base expands.

- In many cases the variable ranges are also fuzzy and flexible and there is a requirement for exploration outside of the default regions. The reason is that the real bounds and limits are not always known from the very beginning and could be rather artificial limitations.

- The output should contain both optimal solutions and suggestions of extending ranges and/or inclusion/removals of constraints.

- A set of results is required which the engineer can analyse off-line. That means that the engineer should be able to input those results to some other programs or to consult some database or persons for different aspects of given solutions.

- The end–user (designer) should not be confused with the number of parameters and possibilities the (optimisation) program offers, cognitive overload must be avoided.

- The engineer wishes to interact with the search process by sampling results after N functions evaluations, and adapting parameters and/or constraints.

The problems of conceptual design relate to the fuzzy nature of initial design concepts and the many different variants that engineers wish to try. Computers should be able to help them in exploration of those variants whilst suggesting some others as well. This problem has been investigated in the Plymouth Engineering Design Centre (PEDC) before (Parmee & Denham 1994, Parmee 1997).

1.1.4 Phases of aircraft design

The collaborative industrial project with British Aerospace (BAe) relates to airframe design and development. The electronic version of Encyclopædia Britannica quotes the following regarding the design of aircrafts (emphasis are mine). However, the similar principles and phases apply to almost any complex product design.

> *... The design of a flight vehicle is a complex and time–consuming procedure requiring the integration of many engineering technologies. Supporting teams are formed to provide expertise in these technologies, resulting in a completed design that is the best compromise of all the engineering disciplines. Usually the support teams are supervised by a project engineer or chief designer for technical guidance and by a program manager responsible for program budgets and schedules. Because of the ever–increasing requirement for advanced technology and the high cost and high risk associated with complex flight vehicles, many research and development programs are cancelled before completion. (see also Index: industrial design)*
>
> *The design process can be dissected into five phases and is the same for most aerospace products.* ***Phase one is a marketing analysis*** *to determine customer specifications or requirements. Aerospace engineers are employed to examine technical, operational, or financial problems. The customer's requirements are established and then passed on to the* ***conceptual design team for the second phase.***
>
> *The conceptual design team generally consists of aerospace engineers, who make the first sketch attempt to determine the vehicle's size and configuration. Preliminary estimates of the vehicle's performance, weight, and propulsion systems are made. Performance parameters include range, speed, drag, power required, payload, and takeoff and landing distances. Parametric trade studies are conducted to optimize the design, but configuration details usually change. This phase may take from a few months to years for major projects.*
>
> ***Phase three is the preliminary design phase.*** *The optimized vehicle design from phase two is used as the starting point. Aerospace engineers perform computer analyses on the configuration;*

then wind-tunnel models are built and tested. Flight control engineers study dynamic stability and control problems. Propulsion groups supply data necessary for engine selection. Interactions between the engine inlet and vehicle frame are studied. Civil, mechanical, and aerospace engineers analyze the bending loads, stresses, and deflections on the wing, airframe, and other components. Material science engineers aid in selecting low-weight, high-strength materials and may conduct aeroelastic and fatigue tests. Weight engineers make detailed estimates of individual component weights. As certain parameters drive the vehicle design, the preliminary designers are often in close contact with both the conceptual designers and the marketing analysts. The time involved in the preliminary design phase depends on the complexity of the problem but usually takes from six to 24 months.

***Phase four, the detailed design phase**, involves construction of a prototype. Mechanical engineers, technicians, and draftsmen help lay out the drawings necessary to construct each component. Full-scale mock-ups are built of cardboard, wood, or other inexpensive materials to aid in the subsystem layout. Subsystem components are built and bench-tested, and additional wind-tunnel testing is performed. This phase takes from one to three years.*

***The final phase concerns flight-testing the prototype**. Engineers and test pilots work together to assure that the vehicle is safe and performs as expected. If the prototype is a commercial transport aircraft, the vehicle must meet the requirements specified by government organizations such as the Federal Aviation Administration in the United States and the Civil Aviation Authority in the United Kingdom. Prototype testing is usually completed in one year but can take much longer because of unforeseen contingencies. The time required from the perception of a customer's needs to delivery of the product can be as long as 10 to 15 years depending on the complexity of the design, the political climate, and the availability of funding.*

High-speed computers have now enabled complex aerospace engineering problems to be analyzed rapidly. More extensive computer programs, many written by aerospace engineers, are being formulated to aid the engineer in designing new configurations.

Discussion with British Aerospace engineers reveals that the project design in their company is roughly as presented in Figure 1.3.

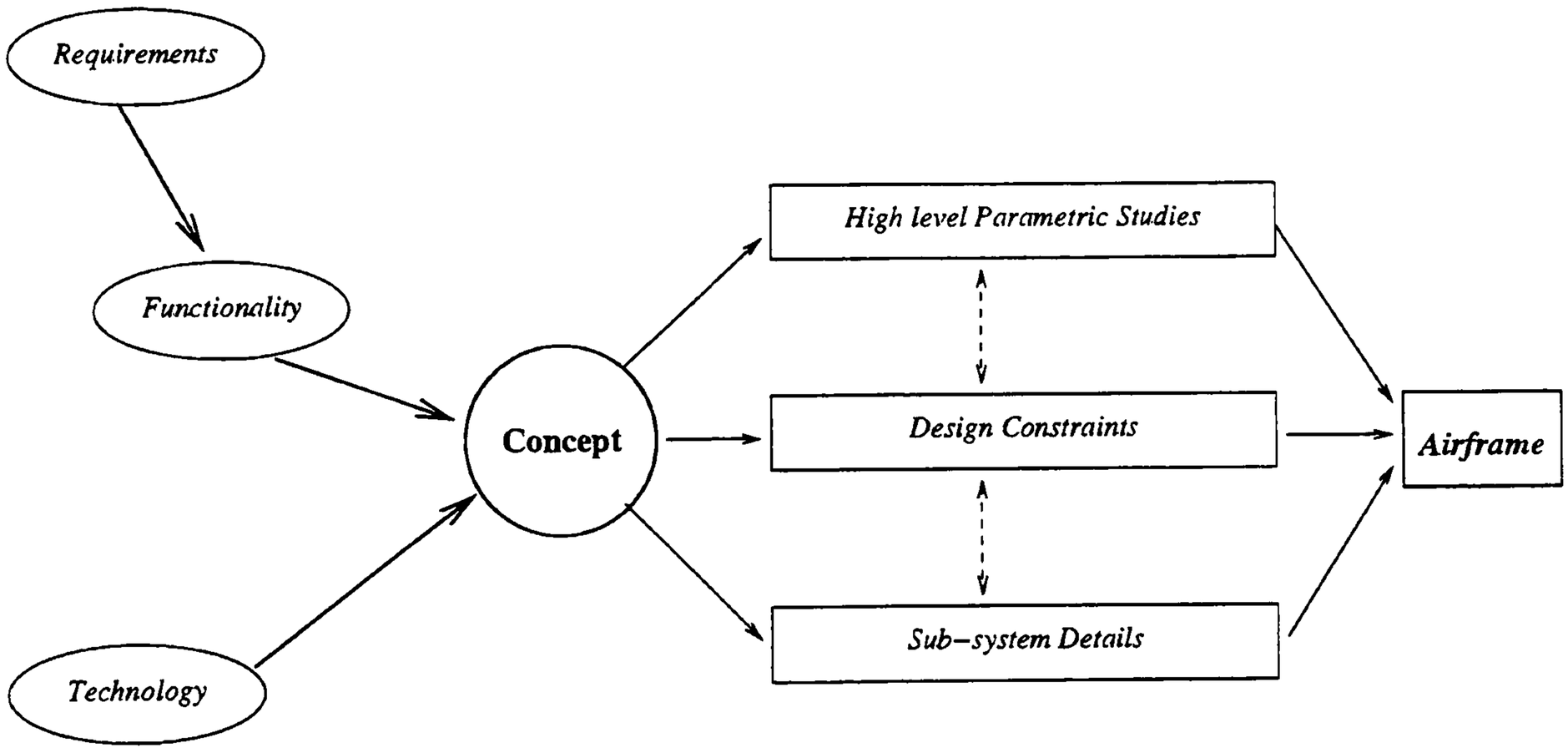


Figure 1.3. Process design

1.2 Multiple criteria decision making and conceptual design

Some more problems of conceptual design are described by Carlsson (1996, 1998) in his talk about "Soft Computing and Decision Support System". He urges the step from Multiple Criteria Decision Making (MCDM) to Multiple Criteria Decision Aid (MCDA).

- Managers are often not satisfied with the optimal solution obtained using MCDM.
- Preferences are formed in a learning process;
- We have a set of designers with different opinions and preferences;
- Optimal solution is created, not found;
- Imprecision, interaction, flexibility ... are needed;
- Support for ambiguity handling and uncertainty;
- There is a need to move from a rational to pro-active approach;

Conceptual design process could be very well described using the *Stream metaphor* (Jantsch 1980): there is a rational approach (this describes the stream in every detail) and pro-active approach (we are an integral part of the stream). There is a third approach which lies somewhere in between.

Multiple Criteria Decision Making (MCDM) and Multiple Criteria Decision Aid (MCDA) are described in more details in section 5.2, page 48.

1.3 Overview of the thesis

This thesis is organised in the following way:

Chapter 2 gives the general description of the genetic algorithms and some of the main operators and techniques used in GA field. It also describes genetic operators (i.e. crossover and mutation) suitable for real–valued chromosomes.

Chapter 3 gives the description of the British Aerospace (BAe) problem and the methods in dealing with multi–objective optimisation. This includes weighted sums, lexicographic order, Pareto with and without ranking etc. One new ranking method and a weighted Pareto method are introduced. They both are original work.

Chapter 4 describes the specific operators and techniques for the GA that was used for the BAe problem optimisation as well as an optimal parameter setting for the GA in the BAe function.

Chapter 5 describes one new method of preferences, its properties and complexity and sensitivity of the preference procedure. The preference method developed is original work and together with the applications represent a main contribution of this thesis to conceptual design and genetic algorithm fields.

Chapter 6 describes the applications of preferences to multi–objective optimisation. Applications include weighted sum multi–objective optimisation, weighted Pareto method and weighted co-evolutionary optimisation. Some of the applications are quite straightforward, but weighted Pareto method and weighted co-evolutionary optimisation present original integration of preferences with some well established optimisation methods.

Chapter 7 introduces the two different concepts of scenarios, their use and some applications. Although scenarios are routinely used in engineering, scenarios developed here are not built on any previous work and therefore represent original work.

Chapter 8 describes some aspects of agent theory, their use in general and in conceptual design in particular. All developed agents, regardless of their complexity, present original work.

Chapter 9 gives the conclusion of the Thesis, discussion and some further research pointers.

At the end of the thesis, there are 3 appendices for the sake of completeness of the thesis including a listing of preference algorithm and list of all scenarios used.

CHAPTER 2

Genetic Algorithms

This chapter describes the basics of genetic algorithms (GA). The first section describes simple GA (SGA) and the later sections introduce the real–valued GA (RGA) and genetic operators suitable for RGA.

2.1 Genetic algorithms – An introduction

Genetic algorithms (GAs) are an optimisation technique that imitates nature: they contain selection, crossover and mutation. The basics of GAs (i.e. Simple Genetic Algorithm (SGA)) are described in (Cvetković 1993) or, widely available in (Goldberg 1989). More advanced literature is given in (Holland 1975).

Almost every genetic algorithm has the structure as presented in Figure 2.1.

```
procedure GA;
    begin
        initialise population P(0);
        evaluate P(0);
        t := 1;
        repeat
            select P(t) from P(t − 1);
            recombine P(t);
            evaluate P(t);
            t := t + 1;
        until (termination condition);
    end.
```

Figure 2.1. The Simple Genetic Algorithm

A genetic algorithm must have 5 components to solve a problem:

- a chromosomal representation,
- a way to create an initial population,
- an evaluation function,
- recombination operators,

- values of the parameters.

A short description of these components is given below. It mostly addresses Simple Genetic Algorithms (SGA).

Chromosomal representation of the problem: Usually, chromosomes are bit strings — strings of 0's and 1's. Of course, other representations (real–valued, lists, trees etc.) are possible if they are more suitable for a specific problem.

Initialising population: The initial population is mostly chosen at random, but it can also be chosen heuristically. This should be done carefully since GAs may quickly converge to a local optimum if the initial population contains a few structures that are far superior to the rest of the population. The techniques used include perturbations of the output of a greedy algorithm, weighted random initialisations, and initialisation by perturbing the results of a human solution to the given problem. The population can also be initialsed by choosing elements with maximal Hamming distance from each other using e.g. Halton sequences (Kocis & Whiten 1997).

Evaluation function: The evaluation function plays the role of the environment, rating solutions in terms of their "fitness" and incorporate rule "survival of the fittest". It evaluates members of the given population, and should have the maximum values at the optimal solutions. Sometimes, fine–tuning evaluation function is very important for GA to obtain best results, so techniques such as scaling, normalisation etc. are used.

Selecting next population: The next population is chosen from the previous one in a process in which individual strings (chromosomes) are copied according to their fitness function. This means that that strings with a higher fitness value have a higher probability of contributing one or more offspring in the next generation.

Recombination operations: After choosing the next population from the previous one, recombination operations are performed on them. The main operations are: crossover, mutation and inversion, but for many problems it is possible to define recombination operators that take problem specific knowledge into account. Description of some of the other (less–often used) operators (such as dominance, diploidy, duplication, deletion etc.) can be found in (Goldberg 1989).

Crossover: Simple crossover is done in two steps. First, members of current population are mated at random. Second, each pair of strings undergoes crossover as follows: a position along the string is selected at random and one or two new strings are created by swapping all alleles between that position and end of string. Besides this *one–point* crossover, there are its generalisations: *n*–point crossover and uniform crossover (Syswerda 1989).

Mutation: Mutation is random (with small probability) alternation of the value of a string position. When used sparingly with other operations, it is an insurance policy against premature loss of

important notions. If used with too high probability (close to 1), a random search process might as well be used — the effect is the same and the process is much easier.

Inversion: Inversion inverts the sequence between two randomly assigned points in a single string. Inversion alone has no immediate effect on string fitness, but if the current population contains bad ordering, there is a high probability that crossover will destroy this schema. Inversion lowers the probability of destroying it. Some theoretic analysis of inversion is given in (Goldberg 1989). Inversion is not used very much nowadays.

2.1.1 Formal definition of GA

Let us now define a genetic algorithm more formaly in the following way:

Definition 2.1 *Suppose that the task is to find an optimal (maximal or minimal) solution to a certain problem $\mathcal{P}$. Let $\mathcal{T}$ be the domain of our problem and let $\mathcal{T}' \subseteq \mathcal{T}$ be its subset so that function $\chi : \mathcal{T}' \mapsto \mathcal{L}^l$ is 1–1 and onto for some fixed integer l and a finite language $\mathcal{L}$. The function χ maps (approximately) our original problem to the problem $\mathcal{P}'$ on $\mathcal{L}^l$.* Genetic algorithm *is defined as an iterative schema:*

$$X_{i+1} = \Phi(X_i), \quad X_0 \textit{ random} \tag{2.1}$$

with $X_i \subseteq \mathcal{L}^l$ and

$$\Phi = \mathcal{E} \circ \mathcal{R} \circ S$$

where $\mathcal{E}$, $\mathcal{R}$ and S are evaluation, recombination and selection operator respectively. Optimal solutions of problem $\mathcal{P}'$ can be mapped back into solutions of problem $\mathcal{P}$ using χ^{-1}.

A more general definition is given by definition 2.2 below.

2.2 Real–valued genetic algorithms

Originally, the concept of GAs was based on binary strings. Regardless of the problem, it would be mapped onto binary strings and then appropriate recombination operators are applied. However, the influence of Evolutionary Strategies (ES) (Rechenberg 1973, Schwefel 1977, Rechenberg 1994) grew stronger and the Genetic Algorithm for real function optimisation was developed. One of the real–valued GAs is the Breeder Genetic Algorithm (BGA) (Mühlenbein & Schlierkamp-Voosen 1993*b*, Mühlenbein & Schlierkamp-Voosen 1993*a*). It uses a vector of real numbers instead of bit strings, as a selection method it uses truncation selection and it utilises different crossover operators and mutations: intermediate crossover, fuzzy crossover, exponential mutation etc.

Formally, an evolutionary algorithm (that includes evolutionary strategies, genetic algorithms and evolutionary programming) could be defined in the following way (Bäck 1995*a*):

Definition 2.2 *An* Evolutionary algorithm *(EA) is defined as an 8–tuple:*

$$EA = (I, \Phi, \Omega, \Psi, s, \tau, \mu, \lambda) \tag{2.2}$$

where

- $I = A_x \times A_s$ *is the space of individuals where* A_x *and* A_s *are arbitrary sets;*
- $\Phi : I \mapsto \mathbf{R}$ *denotes a fitness function assigning real numbers to individuals;*
- $\Omega = \{w_{\theta_1}, \ldots, w_{\theta_z} \mid w_{\theta_i} : I^\lambda \mapsto I^\lambda\} \cup \{w_{\theta_0} : I^\mu \mapsto I^\lambda\}$ *is a set of probabilistic genetic operators* w_{θ_i} *each controlled by specific parameters summarised in the set* $\theta_i \subseteq \mathbf{R}$
- λ *is a natural number denoting number of offspring individuals;*
- μ *is a natural number denoting number of parent individuals;*
- $s_{\theta_s} : (I^\lambda \cup I^{\lambda+\mu}) \mapsto I^\mu$ *denotes the selection operator where* $\mu, \lambda \in \mathbf{N}$.
- $\tau : I^\mu \mapsto \{true, false\}$ *is the termination criterion for EA.*

The above formalisation covers $(\lambda + \mu)$ and (λ, μ) strategies.

Even more general, Schwefel & Bäck (1997, p. 7) give the definition of $(\mu, \kappa, \lambda, \rho)$ evolutionary strategy ($\kappa \geq 1$ is the life span of parents and ρ is the number of ancestors for each descendant) as an ordered 19–tuple!

2.3 Operators for the real–valued GA

The use of real–valued GAs for real function optimisation simplifies/resolves the problem of coding and decoding of genotypes and phenotypes. However, the recombination operator (crossover) and mutation have to be defined differently from operators described above or in say (Goldberg 1989). In the following the operators used are described.

2.3.1 Crossover operator

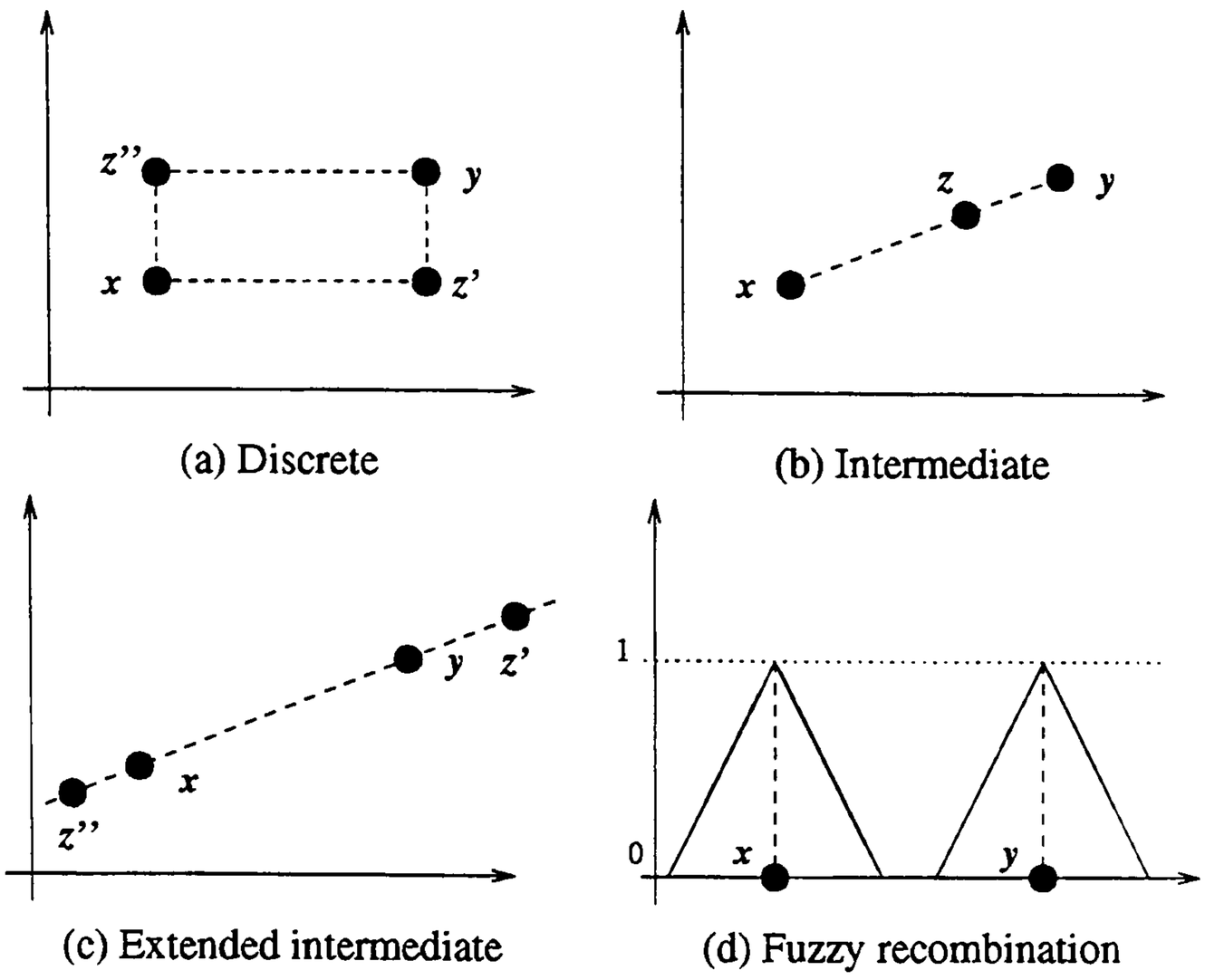
If $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$, there are several crossover operators for producing an offspring $z = (z_1, \ldots, z_n)$:

Discrete recombination (DR): In this case $z_i \in \{x_i, y_i\}$ for each $1 \leq i \leq n$. This corresponds to a standard uniform crossover in the binary case. Geometrically it is represented in Figure 2.2 (a). The probability of choosing allele x_i or y_i could also be specified, biasing choice more towards one of the parents.

Intermediate recombination (IR): In this case the offspring is given by

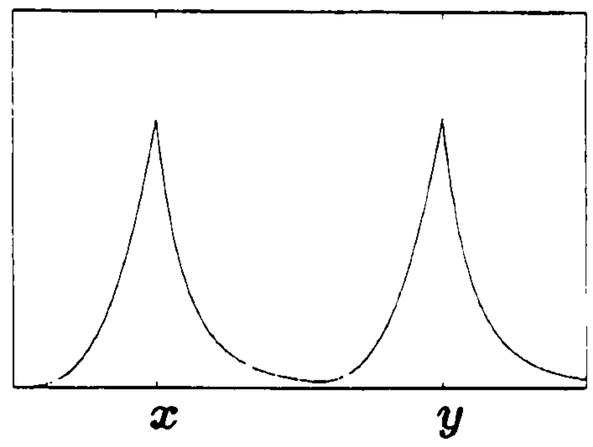
$$z_i = x_i + \alpha \cdot (y_i - x_i), \quad x_i \leq y_i, 0 \leq \alpha \leq 1$$

Here α could be fixed to $\alpha = 1/2$ or chosen randomly. Geometrically, it is represented in Figure 2.2 (b).



(a) Discrete (b) Intermediate

(c) Extended intermediate (d) Fuzzy recombination



(e) SBX

Figure 2.2. Different crossover operators.

Extended intermediate recombination (EIR): It is similar to IR except that $-d \leq \alpha \leq 1+d$ for some parameter d (Mühlenbein & Schlierkamp-Voosen 1993*b*). Geometrical interpretation is as in Figure 2.2 (c).

Fuzzy recombination (FR): In this case (Voigt, Mühlenbein & Cvetković 1995), the probability that the offspring has the value z_i is given by a bimodal distribution

$$p(z_i) \in \{\phi(x_i), \phi(y_i)\}$$

with triangular probability distribution $\psi(r)$ having the modal values x_i and y_i with

$$x_i - d \cdot |y_i - x_i| \leq r \leq x_i + d \cdot |y_i - x_i|$$

$$y_i - d \cdot |y_i - x_i| \leq r \leq y_i + d \cdot |y_i - x_i|$$

for $x_i \leq y_i$ and $d \geq 1/2$. Geometrically, it is represented in Figure 2.2 (d).

SBX: Similar to fuzzy recombination is simulated binary crossover SBX (Deb & Agrawal 1995, Deb & Kumar

1995) where the probability distribution used is presented in Figure 2.2(e). The children are computed using the following algorithm (Deb & Beyer 1999):

1. Generate a random number $u \in \mathcal{U}(0,1)$;

2. Compute

$$\beta_q = \begin{cases} (2u)^{\frac{1}{\eta+1}}, & \text{if } u \leq 0.5 \\ \left(\frac{1}{2(1-u)}\right)^{\frac{1}{\eta+1}}, & \text{otherwise.} \end{cases} \tag{2.3}$$

   Here is η some parameter.

3. Compute children $x_i^{(1,t+1)}$ and $x_i^{(2,t+1)}$ from parents $x_i^{(1,t)}$ and $x_i^{(2,t)}$ using the equations

$$x_i^{(1,t+1)} = 0.5 \cdot [(1+\beta_q)x_i^{(1,t)} + (1-\beta_q)x_i^{(2,t)}] \tag{2.4}$$

$$x_i^{(2,t+1)} = 0.5 \cdot [(1-\beta_q)x_i^{(1,t)} + (1+\beta_q)x_i^{(2,t)}] \tag{2.5}$$

If the variables are within bounded domains (i.e. $x_i^L \leq x_i \leq x_i^U$), the probability distributions need to be adjusted accordingly (Deb & Kumar 1995, p. 435).

2.3.2 Mutation

There are several mutation types that can be used. The common mutation type for the binary case where only one or two bits are flipped cannot be used here because the concept of complementary value is not defined. The following mutation types have been used:

Random mutation For each variable that is going to be mutated, choose a random value within its range and assign this value to the variable. So, every value is possible.

Gauß mutation This mutation is similar to the previous one, the only difference being that mutation step Δx_i is calculated according to Gauß' distribution $\mathcal{N}(0,1)$: smaller mutation steps are much more probable then large mutation steps. This is a standard Evolutionary strategies (Rechenberg 1973, Schwefel 1977, Bäck 1995*a*) mutation. Probability distribution is shown in Figure 2.3 (a).

EXP mutation This mutation type comes from the idea that the role of mutation at the beginning is to make large jumps whereas later on, as the search progresses it should be used more for fine–tuning so small jumps are more desirable. Exponential distribution is presented in Figure 2.3 (b). Here c is the constant that depends on the generation number.

2.3.3 Selection

Selection is the third major genetic operator that selects the best individuals for breeding. There are many different selection methods, and the best known (and used in the developed GA) are described below. In

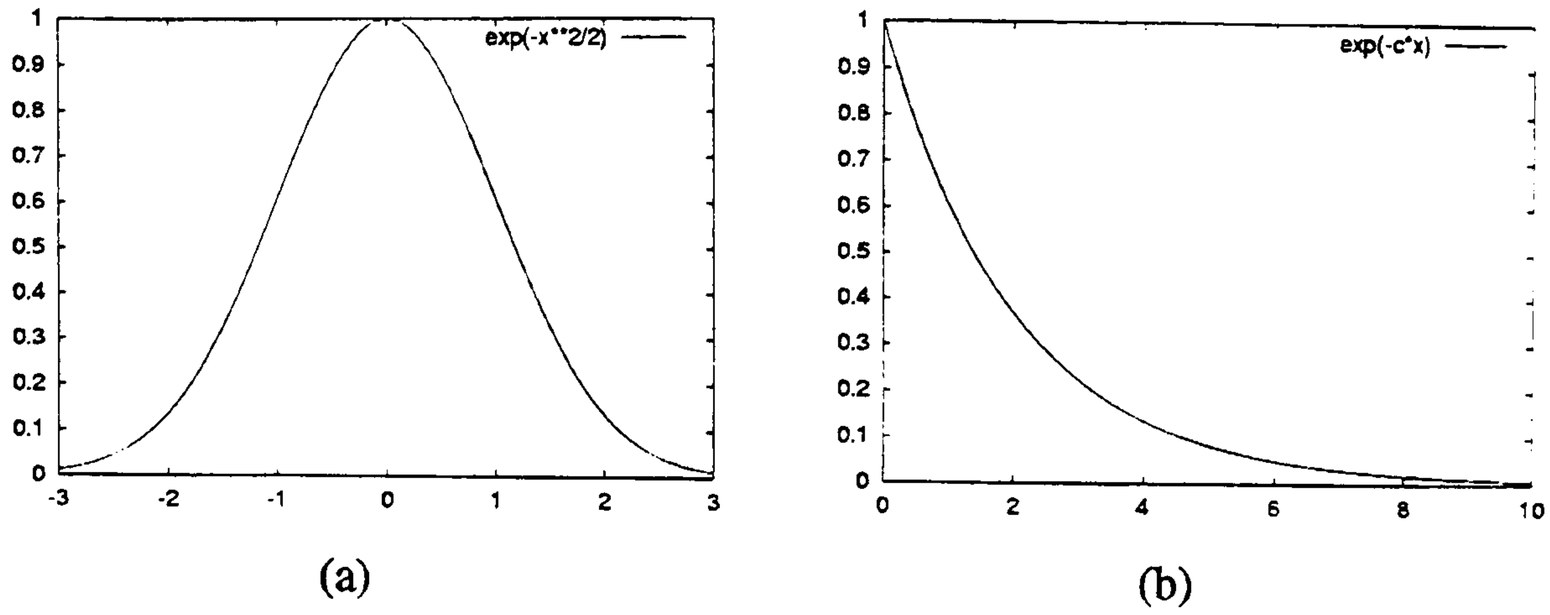


Figure 2.3. Probability distributions for (a) Gauß mutation and (b) EXP mutation.

the most general terms, selection is an operator that selects (according to some criteria) μ parents from λ individuals.

Proportional Selection (or Roulette Wheel) This method assigns probability of being chosen proportional to its fitness:

$$p(x) \sim \frac{f(x)}{\sum f(y)}$$

The main disadvantages of this method is that when the population settles down and individuals have similar fitness, the selection pressure decreases and then it doesn't work any better then random selection. Also, negative fitness values tend to confuse selection process.

Ranking Method This method was proposed by Baker (1985) to overcome the above mentioned weakness: The individuals in population are sorted according to their fitness, and the number of offsprings from a given individual is solely a function of its rank.

Tournament selection In this selection method, k individuals (with replacement) are randomly picked from the population (k–tournament) at a time, and the one with the best fitness is selected. The larger the tournament size, the higher the selection pressure. Mathematical analysis of tournament selection could be found in (Blickle & Thiele 1995) and generalised analysis of different selection methods in (Bäck 1995*b*).

Truncation selection In this selection method, $T \cdot 100\%$ best individuals are selected as parents to produce the next generation. This method, commonly used by breeders in biology, is used in Breeder Genetic Algorithm (BGA), (Mühlenbein & Schlierkamp-Voosen 1993*b*, Mühlenbein & Schlierkamp-Voosen 1993*c*), and its variant (μ, λ)–selection in evolutionary strategies (Bäck & Schwefel 1993).

2.4 Other evolutionary methods

Of course, genetic algorithms are not the only stochastic method for optimisation. Other methods include:

- Evolutionary strategies (Rechenberg 1973, Schwefel 1977, Rechenberg 1994);

- Evolutionary Programming (Fogel 1962, Fogel 1964, Fogel, Owens & Walsh 1966);
- Simulated Annealing (Laarhoven 1988, Laarhoven & Aarts 1987, Ingber 1989, Ingber 1993);
- Scatter search (Glover 1998, Laguna in press, Glover 1999);
- Tabu search (Glover & Laguna 1997, Glover & Laguna in press);
- Differential evolution (Storn & Price 1995, Price 1999);
- Random walk;
- etc.

However, there is no universally best method for optimisation i.e. there is no method that will *in all cases* outperform other methods. The so called "no free lunch" (NFL) theorem (Wolpert & Macready 1997) proves the following result:

Theorem 2.1 (No free lunch theorem) *Let us denote by* $P(d_m^y \mid f, m, \alpha)$ *the conditional probability of obtaining a particular sample* d_m^y *(of costs of points visited by the algorithm sorted in time) with* m *iterations of algorithm* α *on a given cost function* f*. Then, for any pair of algorithms* α_1, α_2*:*

$$\sum_f P(d_m^y \mid f, m, \alpha_1) = \sum_f P(d_m^y \mid f, m, \alpha_2)$$

The simple corollary of the NFL theorem is that, averaged on the class of all functions, no algorithm performs better than (say) random search. On the other hand, the result of NFL theorem is "too general" and mostly theoretically important: we do not need an algorithm which will outperform all other algorithms on *all* functions. Desired is a better performance on a certain class of functions. In order to achieve this, careful choice of operators and parameters is needed. Chapter 4 presents our choice of GA tailored particularly for the BAe design problem and other real–valued multi–objective optimisation problems.

CHAPTER 3

Multi–Objective Optimisation

The genetic algorithm as presented in Chapter 2 is mostly suitable for one–dimensional functions, i.e. for functions with single output. However, there are some extensions for multi–objective functions. In this chapter several different GA–based multi–objective optimisation methods are presented and their advantages and disadvantages are discussed. At the end of the chapter, a new weighted Pareto method is described. It combines the best of both Pareto and weighted sum methods

Definition 3.1 Let $n > 0$, $k > 0$, $\mathcal{D} = \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_n \subseteq \mathbf{R}^n$, and $\mathcal{R} = \mathcal{Y}_1 \times \mathcal{Y}_2 \times \ldots \times \mathcal{Y}_k \subseteq \mathbf{R}^k$. Let further $f_i : \mathcal{D} \mapsto \mathcal{Y}_i$ for $1 \leq i \leq k$ and finally $\mathbf{F} : \mathcal{D} \mapsto \mathcal{R}$, so that $\mathbf{F}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_k(\boldsymbol{x}))$.

Our goal is to optimise function $\mathbf{F}(\boldsymbol{x})$ under constraints i.e.

$$\max_{\boldsymbol{x}} \mathbf{F}(\boldsymbol{x}) \tag{3.1}$$

$$g_1(\boldsymbol{x}, \boldsymbol{p}) \leq 0, \ldots, g_l(\boldsymbol{x}, \boldsymbol{p}) \leq 0 \tag{3.2}$$

$$h_1(\boldsymbol{x}, \boldsymbol{p}) = 0, \ldots, h_m(\boldsymbol{x}, \boldsymbol{p}) = 0 \tag{3.3}$$

where $p = (p_1, \ldots, p_u)$ are additional (real–valued) parameters.

This problem is well known and a number of 'classical' (i.e. non–genetic) (Hwang & Masud 1979, Osyczka 1984) and genetic algorithm approaches exist (Schaffer 1985, Horn & Nafpliotis 1993, Fonseca & Fleming 1995). Good surveys of evolutionary algorithm methods include (Coello Coello 1999, Veldhuizen & Lamont 1998, Veldhuizen 1999). Comparison of several evolutionary approaches are given in (Zitzler & Thiele 1999, Zitzler, Deb & Thiele 2000, Zitzler 1999).

Some of the methods for multi–objective optimisation are presented in the following sections. The most important ones from the aspect of conceptual engineering design were also presented and discussed in (Cvetković et al. 1998).

3.1 Difference between single–objective and multi–objective optimisation

Comparing single–objective optimisation with multi–objective optimisation, an additional problem in the case of multi–objective optimisation is: How to define an optimum? Ideally point $\boldsymbol{x}^* = (x_1^*, \ldots, x_n^*)$ is a *maximum* of the function $\mathbf{F}(\boldsymbol{x})$ if

$$(\forall \boldsymbol{x} = (x_1, \ldots, x_n) \in \mathcal{D})(f_1(\boldsymbol{x}) \leq f_1(\boldsymbol{x}^*) \wedge \ldots \wedge f_k(\boldsymbol{x}) \leq f_k(\boldsymbol{x}^*)) \tag{3.4}$$

However, unless the problem is very easy, usually there is no such ideal point. For those problems with property (3.4), it is enough to optimise one of f_j e.g.

$$\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}} f_1(\boldsymbol{x})$$

The next two subsections give a few definitions needed for defining orderings and discuss multi–dimensional orders.

3.1.1 Partial and total order

In the further text the following definitions are needed (Grätzer 1971, Grätzer 1978).

Definition 3.2 (Definition of order)

- *Binary relation R is a* partial order *on domain $\mathcal{D}$ if and only if it satisfies the following three properties:*

  reflexivity: *For all $x \in \mathcal{D}$, $R(x,x)$;*

  antisymmetry: *For all $x, y \in D$, if $R(x,y)$ and $R(y,x)$ then $x = y$;*

  transitivity: *For all $x, y, z \in \mathcal{D}$, if $R(x,y)$ and $R(y,z)$, then $R(x,z)$.*

- *Binary relation R' is a* total order *on domain $\mathcal{D}$ if*

  1. *It is a partial order, and*
  2. *For all $x, y \in \mathcal{D}$, $R'(x,y)$ or $R'(y,x)$;*

- *Binary relation R'' is a* strict (partial) order *on domain $\mathcal{D}$ if it satisfies the following two properties:*

  irreflexivity *For all $x \in \mathcal{D}$, $\neg R''(x,x)$;*

  transitivity *For all $x, y, z \in \mathcal{D}$, if $R''(x,y)$ and $R''(y,z)$, then $R''(x,z)$.*

The infix notation xRy instead of $R(x,y)$ is more usual for binary relations, and $\leq$ or $\geq$ are commonly used for partial and $>$ or $<$ for strict orders.

Definition 3.3 (Definition of chain) *Subset $\mathcal{D}' \subseteq \mathcal{D}$ is called* chain *with respect to partial order $\leq$ if every two elements of $\mathcal{D}'$ are comparable i.e. for all $x, y \in \mathcal{D}'$, $x \leq y$ or $y \leq x$.*

Example 3.1

1. The usual order on set of real numbers is total order: it is always possible to say for any two real numbers x and y if $x \leq y$ or $y \leq x$;

2. If $\mathcal{D} = \{(x,y) \mid x,y \in \mathbf{R}\}$ and $(x,y) \leq_2 (x_1,y_1) \overset{\text{def}}{\Leftrightarrow} x \leq x_1 \wedge y \leq y_1$, then

   - Order $\leq_2$ is (non–total) partial order on $\mathcal{D}$ since for example $(2,3) \not\leq_2 (3,2)$ and $(3,2) \not\leq_2 (2,3)$;
   - Sets $\{(x,0) \mid x \in \mathbf{R}\}$ and $\{(0,x) \mid x \in \mathbf{R}\}$ are examples of chains according to $\leq_2$.

This is exactly the problem in multi–objective optimisation. Multi–dimensional component–wise order relation $\leq$ is not a *total order*, i.e., it is not fulfilled that for every two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbf{R}^n$, $\boldsymbol{x} \leq \boldsymbol{y}$ or $\boldsymbol{y} \leq \boldsymbol{x}$ i.e. not every two elements are comparable. In multi–dimensional case, this ordering relation is a *partial order*. Instead of one total order, we have (possibly many) chains where every two elements within a chain are comparable. The greatest element of a chain is called a *maximal element*. Figure 3.1 gives an example.

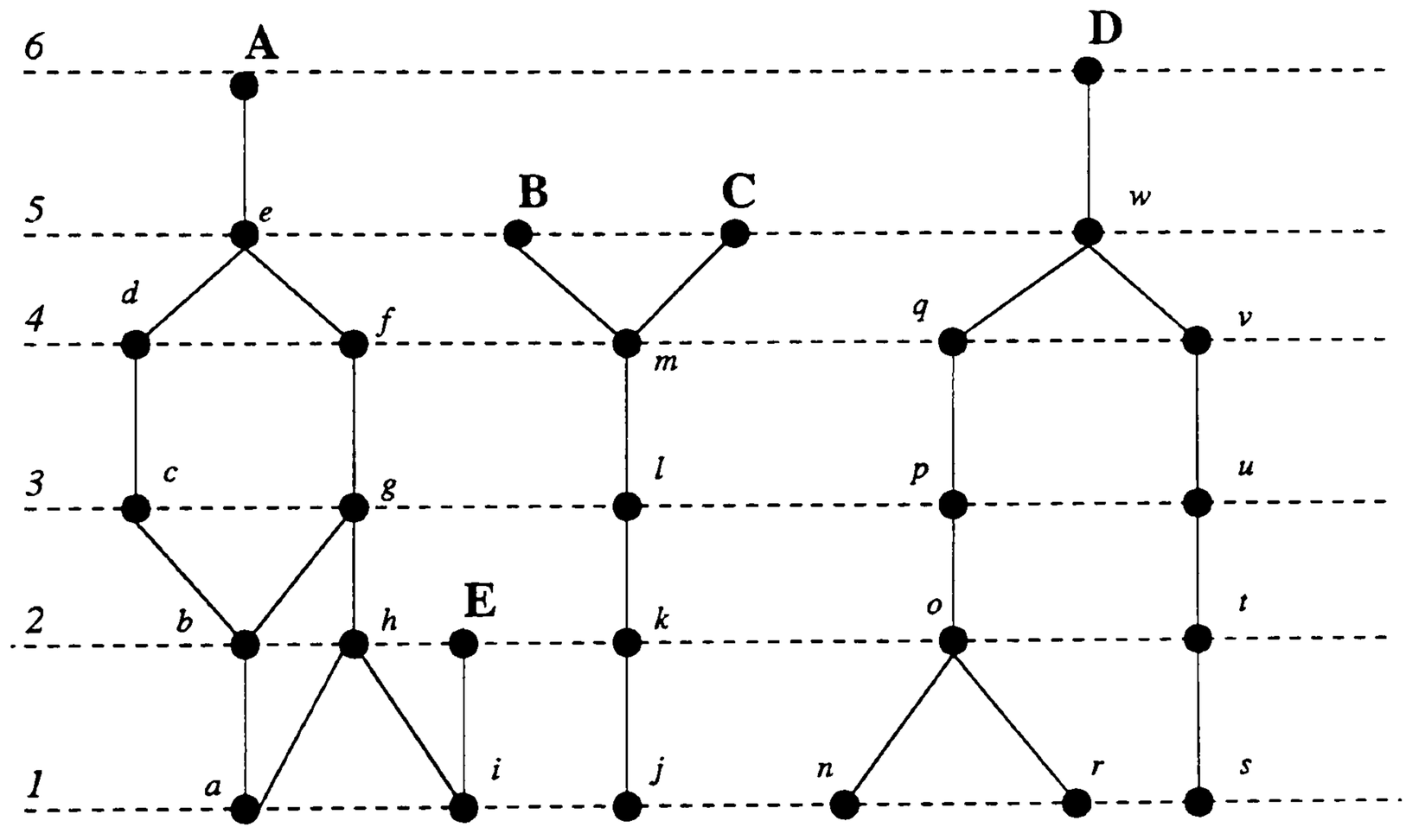


Figure 3.1. Example of partial order. Points A, B, C, D and E are maximal elements of the order.

Figure 3.1 also gives example of chains: set $\{a,b,c,d,e,A\}$ is one chain, $\{a,b,g,f,e,A\}$ is another, $\{i,E\}$ is also a chain etc. Within a chain, the order of elements is known, but it is not possible to say that $i \leq a$ or $a \leq i$. They are *incomparable*. Points A, B, C, D and E are maximal elements of the order in Figure 3.1.

3.1.2 Concept of optimum in multi–objective optimisation

Using characterisation (3.4) as a definition of a maximum of a multi–objective function, in most of the cases there wouldn't be any maximum at all. A classical example is the Schaffer's function (1984) presented at

Figure 3.2:

$$F(x) = (f_1(x), f_2(x)) = (-x^2, -(x-2)^2) \tag{3.5}$$

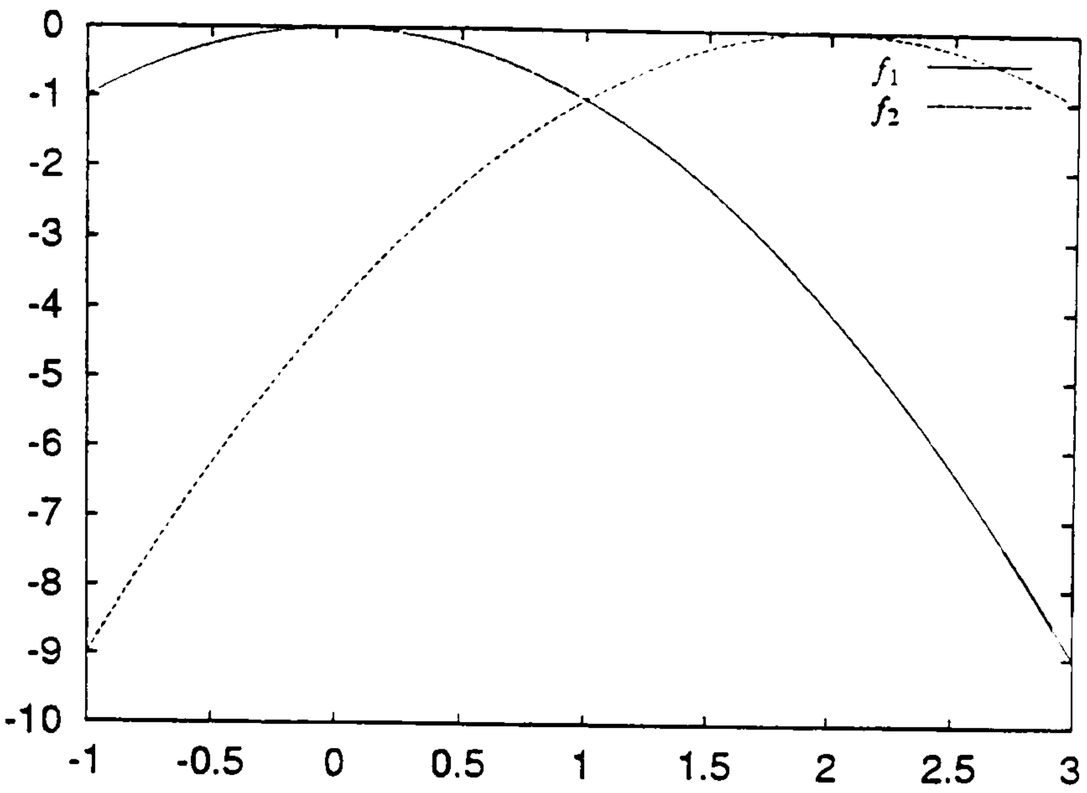


Figure 3.2. Plot of the function (3.5).

Obviously, the first output (function f_1) is at maximum for $x = 0$, whereas the second (function f_2) is at a maximum for $x = 2$, so there is no single point satisfying the condition (3.4).

Therefore, instead of maxima in the sense of (3.4), it can only be talked about maximal elements of chains and there can be many chains. The concept of *non–dominance* becomes very important. It will be defined it and used later with Pareto method in section 3.2.3.

3.2 Multi–objective optimisation methods

In the following text, a short survey of several multi–objective optimisation methods is given with pro– and counter–arguments of each method.

3.2.1 Scalarisation method — Weighted sum

Weighted sum is a method of scalarisation of vector functions (Hwang & Masud 1979, p.32), (Osyczka 1984).

Definition 3.4 For a function $\mathbf{F}(x) = (f_1(x), \ldots, f_k(x))$ from Definition 3.1 and a vector $w = (w_1, \ldots, w_k)$, so that $\sum_{i=1}^{k} w_i = 1$, define

$$\mathbf{F}_w(x) = \sum_{i=1}^{k} w_i \cdot f_i(x) \tag{3.6}$$

Instead of function $\mathbf{F}(x)$, function $\mathbf{F}_w(x)$ is optimised for a suitable vector w, using single–objective optimisation methods.

Usually one assigns weights according to the importance of objectives: the more important objective will get a higher weight, less important objectives will get lower weights. However, since not all the objectives have

the same range of values, they must be normalised.

One method of normalisation is the calculation of $w_i^* = 1/\max_x f_i(x)$. The vector $(\max_x f_i(x))_{i=1,k}$ is called the *positive ideal solution* (Gen & Cheng 1997, p. 90). The other method would be to keep track of the maximal values found so far and dynamically update the normalisation factors. In this case separate runs for optimising each objective are not needed.

The normalisation techniques described assume objective maximisation. If an objective y is being minimised, there are two approaches:

1. Use identity $\min y = -\max(-y)$. However, the genetic algorithm has a problem with negative fitness values[1] — that problem can be avoided by maximising $M - \max(-y)$ for a large enough M. More precisely, if objective y is positive and bounded i.e. $y \in [m, M]$, then $0 \leq (M-y)/(M-m) \leq 1$ and value $1/(M-m)$ can be used for the weight.

2. Assuming that $y > 0$, use identity $\min y = 1/\max(1/y)$ and maximise $1/y$. In this case there are no problems estimating weights, but it can not be always known in advance if the objective y is non–zero in all cases.

After normalisation, the relative importance of each objective $c_i \in [0, 1]$ can be specified and the final weight assigned as $w_i = c_i \cdot w_i^*$.

Advantages using weights:

- Multi–objective function is reduced to a single–objective function;
- Traditional optimisation methods can be used.

Disadvantages using weights:

- Problem how to set weights — results are sensitive to weights ratio;
- Objectives must be normalised — computationally expensive in general;

However, weighted sum based GA optimisation methods, although very useful for multi–objective optimisation in general, are generally not suitable for optimisation during conceptual design. The main reason is that in the conceptual design phase the likelihood of objective and constraint variation is high. Thus the fitness landscape will change therefore necessitating the re–calculation and normalisation of weights many times (Cvetković et al. 1998, p. 259).

The general problem that this method suffers from is the specification of weights. Table 3.1 illustrates this problem: it shows how the change of weights (w_4 for objective y_4 and w_9 for objective y_9 influence their optimal values. In this table only two objectives are considered but in some cases 20 or more objectives will be considered at the same time. Those points, together with the interpolated Pareto front are also presented in Figure 3.3

w_4	w_9	y_4	y_9
0.0	1.0	0.00	7901.88
0.1	0.9	33.02	7513.89
0.15	0.85	55.50	7134.80
0.2	0.8	63.66	6914.39
0.25	0.75	68.79	6731.05
0.3	0.7	72.39	6562.71
0.4	0.6	73.58	6481.46
0.5	0.5	76.16	6198.47
0.6	0.4	81.10	5545.35
0.7	0.3	81.75	5400.89
0.8	0.2	82.77	5031.09
0.9	0.1	82.87	4959.28
1.0	0.0	82.90	3251.60

Table 3.1. Influence of weight on BAe function

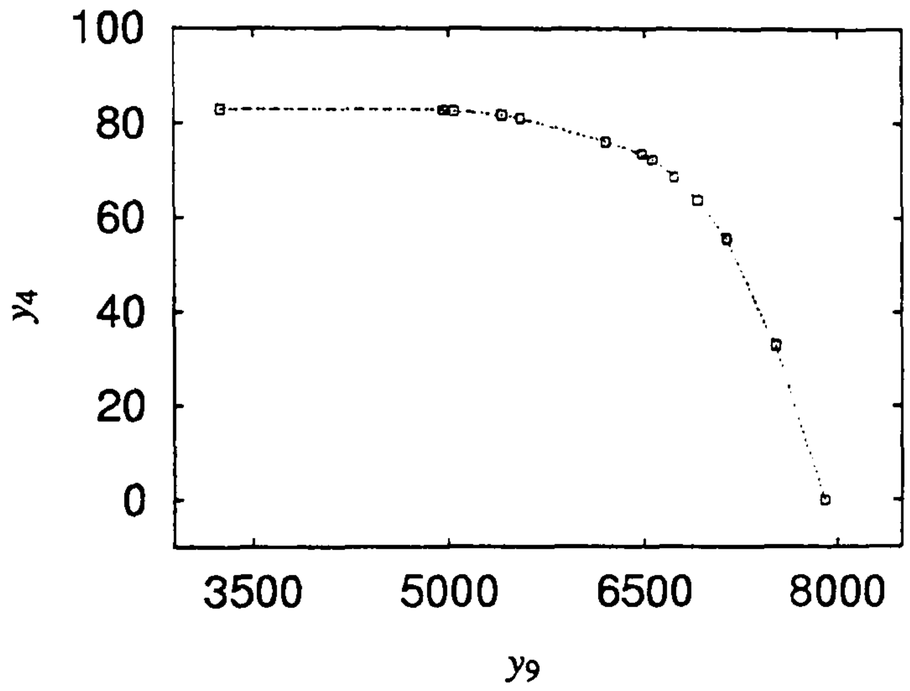


Figure 3.3. Influence of weights on BAe function

The weights specification problem could be overcome by randomly changing weights in each generation (Murata, Ishibuchi & Gen 1998) but the advantages of that method are not quite clear.

3.2.2 Lexicographic ordering

Lexicographic ordering is a commonly used sorting method of names e.g. in telephone directories. This method enable us to define a *total* order on $\bigcup_{i>0} \mathbf{R}^i$:

Definition 3.5 (Lexicographic Order) *The point* $(x_1, x_2, \ldots, x_k) \in \mathbf{R}^k$ *is* less in lexicographic order *then the point* $(y_1, y_2, \ldots, y_n) \in \mathbf{R}^n$, *where* $k \leq n$, *written* $(x_1, x_2, \ldots, x_k) \prec_L (y_1, y_2, \ldots, y_n)$ *if*

1. $x_1 < y_1$, *or*
2. $x_1 = y_1$ *and* $x_2 < y_2$, *or*
3. $x_1 = y_1$, $x_2 = y_2$ *and* $x_3 < y_3$, *or*
4. *etc.*

[1] at least simple GA or any other GA that uses proportional selection

5. $x_1 = y_1, \ldots, x_{k_1} = y_{k-1}, x_k < y_k$, *or*

6. $x_1 = y_1, \ldots, x_{k_1} = y_{k-1}, x_k = y_k$ *and* $k < n$.

Example 3.2 *For order* $\prec_L$ *the following holds true:*

$$(2,3) \prec_L (3,1)$$

$$(2,2) \prec_L (2,3)$$

$$(3,4) \prec_L (3,4,1)$$

The use of lexicographic order has been described, among others, in (Ben-Tal 1979) and (Coello Coello 1996).

Advantages of using lexicographic ordering:

- Lexicographic order is a total order — there is only one maximum according to this order;
- It is easy to implement — sequential optimisation: optimise first the most important objective, then from the set that satisfy the first objective optimise on second etc;

Disadvantages of using lexicographic ordering:

- Importance of each objective is crucial;
- Relative importance of each objective must be known from the very beginning;
- No compromising possible.

However, the disadvantage of having to know order of importance in advance can be overcome using random lexicographic order i.e. changing the order every couple of generations in a GA. This ensures that on average each objective gets an equal share in optimisation.

Strictly speaking, a genetic algorithm combined with lexicographic ordering applies this method for population sorting only and does not use it for direct optimisation of objectives. Some form of total ordering of the population is needed for selection procedures that expect ordered populations (e.g. tournament, truncation etc.).

3.2.3 Pareto based methods

Let us consider the vector function $\mathbf{F} : \mathcal{D} \mapsto \mathcal{R}$ from Definition 3.1, page 18.

Definition 3.6 (Pareto) *The point* $x \in \mathcal{D}$ Pareto–dominates *a point* $y \in \mathcal{D}$ *with respect to the function* $\mathbf{F}$, *denoted* $y \preceq_P^{\mathbf{F}} x$, *if* $\wedge_{i=1}^{k}(f_i(y) \leq f_i(x))$ *and at least one of inequalities is strict. In the following text the superscript* $^{\mathbf{F}}$ *will be omitted.*

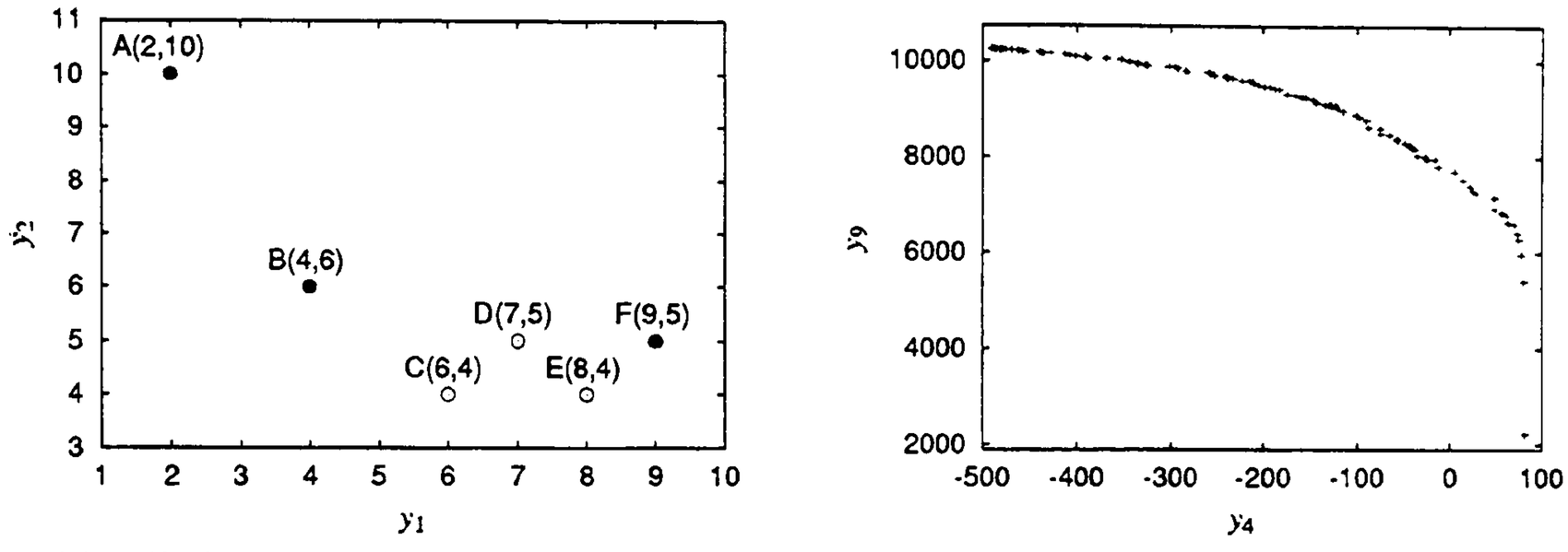
The point $x_P \in \mathcal{D}$ *is* Pareto–optimal *or* non–dominated *(for a given function* $\mathbf{F}$*) if there is no point* $y \in \mathcal{D}$ *that Pareto–dominates* x *i.e.* $(\neg\exists y \in \mathcal{D})(x_P \preceq_P y)$.

Set $\mathcal{F} \subseteq \mathcal{D}$ *is called* Pareto front *with respect to the function* $\mathbf{F}$ *if every element* $x \in \mathcal{F}$ *is Pareto optimal with respect to the function* $\mathbf{F}$. *In other words, Pareto front is a maximal set of non–dominated elements.*

Example 3.3

(a) For a set of points $A(2,10), B(4,6), C(6,4), D(7,5), E(8,4)$ and $F(9,5)$, presented in Figure 3.4(a), Pareto front is $\{A, B, F\}$.

(b) Figure 3.4(b) shows Pareto front for one 'real–world' (BAe) problem.



(a) Filled points belong to Pareto front

(b) real–world problem

Figure 3.4. Pareto front examples.

Pareto is discussed more fully in section 3.5, page 33 and on, where some generalisations are also introduced.

3.2.4 Vector evaluated genetic algorithm (VEGA)

Schaffer (1984) was the first to explore sub–populations based GA methods for multi–objective optimisation. In his thesis, he developed VEGA (Vector Evaluated GA). The only required change from the simple GA is a selection step.

Definition 3.7 *Let* $\mathcal{D}$ *be a real–valued, multi–dimensional domain and* $\mathbf{F}(x) = (f_1(x), \ldots, f_k(x))$ *the function with domain* $\mathcal{D}$. *The following operators are defined:*

$$\begin{aligned} &\text{CROSS} : \mathcal{D}^2 \mapsto \mathcal{D} && \text{– crossover} \\ &\text{MUT} : \mathcal{D} \mapsto \mathcal{D} && \text{– mutation} \\ &\text{SEL}_f : \mathcal{D}^m \mapsto \mathcal{D}^m && \text{– selection according to objective } f \end{aligned}$$

Let $P(t) = P_1 \cup \ldots \cup P_k$ *be the population in generation* t. *Then* VEGA *is defined as an iterative process*

$$P(t+1) = \text{MUT}(\text{CROSS}(\cup_{i=1}^{k} \text{SEL}_{f_i}(P_i(t)))^2)$$

VEGA is illustrated in Figure 3.5: the population is divided into k subpopulations (k being the number of objectives) and in the selection step, parents are chosen from each subpopulation only according to the relevant objective. After the recombination/mutation step performed on the whole population, the population is split randomly into subpopulations.

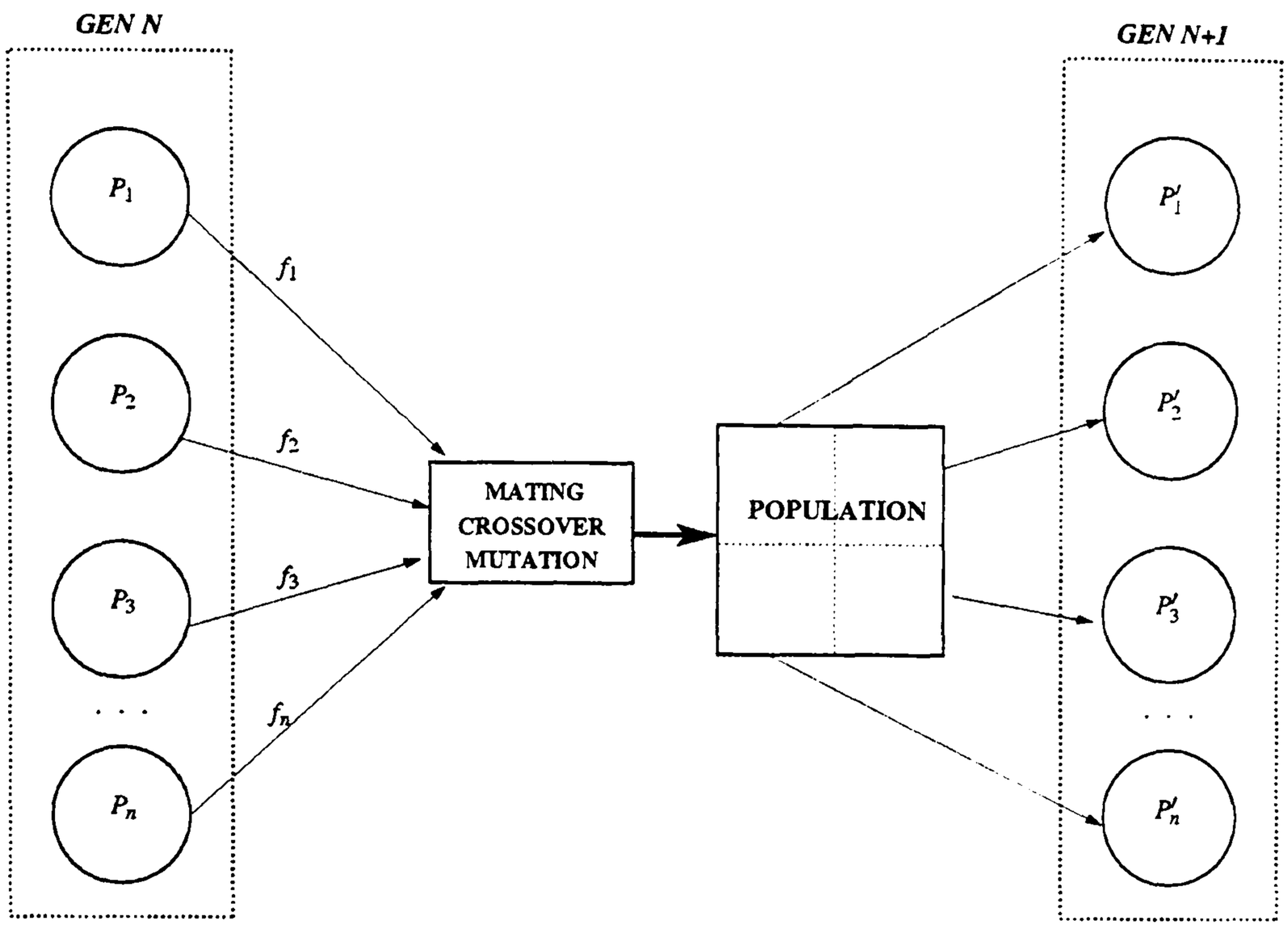


Figure 3.5. Basic VEGA algorithm

Advantages of VEGA:

- Easy to integrate within a GA as only minor modifications are needed;
- Can be easily scaled with the number of objectives;
- Computationally not more expensive then the single–objective GA;

Disadvantages of VEGA:

- Tends to average solutions very quickly;
- No concept of compromising;
- Equivalent to optimising linear combination of objectives with weights changing from generation to generation (Richardson, Palmer, Liepins & Hilliard 1989).

3.2.5 Variants of VEGA

Fourman (1985) applied a similar method to VEGA, but without subpopulations: in the selection step, tournament was applied where the individuals were compared according to a randomly chosen objective.

Keeping subpopulations separate: There exists two variants (see also (Parmee, Johnson & Burt 1994)):

- Subpopulations are not mixed at all, or every t_{mix} generations;
- The n_b best individuals are copied to every subpopulation;

These variants of VEGA have been applied in (Cvetković et al. 1998).

3.3 BAe function

CAPS (Computer Aided Project Studies) is an integrated computer software suite, developed at and by British Aerospace (BAe) for use by engineers and designers during the earliest investigation stages of a new aircraft project. Our project utilises their CAPS system.

The scope of disciplines covered by CAPS is wide: preliminary geometric definition, aerodynamic analysis, mass estimation, performance analysis, cost estimation etc.

In a typical job CAPS is programmed to search for design solutions that meet performance requirements, whilst satisfying a number of constraints.

BAe has developed a miniCAPS model (Webb 1997) based upon the full CAPS representation for use by the Plymouth Engineering Design Centre (PEDC). Initial development was in FORTRAN 77 but it has been ported to C++ and further developed in the PEDC by Dr Andrew Watson, the project research fellow.

This section describes the BAe function (mathematical model of miniCAPS), also described in (Cvetković et al. 1998) and in more details in (Webb 1997). This function is still being developed but at the moment there are $n = 9$ inputs (variables) and $k = 13$ outputs (objectives). For this research, the function is considered a black box and is presented in Figure 3.6. All the optimisation methods that have been described in this chapter will be applied to the BAe function.

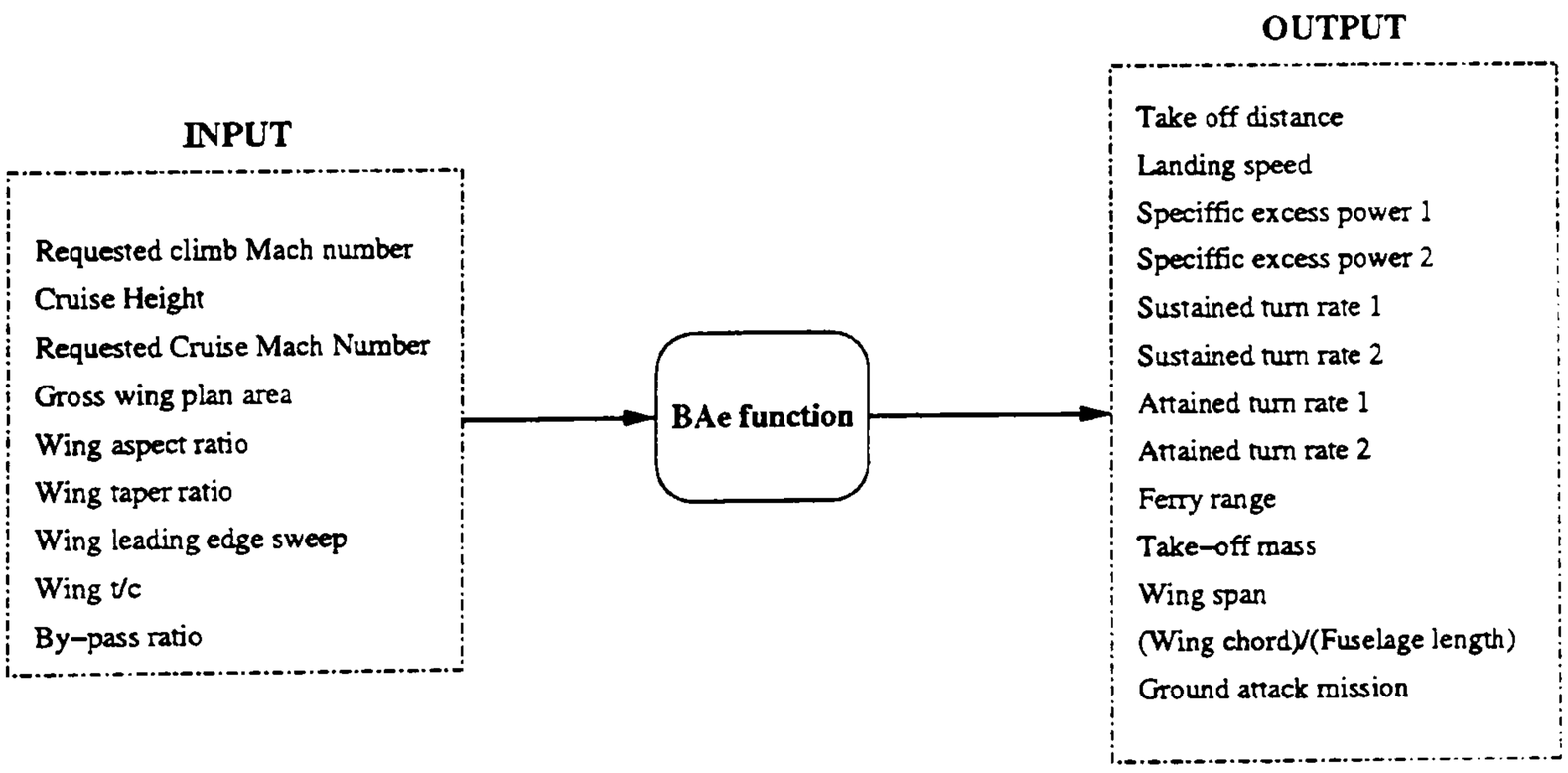


Figure 3.6. Schematic presentation of British Aerospace (BAe) function

3.3.1 Interaction among variables

Since it is very hard or even impossible to visualise functions with more then 3 variables, the visualisation of the BAe function is also not possible. Nevertheless, in order to gain some intuition, some 3D plots of a pair of variables vs. ferry range (FR) are presented in Figure 3.7

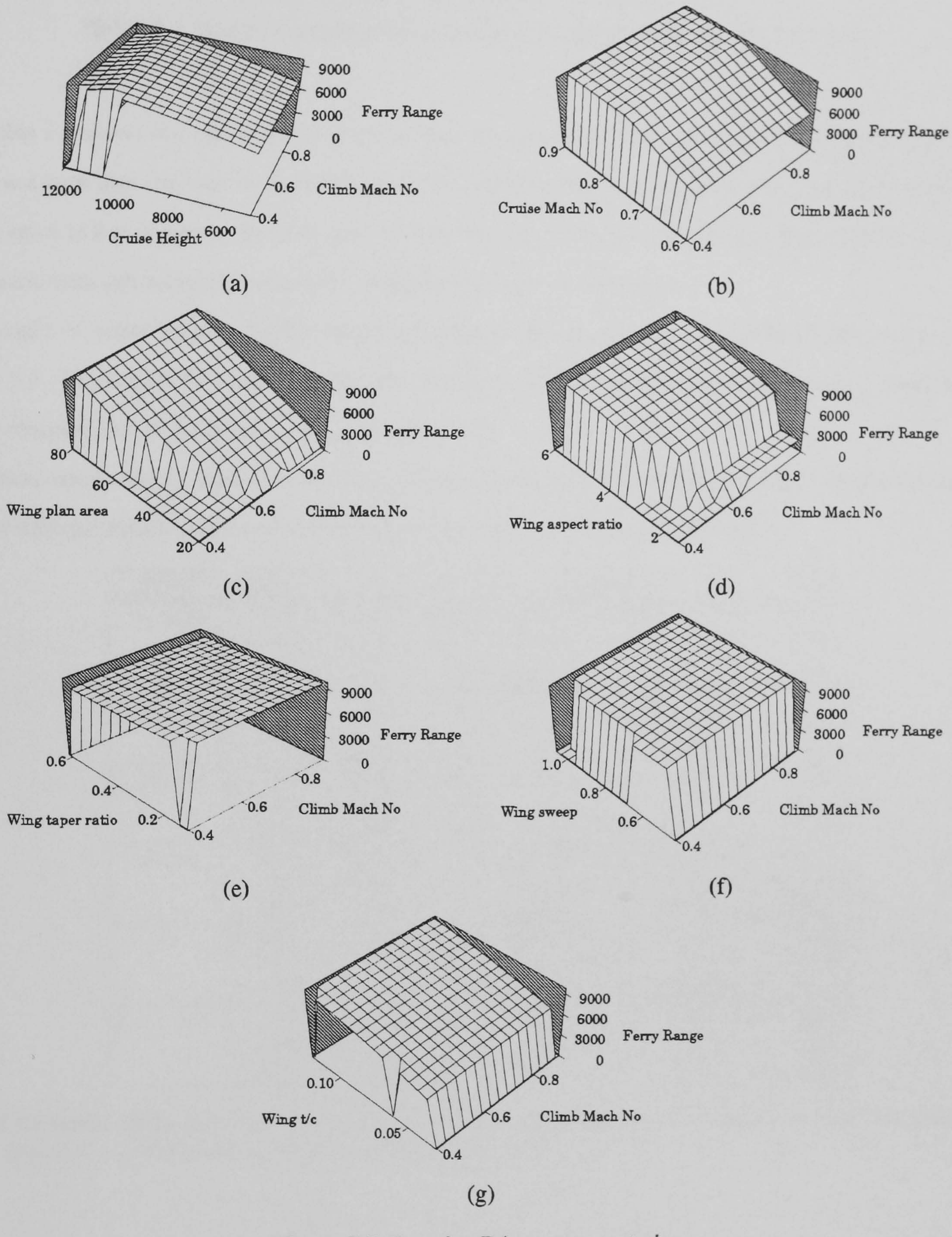


Figure 3.7. Function BAe component vise.

Optimisation conflicts

As already mentioned, there are 13 objectives to optimise, some of them conflict greatly. For instance, optimising y_3, y_4 and y_9, gives the situation[2] as presented in Table 3.2 (one run on GA with Pareto sorting).

[2]If optimising on y, an additional constraint $y \geq 0.001$ is used.

Optimise	y_3	y_4	y_9
y_3	148.1757	76.9852	0.0000
y_4	145.2489	82.9057	0.0000
y_9	-1.7305	-492.6476	10263.6426
y_3, y_4	145.3017	82.8891	2310.7244
y_3, y_9	23.2492	-380.7826	10096.2207
y_4, y_9	115.2269	0.0001	7886.8315
y_3, y_4, y_9	115.7906	0.0547	7873.7720

Table 3.2. Optimising different combinations of objectives gives different results

Table 3.2 shows that optimising y_4 (specific excess power (SEP) for supersonic case) affects ferry range (FR) y_9 a great deal and vice versa. Maximising FR result in a very bad values for SEP, maximising SEP, ferry range equal to 0 is obtained. Since the goal of the project is the design of an aircraft that performs well in all situations, both sub-sonic and supersonic configurations must be considered.

Results of using weights and the weight assignment method using positive ideal solution, are shown in Table 3.3. Because of the interaction between objectives, additional constraints were needed requiring that every objective that is optimised to be greater then 10^{-3}.

Those results clearly show that choosing different optimisation methods and different objectives can give totally different results that are, globally, not necessary better or worse, just different.

What?	How?	y_3	$\sigma(y_3)$	y_4	$\sigma(y_4)$	y_9	$\sigma(y_9)$
y_3, y_4, y_9	PAR	119.72	4.27	0.68	3.65	7747.15	134.69
	LEX	115.61	5.17	0.42	1.33	7826.03	100.81
	WEI	146.84	ε	76.93	ε	6070.39	0.64
y_3, y_9	PAR	5.62	6.27	-459.52	28.27	10220.1	43.41
	LEX	0.34	0.64	-482.91	2.72	10249.8	10.47
	WEI	139.56	0.08	52.26	0.19	6959.5	5.93
y_4, y_9	PAR	117.33	4.41	0.29	1.19	7811.8	106.24
	LEX	113.68	5.35	0.17	0.80	7846.14	84.12
	WEI	145.72	ε	76.16	0.002	6198.94	0.29
y_3, y_4	PAR	145.30	ε	82.89	ε	1838.35	1734.38
	LEX	145.30	ε	82.89	ε	1527.58	1641.24
	WEI	145.29	ε	82.90	ε	1637.69	1686.49
y_9	PAR	-1.74	0.02	-491.83	1.25	10262.8	0.96
	LEX	-1.74	0.01	-491.98	0.79	10263.4	0.45
	WEI	-1.74	0.01	-492.19	0.81	10263.4	0.31
$y_1, \ldots, y_9$	PAR	59.67	ε	9.81	ε	4787.75	0.02
	LEX	59.67	ε	9.81	ε	4787.73	0.03
	WEI	114.66	0.16	37.19	0.21	5769.41	15.22

Table 3.3 Pareto (PAR), random lexicographic (LEX) and weighted sum (WEI) optimisations, average over 50 runs. Here $0 < \varepsilon < 0.005$ and $\sigma(y)$ is the standard deviation of y.

Some results of VEGA and its variants, and a comparison with the Pareto method are presented in Table 3.4. All results are averaged over 200 runs. All GA parameters are the same in all runs except for population size in the case of Schaffer's VEGA. Uniform crossover with probability 1 and exponential mutation with probability 1/9 have been used, applied on real–valued chromosomes.

method	y_3	$\sigma(y_3)$	y_4	$\sigma(y_4)$	y_9	$\sigma(y_9)$	$\sum y_i$
Fourman	115.30	8.37	12.23	10.90	7593.0	143.6	7720.48
Pareto	120.07	4.57	0.44	2.41	7731.7	145.3	7852.17
Schaffer	145.67	4.55	71.8	13.0	5886.4	230.3	6103.93
Subpop (mix)	116.76	8.90	20.7	13.6	7355.5	159.8	7492.96
Subpop (copy)	114.94	11.14	20.3	16.0	7116.1	260.0	7251.32

Table 3.4. Results on maximising y_3, y_4 and y_9 using different optimisation methods.

3.4 Pareto ranking

Instead of just a dominant/non-dominant scheme for Pareto sorting that doesn't distinguish elements very well (in some of our runs, at the end of the run more then 80% are non–dominated), a finer method can be used: Pareto ranking (compare (Fonseca & Fleming 1995, Srinivas & Deb 1995, Valenzuela-Rendón & Uresti-Chare 1997, Deb, Agrawal, Pratap & Meyarivan 2000)).

Our ranking procedure is performed in the following way:

Definition 3.8 *Pareto rank* r_1 in a set $\mathbf{X} = \{x_1, \ldots, x_n\}$ is assigned in the following way:

$(\forall x \in \mathbf{X}) r_1(x) \leftarrow 0$

$(\forall x \in \{x_1, \ldots, x_n\})(\forall y \in \{x_1, \ldots, x_n\} \setminus \{x\})$ // sequentially!

If $(r_1(x) = r_1(y) \wedge x > y)\, r_1(x) \leftarrow r_1(x) + 1$

If $(r_1(x) = r_1(y) \wedge x < y)\, r_1(y) \leftarrow r_1(y) + 1$

Elements with the highest rank are considered the best. Alternative Pareto ranking r_2 (Srinivas & Deb 1995, Deb et al. 2000) uses the following top–down approach:

$m \leftarrow 1$

$\mathbf{X}' = \mathbf{X}$

Repeat

$m \leftarrow m - 1$;

$(\forall x \in \mathbf{X}')$ if x is non–dominated

$r_2(x) \leftarrow m$; $\mathbf{X}' = \mathbf{X}' \setminus \{x\}$

Until $\mathbf{X}' = \emptyset$

$(\forall x \in \mathbf{X})\ r_2(x) \leftarrow r_2(x) + |m|$.

In both ranking algorithms non–dominance can be tested using the whole population (or the rest of the population), or a random subset of the population. In the later case the non–dominated elements obtained might not be truly non-dominated considering the whole population, however the process is much faster. It is analogous to the distinction between local and global Pareto fronts (Deb 1999*b*, Zitzler 1999)[3].

[3]Set P is a *local Pareto–optimal set* if for every solution $x \in P$, there exists no solution y satisfying $||x - y|| \leq \varepsilon$ dominating any member of P.

The best know algorithm for finding all maximal elements (i.e. for finding Pareto set) in a partially ordered set of m–dimensional vectors of cardinality n is of the complexity $C_m(n) \leq O(n \log_2 n)$ (for $m < 4$) and $C_m(n) \leq O(n(\log_2 n)^{m-2})$ (for $m \geq 4$) (Kung, Luccio & Preparata 1975).

Example 3.4 Suppose that the population has 15 elements with the objective values presented in Table 3.5(a).

#	y_1	y_2
0:	-87.44	7139.47
1:	-34.74	3205.95
2:	14.38	4902.83
3:	-41.99	6320.33
4:	-81.61	5539.14
5:	-7.82	4896.72
6:	-226.22	0.00
7:	-117.26	3537.79
8:	56.81	3731.48
9:	-192.15	5365.38
10:	-19.63	4108.06
11:	-64.94	6959.60
12:	-158.67	6537.84
13:	0.56	6462.38
14:	29.52	2304.88

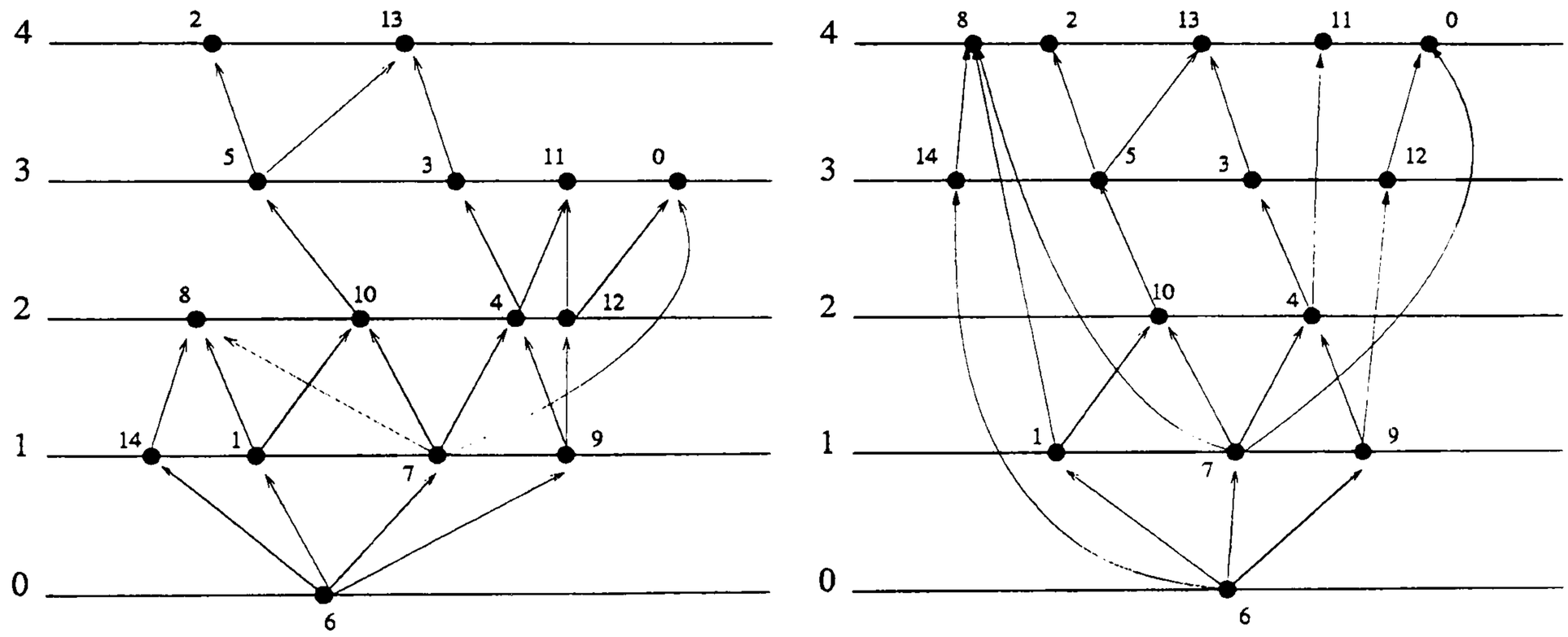
(a)Random population.

#	y_1	y_2	dominated?	rank r_1	rank r_2
0:	-87.44	7139.47	no	3	4
1:	-34.74	3205.95	yes	1	1
2:	14.38	4902.83	no	4	4
3:	-41.99	6320.33	yes	3	3
4:	-81.61	5539.14	yes	2	2
5:	-7.82	4896.72	yes	3	3
6:	-226.22	0.00	yes	0	0
7:	-117.26	3537.79	yes	1	1
8:	56.81	3731.48	no	2	4
9:	-192.15	5365.38	yes	1	1
10:	-19.63	4108.06	yes	2	2
11:	-64.94	6959.60	no	3	4
12:	-158.67	6537.84	yes	2	3
13:	0.56	6462.38	no	4	4
14:	29.52	2304.88	yes	1	3

(b) Dominance and ranks of the population.

Table 3.5. Examples of ranks.

After application of the above ranking procedure, the following Pareto set and the rankings r_1 and r_2 are obtained as presented in Table 3.5(b) with the graphical inter-dominance and ordering representation in Figure 3.8.



(a) Rank r_1 (b) Rank r_2

Figure 3.8. Graphical representation of Pareto rankings r_1 and r_2.

Looking at the results in Table 3.5(b), it can be seen that those two ranking procedures sometimes give different ranks, but, being based on the same ordering, the following property is valid:

$$r_1(x) \leq r_1(y) \Leftrightarrow r_2(x) \leq r_2(y) \tag{3.7}$$

It can be noted that the ranking r_2 assigns the same highest rank to all non-dominated elements. On the other hand, ranking r_1 assigns different ranks to non-dominated elements (e.g. non-dominated element 8 in Figure 3.8 has rank 2, non-dominated elements 0 and 11 have rank 3 and non-dominated elements 2 and 13 have the highest rank 4) allowing further classification among non-dominated elements.

3.4.1 Why ranking?

It is interesting to observe how the number of non-dominated elements in a population of a non–ranking Pareto GA increases. Figure 3.9, presents a diagram of number of non–dominated elements in a population versus generation number for the BAe function. Population size used is 100. Standard tournament of size 2 has been used as a selection method. The results are averaged over 50 runs and standard deviation is presented using error bars. Optimisation has been performed on y_3, y_4 and y_9 simultaneously. It can be immediately seen that after some 150 generations more then 50% of the populations is formed by non–dominated individuals and that the selection pressure gradually decreases with the number of generations.

In the case of conceptual design, this also means that the Pareto front, the set the designer is usually interested in, will be much too large for a human to handle. Even with a small population of 100 individuals, after 400 generations, the size of the Pareto front increases to 80 i.e. 80 solutions (more or less different) to deal with. Therefore, it is desirable to have an additional classification method that is able to classify non–dominated elements. Our ranking method r_1 has that property.

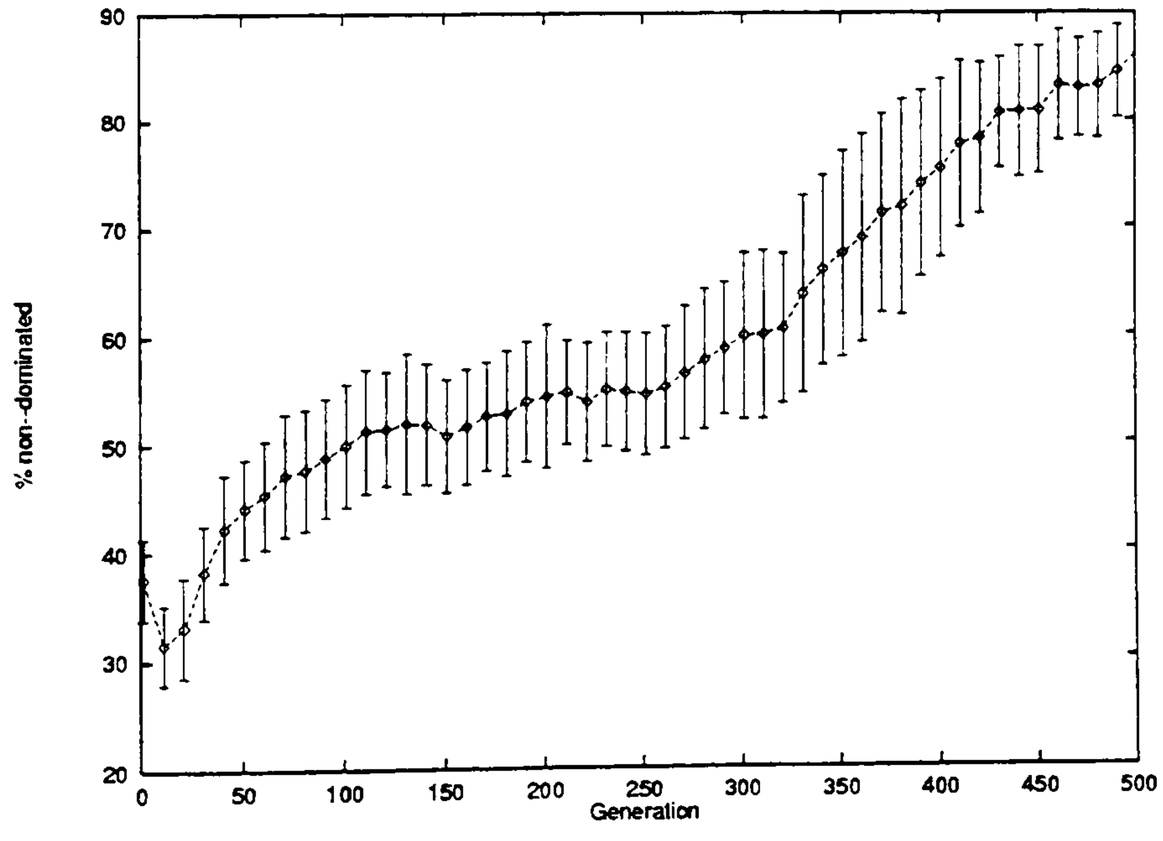


Figure 3.9 Non–dominated percent of the population for tournament selection of size 2, average over 50 runs. Average with standard deviation as error bars.

3.4.2 Some general remarks about searching

For one searching process to successfully perform its function, three components are needed:

Ranking: Use of *ranking* to distinguish between elements; Fine–grain distinguishing might be necessary;

Diversity: Some way to keep *diversity* of the population is needed: niching and sharing (Horn & Nafpliotis 1993, Mahfoud 1995) could be used or some form of steady–state algorithm (Whitley & Kauth 1988, Whitley 1989, Syswerda 1990), where new generated elements (e.g. in Pareto set) are accepted only if their distance (genotypic/phenotypic or in function space) from all other elements are greater then some threshold. One alternative method is given in (Bonham & Parmee 1999) for solving so called non–efect problems of crossing over with population immigrants. They suggest "injecting low discrepancy chromosomes into the crossover phase" (Bonham & Parmee 1999, p. 1492) to avoid this problem.

Elitism: Recent investigations by Zitzler et al. (2000) have shown that *elitism* is an important factor for improving evolutionary multi–objective search.

3.5 Pareto optimisation based methods

In comparing the Pareto principle based multi-objective optimisation with lexicographic order based optimisation (Ben-Tal 1979), there are two extremes concerning the objective importance: in the case of Pareto optimisation, all objectives are considered equally important whereas in the case of lexicographical order the first objective is the most important one and only when the first objective is optimised, the second objective is then considered etc. This section describes the development of a new optimisation method based on the Pareto principle with the possibility of specifying the relative importance of objectives.

As in the case of weighted sum based methods, the relative importance of objectives in this weighted Pareto method could be specified using weights (quantitatively) or they could be combined with (in section 5.3.2 developed) preference method that translates qualitative into quantitative specifications. Without this combination, our weighted Pareto method would suffer from the same problem as the weighted sum method: how to specify weights in the case of 15–20 or more objectives (it is estimated that 7 ± 2 is the maximal number of chunks of similarly classified data a person can work on at the same time (Miller 1956, Cha 1997)).

3.5.1 Definition of weighted Pareto method

The terms strong and weak dominance (Luce & Raiffa 1957, p. 286) are defined here due to the slight terminological confusion with game theory concepts concerning dominance: In game theory *A strongly dominates B* if *A* is preferred to *B* for each state of nature, and *A weakly dominates B* if *A* is preferred to *B* for at least one state of nature and is preferred or indifferent to *B* for all other states. For similar definition of dominance see (Deb 1998, Lin 1976).

Lin (1976) also distinguish between 3 orders on k–dimensional vectors:

$$\boldsymbol{x} \geq \boldsymbol{y} \quad \text{if and only if} \quad (\forall i \leq k)(x_i \geq y_i) \tag{3.8}$$

$$\boldsymbol{x} > \boldsymbol{y} \quad \text{if and only if} \quad (\forall i \leq k)(x_i > y_i) \tag{3.9}$$

$$\boldsymbol{x} \gneqq \boldsymbol{y} \quad \text{if and only if} \quad (\boldsymbol{x} \geq \boldsymbol{y}) \wedge (\exists j \leq k)(x_j > y_j) \tag{3.10}$$

He notes that the orders (3.8) and (3.10) are definable in terms of each other and thus ordering a set in $\mathbf{R}^k$ by $\gtrdot$ is equivalent to ordering a set by $\geq$ (Lin 1976, p. 46), see Appendix A, page 130 for more details.

In order to avoid this terminological confusion, the definition of *non–dominance* used in this work is given below:

Definition 3.9 *The vector* $\boldsymbol{x} = (x_1, \ldots, x_k)$ *is* non–dominated *(in object space) by the vector* $\boldsymbol{y} = (y_1, \ldots, y_k)$, *denoted* $\boldsymbol{x} \succeq \boldsymbol{y}$, *if* $x_i \geq y_i$ *for all* $1 \leq i \leq k$. *In other words,*

$$\boldsymbol{x} \succeq \boldsymbol{y} \Leftrightarrow \frac{1}{k} \sum_{i=1}^{k} I_{\geq}(x_i, y_i) \geq 1, \tag{3.11}$$

where

$$I_{\geq}(x, y) = \begin{cases} 1, & x \geq y \\ 0, & x < y \end{cases}$$

Equation (3.11) can be generalised in the following way (assuming $\sum_{i=1}^{k} w_i = 1$):

$$\boldsymbol{x} \succeq_{w} \boldsymbol{y} \text{ if and only if } \sum_{i=1}^{k} w_i \cdot I_{\geq}(x_i, y_i) \geq 1, \tag{3.12}$$

or some threshold $\tau \leq 1$ can be introduced i.e.

$$\boldsymbol{x} \succeq_{w}^{\tau} \boldsymbol{y} \text{ if and only if } \sum_{i=1}^{k} w_i \cdot I_{\geq}(x_i, y_i) \geq \tau. \tag{3.13}$$

Definition 3.10 *Relation* $\succeq_{w}$ *defined by (3.12) is called* w–non-dominance *and the relation* $\succeq_{w}^{\tau}$ *defined by (3.13)* (w, τ)–non-dominance.

Note: The standard dominance relation is just a special case of (3.13) for $w = (\frac{1}{k}, \frac{1}{k}, \ldots, \frac{1}{k})$ and $\tau = 1$.

Example 3.5 Let $\mathbf{F}(x) = (x_1^2, x_2^2, x_3^2, x_4^2)$ and $w = (1/2, 1/3, 1/6, 0)$. Using above defined orders:

$$\mathbf{F}(1,2,3,-4) \succeq \mathbf{F}(1,2,3,1)$$

$$\mathbf{F}(1,2,3,5) \succeq_{w} \mathbf{F}(1,2,3,7) \text{ but } \mathbf{F}(1,2,3,7) \succeq \mathbf{F}(1,2,3,5)$$

$$\mathbf{F}(1,3,7,4) \succeq_{w}^{0.6} \mathbf{F}(0,4,5,9)$$

Setting all weights to $1/k$ (for k objectives) and threshold $\tau < 1$, the case of so called 'weak dominance' is obtained, where for the dominance at least $[k \cdot \tau]$ components are required to perform better. Related is also a concept of 'restricted dominance' (Bana e Costa 1990*a*, p. 365) which can occur in the situation where the set of points of indifference among objectives is disjoint with the set of feasible points.

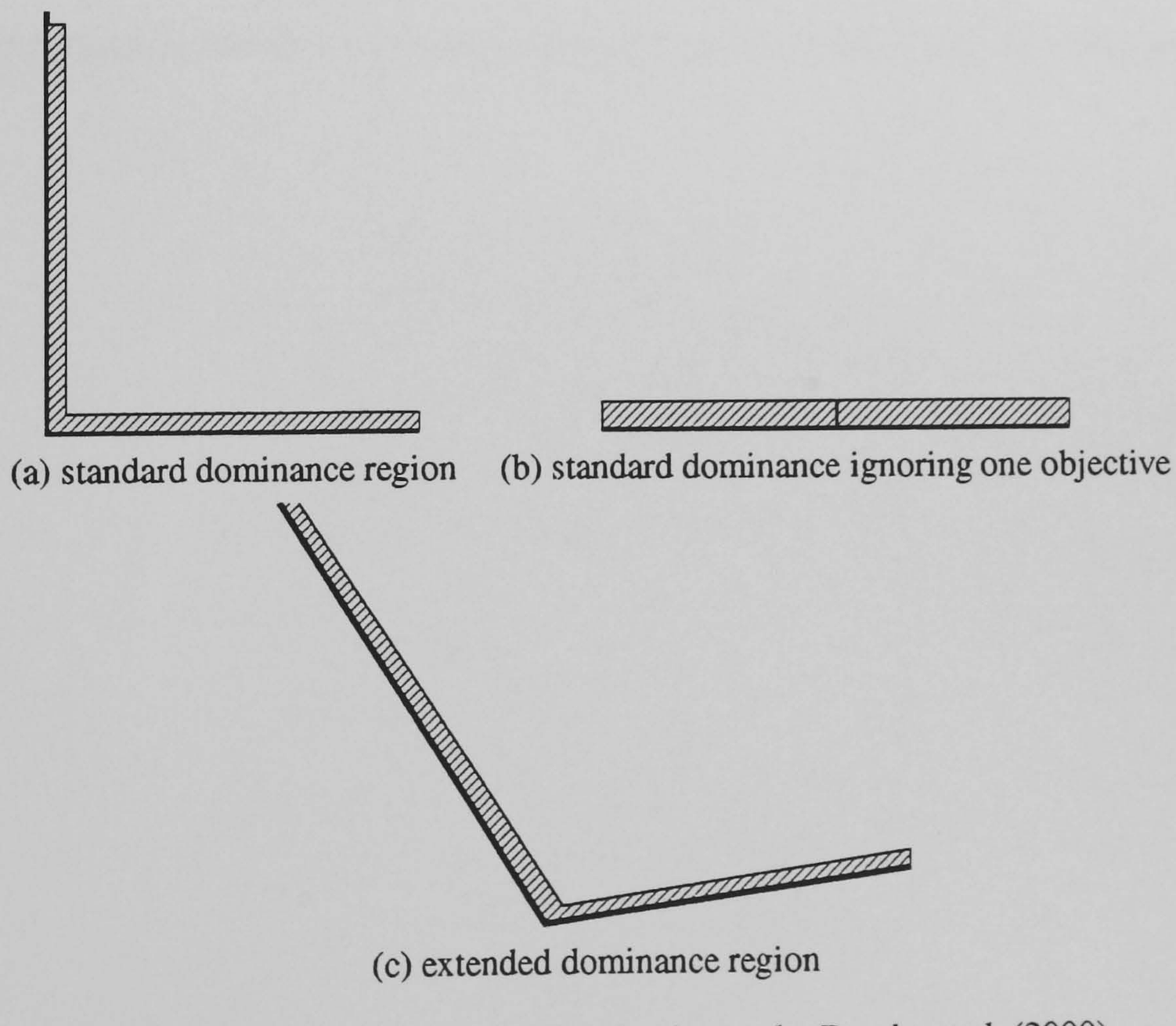
Note: The relation $\succeq_{w}^{\tau}$ is transitive as a product of transitive (component–wise) orders and has all the usual features of an order relation. Also, it is assumed that the weights do not change during the optimisation process.

Definition 3.11 *The* Pareto front *is defined as a maximal set of non–dominated elements (according to a given order* $\succeq$*) and this definition is naturally extended to* w*–Pareto front and to* (w, τ)*–Pareto front for a given vector of weights* w *and threshold* τ *i.e. according to the order* $\succeq_w$ *and* $\succeq_w^\tau$ *given by (3.12) and (3.13) respectively. It is assumed that at least one of the inequalities is strict.*

Vector w can be specified directly by the designer or it can be calculated from his preferences which would help the designer to work in more qualitative terms without the burden to reason if the weight should be set of 0.1 or to 0.09 and how is it going to affect his search.

The Pareto front method combined with genetic algorithms is a very powerful optimisation method since it maintains the diversity of population. However, it could be computationally very expensive.

One very recent method that introduced bias among the Pareto solutions is described in (Deb 1999*a*). Weightings are introduced in the sharing function to change the density of solutions found: in the more interesting regions they allow more points, in less interesting regions Pareto points are sparser. However, it is not immediately clear how small changes in weights influence the density and the whole Pareto front is still generated. Another bias–introducing method is given in (Branke et al. 2000) that changes the shape of dominance region from right angle to cover the wider region and restrict the Pareto front in that way as presented in Figure 3.10. However, so far this research has only been restricted to 2–dimensional optimisation problems and its generalisation to higher dimensional non–dominance is not straightforward.



(a) standard dominance region (b) standard dominance ignoring one objective

(c) extended dominance region

Figure 3.10. Standard and extended dominance by Branke et al. (2000).

Examples of using a weighted Pareto front are given is chapter 6, page 70.

Considering the Pareto front of y_4 versus y_9 obtained using multi–objective Genetic Algorithm optimising y_4 and y_9 only, Figure 3.11 shows the final size of the (w, τ)–Pareto front at the end of GA run for different

weights w and different Pareto thresholds τ. Results were obtained running GA with population size 50 for 200 generations with τ from 0 to 1, step 0.1, and y_4–weight from 0 to 1 in steps of 0.05, and are averaged over 15 runs (3465 runs in total).

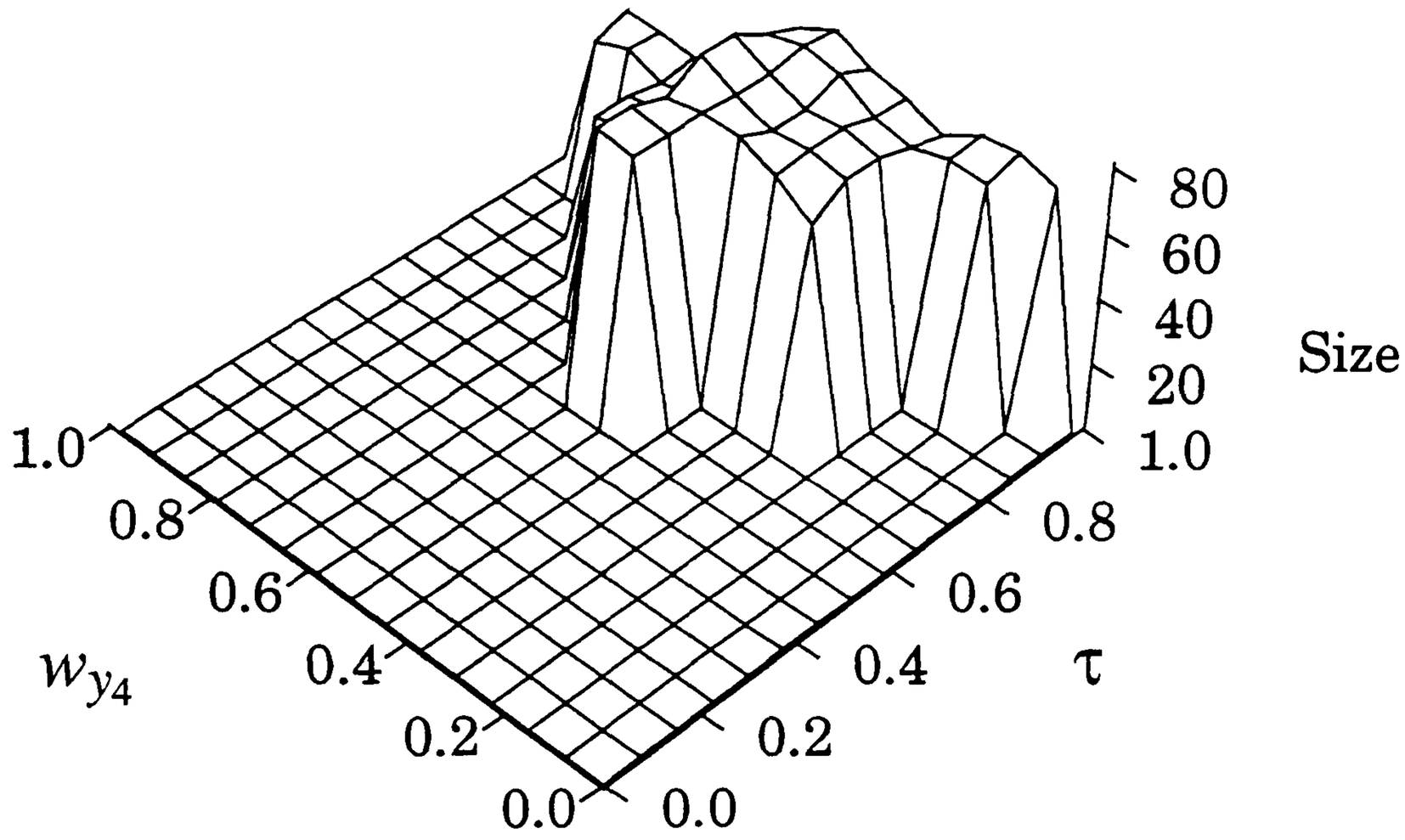


Figure 3.11. Size of (w, τ)–Pareto front of y_4 versus y_9 for the BAe function as a function of w and τ.

The weighted Pareto method developed here is integrated with preference method in chapter 5 and applied in chapter 6.

CHAPTER 4

Optimal Parameter Setting for the Real Valued Genetic Algorithm for Multi–Objective Optimisation of the BAe Function

---

This chapter describes experiments run for the identification of an optimal parameter setting relating to genetic operators and /or parameters for real valued genetic algorithm (RGA) developed for multi–objective optimisation. Since an universal optimal setting does not exist (as implied by "No free lunch" theorem in 2.4, page 17), these experiments concentrate on the choice of parameters giving optimal results for the BAe function. All GA operators used in the RGA are described in detail in chapters 2 and 3.

The GA for multi–objective optimisation has many different options:

```
RealGA v 1.105, Dragan Cvetkovic, EDC, 2000/04/12
Usage: main.BAe [options] [log_file], where options are:
     -a number    If -V 4 copy number individuals between subpopulations
     -A file[,0]  Ask about preferences and compute weights.
                  If file is '_' read from stdin
                  If ,0 is added, read order, not preferences
     -B string    Mask for input variables: 0 if fixed, 1 if variable
     -c number    Crossover probability
     -C char      Crossover type: N (n point), U (uniform), S (SBX), F (Fuzzy)
                  I (intermediate), R (random intermediate) or _ (none)
     -d           Toggle penalty on negative values. Default 1
     -e number    If 1, use elitist strategy otherwise do not
     -E number    Maximal number of functions evaluations
     -f file_name File name with input arguments (default caps.dat)
     -G n[,n1,n2] Show graphics every n generation and n1 vs. n2 graph
                  Parameters n1 and n2 are optional
     -g number    Maximal number of iterations
     -h           Print this help screen
     -H number    Do the hill climbing on the number best element
     -i number    Don't perform mating if distance less then
                   number*average distance. Default number is 0.350000
     -I number    Shuffle/Migrate population every number generations.
                    Works only with -V 3 or -V 4 option. Default 5 gen
     -K number    Cooling factor for EXP mutation (>=0). Default 0.100000
                  Actual cooling factor will be gen times this factor
     -l string    Do lexicographic sort according to the mask.
                  Elements in mask are separated with ',' and indexing starts
                  from 0. Example: 2,0,3,1 means sort for 3rd then for 1st etc
     -L number    Do random lexicographic sort with order
                  changed every number generation. Default 0 gen
     -m number    Mutation probability. Default 0.111111
     -M char      Mutation type: S (random), E (exp) G (Gauss) or _ (none)
```

```
-n number     Number of crossover points (>0), default is 2
-N string     String with OUTPUT variables mask.
              + means maxmimise, - minimise, 0 ignore. Default 000+0000+0000
-O number     Part of the variable range outside definition
               range to be explored (in [0,1], default 0)
-p number     Number of individuals in population
-P n,sw       Do Pareto sort instead of fitness based one
               Here n is the max size of Pareto set.
               sw is R for ranking, S for stochastic and
               T for vanilla Pareto. Default is T
-q n1[,n2,n3]  Run in quiet mode: don't argue about bad
                    parameters, provide your own instead.
              n1==1 means stop after specified number of gen (default),
              n1==2 means ask user if to continue for n2 (default 100) gen,
              n1==3 means continue for n2 gen if the last
                improvement was within n3 (default 400) gen
-R number     Use ranking method for selection with number
              as rank_min, otherwise use roulette wheel
-r number     Starting random seed
              if not presented will be generated
-s number     Probability in stoch. tournament, default 0.75
-S char       Selection type: one of W (roulette wheel), T (tournament)
              B (truncation), P (Pareto truncation)
              or R (random).
-T number     Using tournament or pareto method with number
              as a tournament size. If 1, will use stochastic tournament
              If selection is truncation, fraction of the population
-t number     Pareto sort threshold (between 0 and 1, default 1)
-U number     Probability for parameterized uniform
              crossover, default value is 0.500000
-V number     Simulate VEGA. Meaning of the number:
      0      --- no VEGA
      1      --- Schaffer VEGA: maintain subpopulations etc
      2      --- Fourman VEGA: choose using random objective
      3      --- Work with subpopulations and shuffle them
      4      --- Work with subpopulations and migrate bests.
-v number     What amount of information to display:
              0 -- nothing, 2 -- all, 3 -- every individual,
              1 --  current run and best fitness. Default 0
-W file_name File with non 1 weights of OUTPUT array
-w file_name File with min and maxs of OUTPUT array
-X number     Number of input parameters (default 9)
-Y number     Number of output parameters (default 13)
-z file[,P|O[,file]] Specify scenarios file.
              '_' specifies stdin
              If 'P' set ask for preferences among scenarios
              If 'O' set specify preference order instead
              If file after P is specified, use as response file.
```

However, most of the options have default settings and therefore

```
./RGA -q1 -v1
```

is more usual.

4.1 Related work

First results concerning genetic operators and parameter settings were obtained by de Jong (1975) and he suggested:

- population size 50–100,
- crossover rate 0.6, and
- mutation rate of 0.01.

Study by Grefenstette (1986) gives the following setting:

- population size 30,
- crossover rate 0.95, and
- mutation rate of 0.01.

Trying to improve on Grefenstette (1986), Schaffer, Caruana, Eshelman & Das (1989) performed extensive testing (for improving online performance of GAs) and suggested the following setting:

- population size 20–30,
- crossover rate 0.75–0.95, and
- mutation rate of 0.005–0.01.

The study by Cvetković & Mühlenbein (1994) gives an optimal population size N^* for ONEMAX function using Breeder Genetic Algorithm (BGA) obtained by extensive testing:

$$N^* = 1 + (10.28 - 12.07I + 7.30I^2)\sqrt{n} \cdot \ln n \cdot (\frac{1}{\sqrt{p_0}} - 1) \quad 0.8 \leq I \leq 1.4 \tag{4.1}$$

where n is the size of the problem (length of a chromosome), p_0 is the probability of allele 1 in initial generation (usually 1/2) and I is the selection intensity of the truncation selection T:

$$I = \frac{1}{T\sqrt{2\pi}} e^{-t_0^2/2}$$

where t_0 is defined so that:

$$T = \begin{cases} \frac{1}{2} - \frac{1}{\sqrt{\pi}} \int_0^{t_0} e^{-t^2}\, dt, & \text{if } T \leq \frac{1}{2}; \\ \frac{1}{2} + \frac{1}{\sqrt{\pi}} \int_0^{t_0} e^{-t^2}\, dt, & \text{if } T > \frac{1}{2}. \end{cases}$$

More recent study is by Harik, Cantú-Paz, Goldberg & Miller (1999) and their estimate of the population size is

$$N = -2^{k-1} \ln(\alpha) \frac{\sigma_{bb}\sqrt{\pi m'}}{d}$$

where k is the order of building block, α is the probability of failure, σ_{bb} is the fitness variance of the partition considered and m' is one less than the number of building blocks in a string.

4.2 Mutation

In RGA the user has a choice of three different mutation types (for details see section 2.3.2 on page 15):

1. Random mutation
2. EXP mutation with parameter c

3. Gauß mutation

In the sequel, tests for finding optimal mutation type and the optimal value of EXP mutation parameter c are described.

In all runs mutation probability $p_m = 1/n$ has been used, where n is the number of variables, as suggested by Mühlenbein & Schlierkamp-Voosen (1993*a*) and Bäck (1993).

4.2.1 Mutation type

All three mutation types have been tried in their default settings i.e.:

- Gauß noise is with normal $\mathcal{N}(0;1)$ distribution;
- Default cooling factor for EXP mutation is $c = 0.1 \times \text{gen}$ and the new value for variable x is

$$x' = x \pm (x_M - x_m) \cdot 2^{-c\alpha} \tag{4.2}$$

where $\alpha : \mathcal{U}(0,1)$ is a random number with uniform $(0,1)$ distribution.

Experiments were run 100 times and results in combination with uniform crossover are given in Table 4.1(a) and in Table 4.1(b) in combination with 1–SBX crossover. Objectives to optimise are SEP2 and FR of the BAe function (described in Section 3.3, page 27) and the search process would stop if the best fitness hasn't changed for 400 generations. All other parameters of the GA were kept on their default values. As a random number generator, Mersenne Twister (Matsumoto & Nishimura 1998)[1] has been used as it possess much better 'randomness' properties than any 'built–in' random number generator (such as `rand()` or `random()` in UNIX).

Mutation Type	Fitness		Generation	
	Aver	Dev	Aver	Dev
Random	7888.09	17.97	1857.4	936.45
Gauß	7802.69	120.6	2822.0	1836.4
EXP	7863.48	69.08	521.86	199.64

(a) with uniform crossover

Mutation Type	Fitness		Generation	
	Aver	Dev	Aver	Dev
Random	7905.22	5.67	1840.8	1092.9
Gauß	7850.96	95.78	1185.8	1116.3
EXP	7908.28	2.32	1215.1	495.27

(b) with 1–SBX

Table 4.1. Results on different mutation types.

As it can be seen, in combination with uniform crossover, random mutation gives the best results but it takes quite long to find them. Gauß mutation (with default setting) performed quite bad in both fitness and convergence, whereas EXP mutation managed to find very good results very fast. Combined with 1–SBX, EXP mutation performs the best and random as a second best.

4.2.2 Parameters of EXP mutation

Further runs have been performed modifying parameter c in EXP mutation. Averaged results over 100 runs for different c values in combination with uniform crossover are presented in Table 4.2(a) and plotted in Fig-

[1] available from `http://www.math.keio.ac.jp/~matumoto/emt.html`

ure 4.1(a). Averaged results over 100 runs for different c values in combination with 1–SBX crossover are presented in Table 4.2(a) and plotted in Figure 4.1(b). Table 4.2(c) and Figure 4.1(c) show different c values in combination with 1–SBX with run time limited to 200 generations.

Setting $c = 0.1$ gives the best compromise between quality of results and convergence speed. Note that as $c \to 0$, EXP mutation behaves more and more like the random mutation.

EXP	Fitness		Generation	
factor	Aver	Dev	Aver	Dev
0.01	7890.55	21.80	2169.8	409.14
0.05	7868.55	63.67	816.82	242.10
0.1	7859.31	69.75	520.14	219.44
0.25	7906.05	9.48	898.29	412.99
0.5	7842.04	72.27	340.03	214.01
0.8	7802.04	134.6	422.45	284.56

(a) with uniform crossover

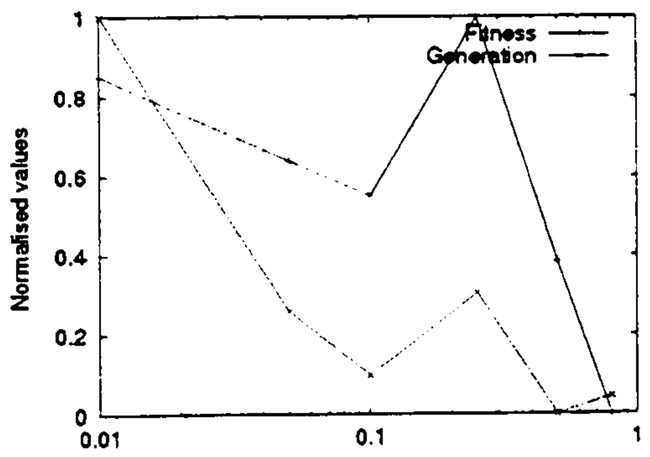
EXP	Fitness		Generation	
factor	Aver	Dev	Aver	Dev
0.01	7909.24	0.71	1952.8	798.70
0.05	7909.11	0.98	1686.5	560.62
0.1	7908.33	3.01	1186.5	445.74
0.25	7904.95	12.57	871.31	409.37
0.5	7902.56	13.13	885.33	621.34
0.8	7892.36	33.79	684.90	440.87

(b) with 1–SBX

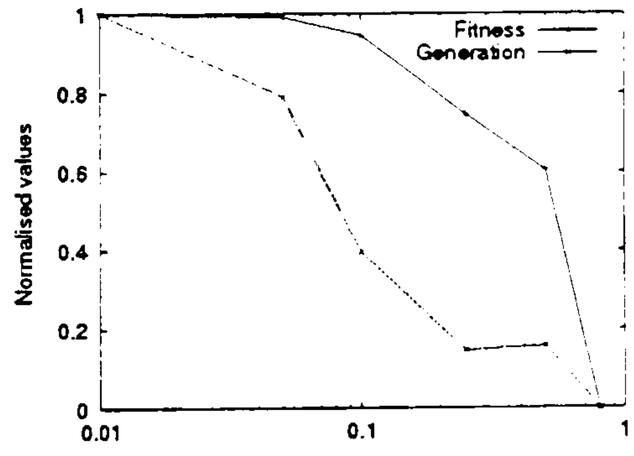
EXP	Fitness		Generation	
factor	Aver	Dev	Aver	Dev
0.01	7887.70	27.62	176.88	21.70
0.05	7901.24	10.14	190.62	10.70
0.1	7905.47	7.37	191.03	12.53
0.25	7901.69	17.86	185.56	16.70
0.5	7884.79	64.64	182.58	19.83
0.8	7872.28	52.99	180.78	24.77

(c) 1–SBX, run limited to 200 generations

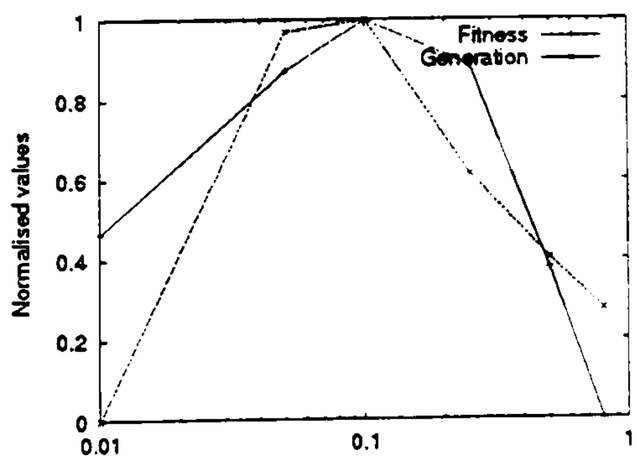
Table 4.2. Results on different mutation factors for EXP mutation.



(a) with uniform crossover



(b) with 1–SBX



(c) 1–SBX, run limited to 200 generations

Figure 4.1. Normalised fitness and number of generations for different mutation factors for EXP mutation.

4.3 Crossover

Concerning crossover, the user has a choice of the following crossover types (described in Section 2.3.1, page 13):

1. n–points crossover;
2. uniform crossover;
3. intermediate crossover (child is arithmetical mean of parents);
4. random intermediate crossover (child is computed as $\lambda p_1 + (1-\lambda)p_2$, where λ is $\mathcal{U}(0,1)$ random number with uniform distribution);
5. SBX with $\eta \in \{0.5, 1, 1.5, 2, 3, 4, 6, 8\}$.

4.3.1 Crossover type

Similar tests as for the mutation runs have been performed : 100 runs per crossover type (all with probability 1) with all other parameters kept constant: optimisation of SEP2 and FR, population size 50 etc. Results are given in Table 4.3(a) and Figure 4.2(a). Limiting the run time for all crossover types to 200 generations only, the results are as in Table 4.3(b) and Figure 4.2(b). From the results it can be seen that the intermediate crossover gives quite good results but it takes many generations to converge to those results. Uniform and 4–point crossover give good compromise. The SBX gives exceptionally good results for $\eta \leq 2$ but unfortunately the convergence time is excessive. Results of running the SBX without mutation, are presented in Table 4.4 and in Figure 4.2(c).

Crossover Type	Fitness		Generation	
	Aver	Dev	Aver	Dev
1–point	7862.27	57.28	578.37	244.89
2–points	7848.45	76.66	480.75	186.17
4–points	7860.57	73.40	509.25	196.07
6–points	7853.74	98.88	564.95	244.49
0.5–interm	7882.91	29.25	1089.4	267.18
R–interm	7890.94	23.41	1058.3	300.52
uniform	7860.12	65.08	526.00	195.95
fuzzy	7899.33	14.26	773.09	206.18
0.5–SBX	7909.15	1.70	1398.4	610.08
1–SBX	7908.94	23.41	1058.3	300.52
1.5–SBX	7905.85	8.99	1064.3	429.40
2–SBX	7902.94	10.83	972.24	353.88
3–SBX	7895.22	19.30	863.48	391.70
4–SBX	7892.89	27.26	827.45	341.94
6–SBX	7879.48	37.22	652.20	234.75
8–SBX	7877.59	47.87	729.44	308.56
20–SBX	7843.76	90.56	663.95	205.93

(a) unlimited run

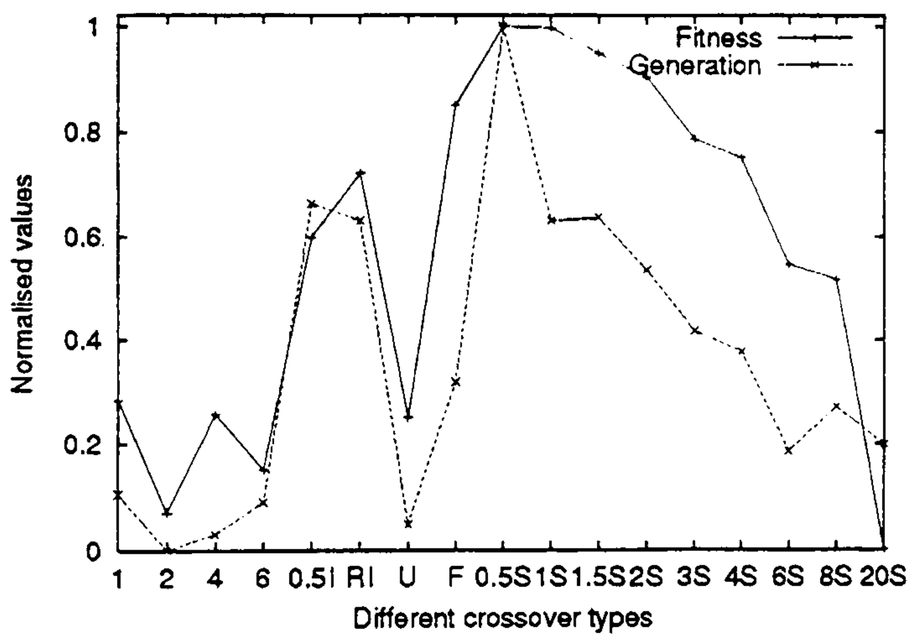
Crossover Type	Fitness		Generation	
	Aver	Dev	Aver	Dev
1–point	7846.09	88.84	188.48	12.6
2–points	7859.32	74.50	187.15	13.9
4–points	7855.47	61.42	188.66	11.4
6–points	7864.19	57.48	186.56	12.8
0.5–interm	7883.07	26.85	193.73	6.49
R–interm	7882.16	29.81	194.73	6.23
uniform	7855.69	81.46	188.77	10.6
fuzzy	7900.34	15.29	188.00	14.15
0.5–SBX	7905.97	5.14	187.77	15.89
1–SBX	7906.30	3.94	190.78	13.2
1.5–SBX	7904.65	7.84	186.89	14.7
2–SBX	7898.64	14.35	189.21	12.7
3–SBX	7893.09	24.84	187.78	13.5
4–SBX	7886.65	34.49	184.55	18.3
6–SBX	7878.08	46.61	188.33	15.4
8–SBX	7863.36	56.18	185.71	15.7
20–SBX	7868.48	54.79	184.43	16.39

(b) 200 generations only

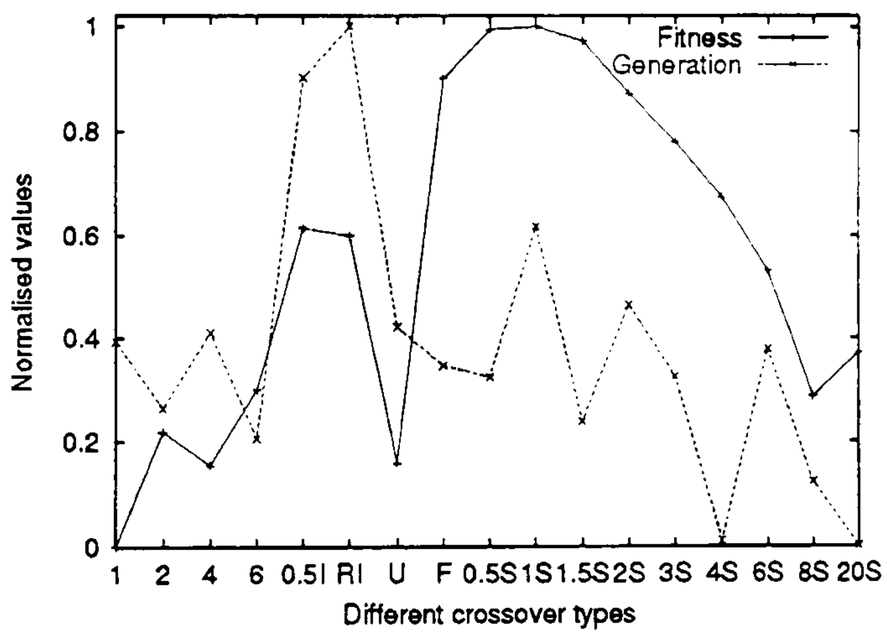
Table 4.3. Results on different crossover types.

Crossover Type	Fitness		Generation	
	Aver	Dev	Aver	Dev
1–SBX	7672.43	179.6	6237.7	5344.2
1.5–SBX	7370.36	253.0	3898.8	3744.7
2–SBX	7221.20	290.8	2519.2	3300.4
3–SBX	7073.62	311.7	397.27	790.64
4–SBX	7016.57	291.7	213.63	389.51
6–SBX	6925.35	313.0	125.09	189.52
8–SBX	6901.72	335.3	63.88	77.597

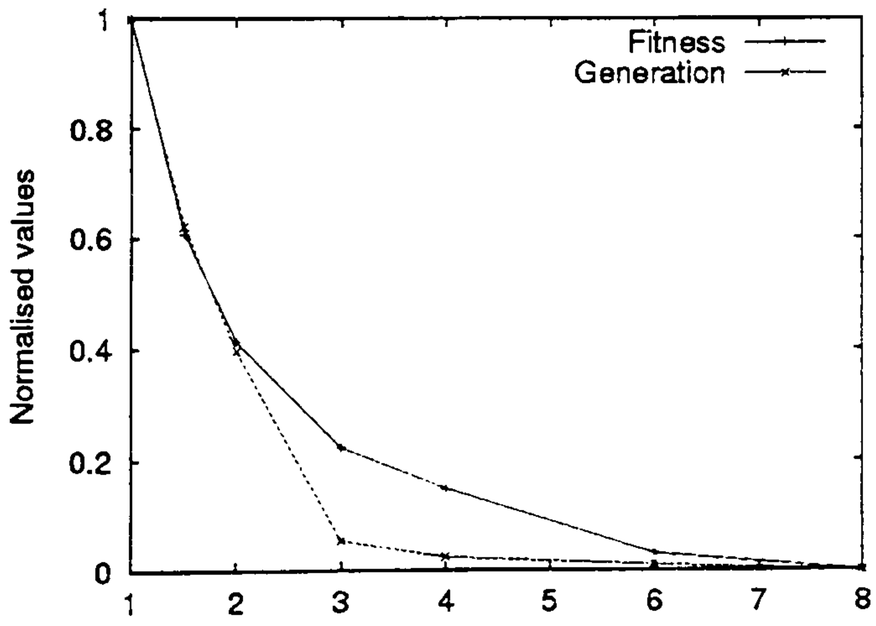
Table 4.4. Results on different SBX types without mutation.



(a) unlimited run



(b) 200 generations only



(c) SBX without mutation

Figure 4.2. Normalised fitness and number of generations for different crossover types.

4.3.2 Crossover probability

Next, the optimal crossover probabilities for uniform crossover and for SBX are computed. Results for uniform crossover are given in Table 4.5(a) and the results for 1–SBX are given in Table 4.5(b). The corresponding graphs are given in Figure 4.3(a) and 4.3(b).

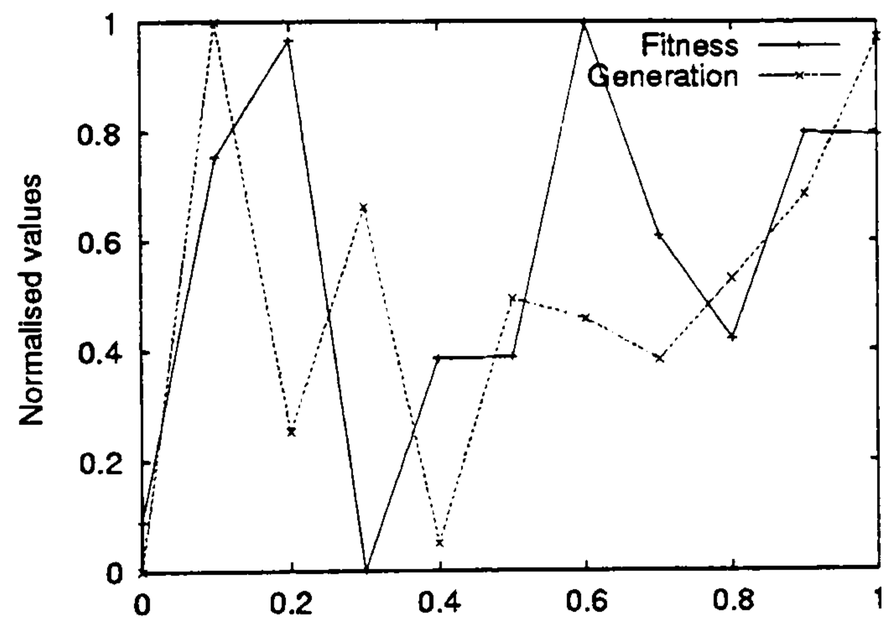
Crossover Probability	Fitness		Generation	
	Aver	Dev	Aver	Dev
0.0	7846.78	107.6	509.98	203.13
0.1	7862.71	66.98	553.59	250.27
0.2	7867.83	58.30	521.03	184.34
0.3	7844.66	90.46	538.93	239.30
0.4	7853.91	71.00	512.20	205.74
0.5	7853.97	92.06	531.58	213.07
0.6	7868.59	58.17	530.00	211.49
0.7	7859.25	79.86	526.72	200.64
0.8	7854.74	71.61	533.14	223.38
0.9	7863.79	69.06	539.88	198.51
1.0	7863.71	78.21	552.39	210.74

(a) uniform crossover

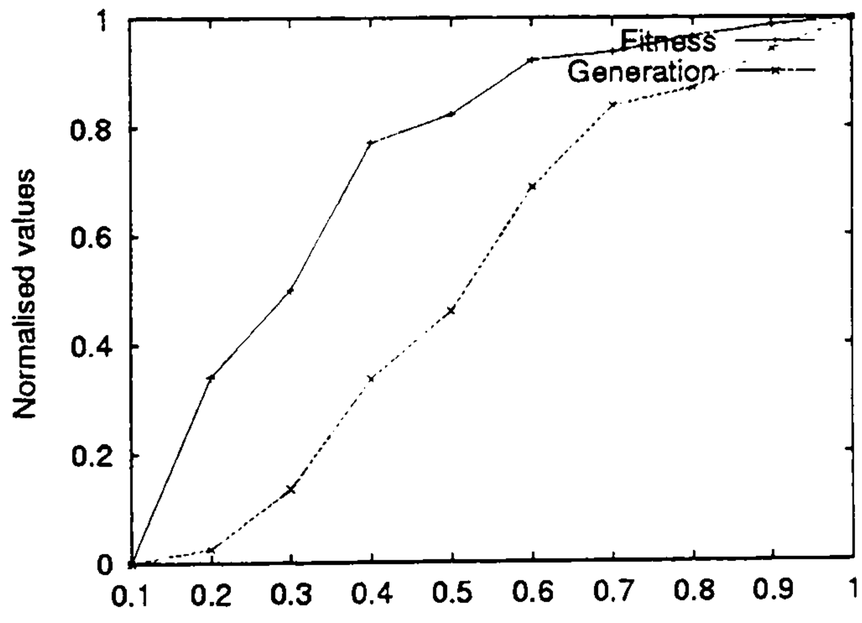
Crossover Probability	Fitness		Generation	
	Aver	Dev	Aver	Dev
0.1	7866.33	48.46	597.92	259.72
0.2	7880.75	36.43	614.26	257.63
0.3	7887.48	36.64	683.34	263.18
0.4	7898.92	13.80	811.89	349.55
0.5	7901.13	17.07	889.77	395.64
0.6	7905.32	6.41	1035.0	477.71
0.7	7905.95	7.98	1130.2	441.83
0.8	7907.16	4.47	1150.5	435.98
0.9	7908.02	2.98	1195.3	551.87
1.0	7908.53	2.75	1231.6	481.86

(b) 1–SBX

Table 4.5. Results on different crossover probabilities.



(a) uniform crossover



(b) 1–SBX

Figure 4.3. Fitness and generations as a function of crossover probability.

It can be noticed that increasing crossover probability improves the results but (in the case of SBX) also increase the convergence time. Another factor that requires some consideration is that increased crossover probability increases the number of crossovers performed. However, in most cases, the crossover operator is not expensive, and the fitness calculation is usually the main bottleneck. Therefore setting crossover probability to its maximal is not a problem here.

4.4 Selection type

Regarding selection types, the user has a choice of the following selection types (described in section 2.3.3 on page 15):

1. Roulette wheel selection (i.e. proportional selection);

2. Tournament selection;

3. Truncation selection;

4. Random selection (where parents are put completely random in a mating pool);

The standard set of experiments (100 runs, run until the best fitness is constant for 400 generations) has been run. Results for uniform crossover are presented in Table 4.6(a) and plotted in Figure 4.4(a). Results for 1–SBX are presented in Table 4.6(b) and plotted in Figure 4.4(b). Taking into account both the best fitness and number of generation, it can immediately be seen that 2–tournament produces the best results. Considering fitness only, proportional and random selection both perform very good but it takes many generations to obtain results: 3 to 4 times longer than when using 2–tournament.

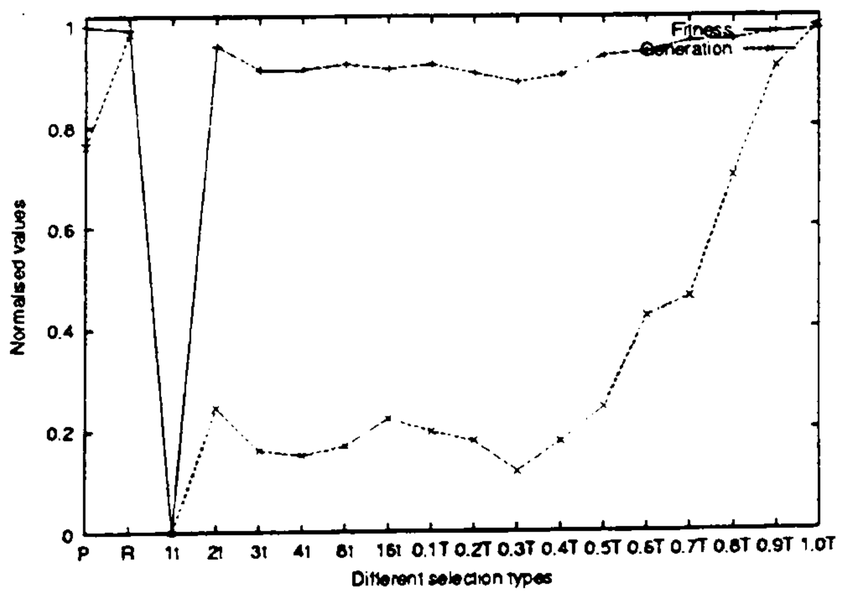
Selection Type	Fitness		Generation	
	Aver	Dev	Aver	Dev
Proport.	7886.49	27.48	1231.4	548.00
Random	7881.86	33.18	1534.9	655.92
1–tourn	7173.65	251.7	195.65	272.92
2–tourn	7859.03	65.74	526.44	219.92
3–tourn	7825.26	97.69	412.84	192.44
4–tourn	7825.53	96.28	399.23	208.72
8–tourn	7832.85	105.5	423.48	181.57
16–tourn	7825.82	104.0	497.10	308.10
0.1–trunc	7832.33	88.99	462.13	276.51
0.2–trunc	7820.23	112.9	436.78	240.07
0.3–trunc	7807.09	122.8	355.07	156.58
0.4–trunc	7817.24	122.7	435.18	182.22
0.5–trunc	7844.31	91.45	525.29	264.98
0.6–trunc	7850.26	84.01	769.52	738.47
0.7–trunc	7864.86	64.96	821.40	514.54
0.8–trunc	7866.88	47.50	1146.9	458.62
0.9–trunc	7877.43	43.91	1443.5	496.75
1.0–trunc	7880.78	34.19	1553.0	652.84

(a) uniform crossover

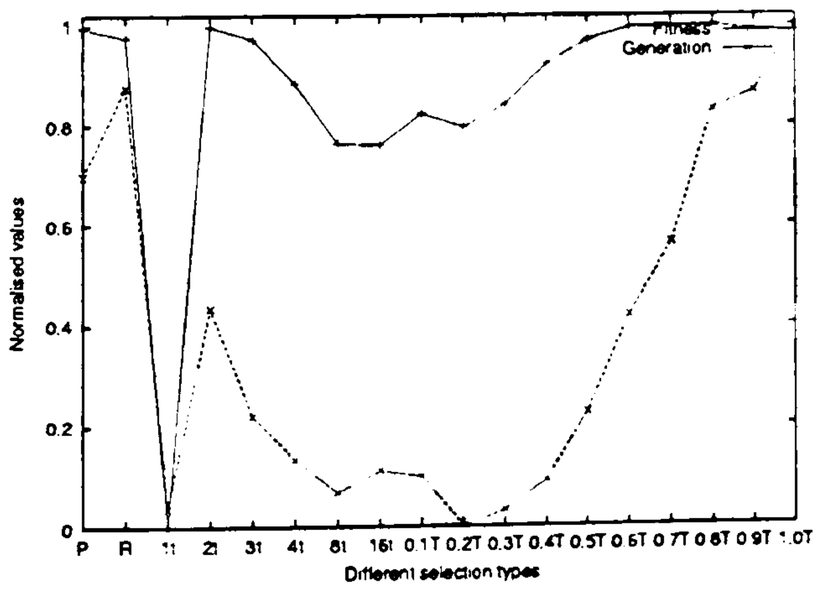
Selection Type	Fitness		Generation	
	Aver	Dev	Aver	Dev
Proport.	7906.89	3.95	1671.7	830.24
Random	7897.29	36.59	1996.9	919.66
1–tourn	7434.31	160.66	461.01	402.61
2–tourn	7908.32	2.60	1188.5	418.31
3–tourn	7895.99	22.62	797.89	349.94
4–tourn	7853.90	69.03	638.21	351.87
8–tourn	7796.94	132.57	518.75	287.15
16–tourn	7795.03	146.01	597.62	346.27
0.1–trunc	7824.83	113.78	576.80	396.91
0.2–trunc	7812.23	128.45	395.88	208.51
0.3–trunc	7833.58	109.82	453.21	246.06
0.4–trunc	7872.09	50.51	560.57	326.41
0.5–trunc	7894.93	31.38	808.67	413.93
0.6–trunc	7906.82	6.92	1159.8	539.21
0.7–trunc	7907.93	5.01	1427.1	623.68
0.8–trunc	7908.93	1.99	1914.2	807.47
0.9–trunc	7904.17	8.07	1978.9	1078.0
1.0–trunc	7902.98	8.12	2224.2	1013.6

(b) 1–SBX

Table 4.6. Results for BAe using uniform crossover and different selection methods.



(a) uniform crossover



(b) 1–SBX

Figure 4.4. Fitness and Generations for different selection types.

4.5 An ideal setting – discussion and conclusions

From the above tests, it follows that the ideal parameters and/or operator setting for RGA is:

- 1–SBX as crossover operator;
- Crossover probability 1.0;
- 2–tournament as selection method;
- EXP–mutation with $k = 0.01$;
- Mutation probability $1/n$ (n is the number of input variables);
- Population size 50;
- Number of generations 200;

Each RGA run needs to be limited to some fixed number of generations (say 200) otherwise, it would keep converging for quite a long time, because of the SBX features.

It is to be stressed that these findings are very problem specific and a different optimisation problem might require different genetic operators and parameters. However, (continuous) real function optimisation problems are generally less sensitive to parameter setting than combinatorial problems (travelling salesman (TSP), job–shop, bin packing etc.) where Hamming cliffs and "Himalaya effects" are much more noticeable.

However, we must not forget that:

- Every setting is generally only suitable for the class of the problem it was investigated on;
- Some of the results reported in subsection 4.1 (related work) above were obtained for the 'toy problems' (ONEMAX etc.) and with the very simple GA;
- In general, the best way is to always perform tests for optimal parameters (resources and/or time permitting).

CHAPTER 5

Use of Preferences in Multi–Objective Optimisation

This chapter presents a transformation method based on preference relations. It transforms non–crisp (qualitative) relationships between objectives in multi–objective optimisation into quantitative attributes (numbers). The preference method is integrated with two multi–objective Genetic Algorithms: the weighted sum GA and the weighted Pareto method, introduced in section 3.5.1, page 33. The complexity issue is discussed here as well as the influence of different preference valuations on the results. Examples of preference relation applications with traditional Genetic Algorithms will be presented in chapter 6.

5.1 Introduction — Accuracy of quantitative qualification

This sections discuss some observational problems. The following quote from Nisbett & Wilson (1977, p. 254) gives an indication of the problem that humans inherently face in the process of judgement and particularly in the process of reporting on the influential factors.

> *Slovic and Lichtenstein (1971) have reviewed the literature concerning the ability of subjects to report accurately on the weights they assign to various stimulus factors in making evaluations. Most of the investigations of this question have employed either clinical psychologists or stockbrokers as subjects, and the judgemental domain has been largely limited to clinical diagnoses and assessments of the financial soundness of stocks. Subjects are asked to diagnose patients using Minnesota Multiphasic Personality Inventory (MMPI) scores or to assess stocks using such indicators as growth potential and earning ratio. Then subjects are asked to state the degree of their reliance on various factors. These subjective weights are then compared to the objective weights derived from regression of the subject's judgement on the various factors. Slovic and Lichtenstein (1971) concluded that self-insight was poor and that of the studies which allowed for a comparison of perceived and actual cue utilization "all found serious discrepancies between subjective and objective relative weights".*

In other words: even when the correct solution to the problem is obtained, we cannot always accurately name the factors affecting the solution process. In chapter 3 two multi–objective optimisation methods are presented that are sensitive on weights chosen: weighted Pareto and weighted sums. Taking into account the results of the quoted study by Nisbett & Wilson, one arising question is: is it possible to develop a method to help the engineer in estimating factor weights during the design process? This chapter introduces one

such method for transformation of qualitative into quantitative values. Nevertheless, the process does not give an universal solution: if the designer chooses the wrong factor to optimise, it will not correct the designer. However, the interactivity of the conceptual design process would give some feedback to the designer enabling him to realise his mistake.

When dealing with industrial design problems, it rapidly becomes apparent that there are significant differences between so called 'textbook optimisation problems' and 'real world applications'. In both cases, a function to be optimised is a multi–objective one (as given by definition 3.1 and equations (3.1)–(3.3) in chapter 3, page 18). Additionally, in this kind of multi–objective optimisation not all objectives are equally important, which necessitates the use of weights or preferences.

As already mentioned, people generally have problems in reporting accurately on the weights they assign to the various stimulus factors in making evaluations. In conceptual design, this means that the designer cannot always completely objectively define the preferences regarding the objectives to be optimised. Therefore, the following situation occurs frequently: A subjective statement "objective A is much more important than objective B" cannot always be supported with a quantitative representation. One method for overcoming this problem is a fuzzy multiple objective optimisation (Lai & Hwang 1996). In this chapter the problem is addressed in a different manner and the developed methods are integrated with different GA–based optimisation techniques.

5.2 Multiple Criteria Decision Making (MCDM) and Multiple Criteria Decision Aid (MCDA)

This section is based on (Roy 1990*a*), and gives a short introduction to MCDM and MCDA .

5.2.1 MCDM and its main features

Until the end of 1960s in operations research (OR), decision–making problems were formulated on the following three bases:

1. A well–defined set A of feasible alternatives a;
2. A real–valued function g defined on A precisely reflecting the preference of the decision–maker D;
3. A well–formulated mathematical problem:

$$\text{Find } a^* \text{ in } A \text{ such that } g(a^*) \geq g(a)\ \forall a \in A.$$

Accordingly, the general framework of MCDM consists of:

1. A well–defined set A of feasible alternatives a;

2. A model of preferences, well shaped in D's mind, rationally structured from a set of attributes. The preferences are defined using utility function U:

$$a'Pa \quad \text{if and only if} \quad U(a') > U(a)$$
$$a'Ia \quad \text{if and only if} \quad U(a') = U(a)$$

3. A well–formulated mathematical problem: the discovery of an optimal alternative a^* in A such that $U(a^*) \geq U(a)\ \forall a \in A$.

5.2.2 From MCDM to MCDA

The practice of operations research (OR) and MCDM has shed light on some fundamental limitations on objectivity. Five major aspects have to be taken into account (see also discussion in section 5.1 and (Cvetković et al. 1998)):

1. The frontier of A (of feasible alternatives) is often fuzzy. Because of this, the borderline between what is and what is not feasible has inevitably a certain amount of arbitrariness. A more crucial limitation on objectivity comes from the fact that this borderline is frequently modified in the light of what is found through the decision process itself (c.f. quote by Goel in section 1.1.1, page 3).
2. In many real world problems, decision maker D, as a person truly able to make the decision, does not really exist: usually, several people take part in the decision process and we tend to confuse the one who ratifies the decision with what is called the decision–maker.
3. Even when D is not a mythical person, D's preferences very seldom seem well–stated: in and among the areas of firm convictions lie hazy zones of uncertainty, half–held belief or, indeed, conflicts and contradictions. We have to admit, therefore, that the study itself contributes to answering questioning, solving conflicts, transferring contradictions and destabilising certain convictions.
4. Data such as the numerical values of performances $g_k(a)$, the analytical forms of distributions such as $\delta_k^a(y_k)$ or $\delta^a(y_1, \ldots, y_n)$ and numerical values of the characteristics of those distributions are, in many cases, imprecise and/or defined in an arbitrary way.
5. In general, it is impossible to say that a decision is a good one or a bad one by referring only to a mathematical model: organisational, pedagogical, and cultural aspects of the whole decision process which leads to making a given decision also contribute to the quality and success of this decision.

Therefore, the general framework of multiple criteria decision aid (MCDA) consist of:

- A not necessarily stable set A of potential actions s;
- Comparison based on n criteria (or pseudo–criteria) g_k;

- An ill–defined mathematical problem.

5.2.3 Requisite for preferences

For an appropriate dominance theory, the following seven requisites could realistically be formulated (Brans & Mareschal 1994):

1. The amplitude of the deviations between the alternatives should be taken into account;
2. As the criteria are generally expressed in different units, the scaling effect should be completely eliminated;
3. When comparing two alternatives a and b, an appropriate multiple criteria decision aid method should come to one of the following conclusions:
   - a is preferred to b, or b is preferred to a;
   - a and b are indifferent;
   - a and b are incomparable.
4. Multiple criteria problems are not mathematically well–stated. Depending on the logic of the method and on the kind of additional information that is required, different results can be obtained. It is therefore important that the method be understandable by the decision maker. Black box effects should be avoided.
5. An appropriate method should not include any technical parameters having no economical significance.
6. The analysis of the conflicting aspects of the criteria must be available.
7. Finally, it is also important to have a clear interpretation of the weights of the criteria.

In our approach, as many aspects as possible will be addressed. However, a very deep understanding of the specific decision making problem is needed in order to fully address all these aspects.

5.3 Fuzzy preferences and orders

The following section describe some methods for specifying weight coefficients that will be used. Concerning the use of preferences, the most common situation found in the literature is given in (Chiclana, Herrera & Herrera-Viedma 1998):

For a finite set of alternatives $X = \{x_1, x_2, \ldots, x_{m_a}\}$, $(m_a \geq 2)$ and a set of experts $E = \{e_1, e_2, \ldots e_{m_e}\}$, $(m_e \geq 2)$, experts can represent their preferences in three different ways:

Preference ordering of the alternatives: an expert e_j provides his preferences on X as an individual preference ordering $O^j = \{o^j(1), \ldots, o^j(m_a)\}$ from the best to the worst.

Fuzzy preference relation: the expert's preferences on X is described by a fuzzy preference relation, $P^j \subseteq X \times X$ with membership function $\mu_{Pj} : X \times X \mapsto [0,1]$ where $\mu_{Pj}(x_{i_1}, x_{i_2}) = p^j_{i_1 i_2}$ denotes the preference degree of the alternative x_{i_1} over x_{i_2}.

Utility function: an expert e_j provides his preferences on X as a set of m_a utility values, $U^j = \{u^j_i \mid 1 \leq i \leq m_a\}$, where $u^j_i \in [0,1]$ represents the utility evaluation given by expert e_j to the alternative x_i.

However, this is not directly applicable to multi–objective optimisation problems of conceptual design, since it the not the problem of choosing *one* alternative from a finite set, but to choose the order of importance of *all* of them. Fuzzy preferences could support the estimation of relative importance (weights) of objectives in multi–objective optimisation problems.

5.3.1 Preference order

The *intensity of preference* or, shortly the *preference* of x over y is usually defined as $R(x,y) = f(g(x), Ng(y))$, where f is a non–decreasing function of both arguments and N is a strong negation (strictly decreasing, continuous and idempotent i.e. with $N(N(x)) = x$ property) (Fodor & Roubens 1994, p. 177).

For a given preference relation R, *strict preference* relation P and *indifference* relation I are defined in the following way:

$$P(x,y) = 1 - R(y,x), \quad I(x,y) = \min(R(x,y), R(y,x))$$

For a complete (fuzzy-)preference matrix, it is possible to define a complete order among the objects.

Relation R defines the directed valued graph $G = (A,R)$ and *entering score* $S_E(a,R)$, *leaving score* S_L and *net flow* $S_{L/E}(a,R)$ can be defined:

$$\begin{aligned} S_E(a,R) &\stackrel{\text{def}}{=} -\sum_{c \in A \setminus \{a\}} R(c,a) \\ S_L(a,R) &\stackrel{\text{def}}{=} \sum_{c \in A \setminus \{a\}} R(a,c) \\ S_{L/E}(a,R) &\stackrel{\text{def}}{=} \sum_{c \in A \setminus \{a\}} [R(a,c) - R(c,a)] \; (= S_E(a,R) + S_L(a,R)) \end{aligned}$$

and the corresponding orders (Fodor & Roubens 1994, p. 151). In the case when $R(a,b) + R(b,a) = 1$ for all a, b (probabilistic relation) i.e. $R(a,b) = P(a,b)$, they all give the same order[1], therefore, as the relations considered will satisfy this property, only the leaving score and the induced order will be used:

$$S_L(a,R) = \sum_{c \in A \setminus \{a\}} R(a,c) \tag{5.1}$$

$$a \geq_L b \text{ if and only if } S_L(a,R) \geq S_L(b,R) \tag{5.2}$$

[1] if n is the cardinality of A and R is probabilistic, then $S_L(a,R) - S_E(a,R) = n - 1$ and $S_{L/E}(a,R) = 2S_L(a,R) - (n-1)$.

Example 5.1 (From Fodor & Roubens 1994, p. 150) Wine experts give their preferences on five Médoc wines a, b, c, d and e using the following matrix:

$$R = \begin{bmatrix} 0.50 & 0.57 & 0.57 & 0.29 & 0.67 \\ 0.43 & 0.50 & 0.70 & 0.52 & 0.28 \\ 0.43 & 0.30 & 0.50 & 0.72 & 0.48 \\ 0.71 & 0.48 & 0.28 & 0.50 & 0.48 \\ 0.33 & 0.72 & 0.52 & 0.52 & 0.50 \end{bmatrix}$$

The table for the score is the following:

	S_L	S_E	$S_{L/E}$
a	2.10	-1.90	0.20
b	1.93	-2.07	-0.14
c	1.93	-2.07	-0.14
d	1.95	-2.05	-0.10
e	2.09	-1.91	0.18

giving the same complete order presented in Figure 5.1.

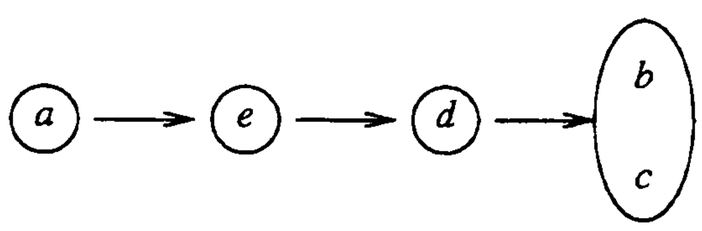


Figure 5.1. Partial order $a \geq e \geq d \geq b \approx c$

5.3.2 Our approach

Our approach is in a way similar to linguistic ranking methods (Chen, Hwang & Hwang 1992, p. 265) using the concept of linguistic variables and modifiers (Zadeh 1973, Zadeh 1975). For every two objectives the designer is asked to specify one of the following characterisations:

- Less important;
- Much less important;
- Equally important;
- More important;
- Much more important;
- Don't care.

The number of degrees of importance (such as slightly more important, vastly more important etc.) can be easily and in quite straightforward way extended but there is a danger of specifying too many levels losing the main advantage of this approach over quantitative methods.

Since k objectives require in the worst case $k(k-1)/2$ questions, the designer is first asked to identify the objectives of interest at this stage of the optimisation process and to specify the relative importance of these only. However, for clarity and brevity, this step will be skipped here.

Concerning objective importance escalation (from much less important to much more important), it is worth mentioning the work of Lootsma (1996, 1997*a*, 1997*b*) described in section 5.4.3, and the work of Greenwood, Hu & D'Abrosio (1996) described in section 5.4.4, page 68. Coello Coello (2000) gives a survey of use of preferences in optimisation.

Treatment of *don't care* relation

In the further text, *don't care* will be treated exactly as *equally important*. There are some psychological justification for this: if we do not care in respect of two objectives, then we also do not care which one is preferred. However, in future research it is intended to analyse this relation more closely. Alternatively, the approach by Vincke (1990, p. 113) can be used: "Given a partial preorder structure, it is always possible to replace the incomparabilities by preferences or indifferences in order to obtain a complete preorder structure"

Roy (1990*b*) distinguishes two types of hesitations:

weak preference If it could not be decided between $a \succ b$ and $a \approx b$, but being sure that $b \not\succ a$;

incomparability if it could not be decided between $a \succ b$ and $b \succ a$.

Roy further states that the hesitations may come from (Roy 1990*b*, p. 158):

- the existence in decision–maker's mind of zones of uncertainty, half–held belief, or conflicts and contradictions;
- the vaguely defined quality of the decision–maker D;
- the fact that the scientist who built the model ignores, in part, how D compares a and b;
- The imprecision, uncertainty, inaccurate determination of the maps $g(a)$ and $g(b)$ by means of which a and b are compared.

5.3.3 Formal definitions and properties of our preference relations

The following relations are introduced:

relation	intended meaning
$\approx$	is equally important
$\prec$	is less important
$\ll$	is much less important
$\neg$	is not important
!	is important

The required properties of these relations are (see Definition 3.2, page 19 for definition of reflexivity, transitivity etc):

- Relation $\approx$ is an *equivalence relation* (reflexive, symmetric and transitive):

$$x \approx x \tag{5.3}$$

$$x \approx y \Rightarrow y \approx x \tag{5.4}$$

$$x \approx y \wedge y \approx z \Rightarrow x \approx z \tag{5.5}$$

- Relations $\prec$ and $\ll$ are *strict orders* (irreflexive and transitive):

$$x \not\prec x \tag{5.6}$$

$$x \not\ll x \tag{5.7}$$

$$x \prec y \wedge y \prec z \Rightarrow x \prec z \tag{5.8}$$

$$x \ll y \wedge y \ll z \Rightarrow x \ll z \tag{5.9}$$

- Relation $\approx$ is *congruent* with $\ll$ and $\prec$:

$$x \prec y \wedge y \approx z \Rightarrow x \prec z \tag{5.10}$$

$$x \ll y \wedge y \approx z \Rightarrow x \ll z \tag{5.11}$$

- Relation $\ll$ is sub–relation of $\prec$:

$$x \ll y \Rightarrow x \prec y \tag{5.12}$$

- Miscellaneous properties:

$$!x \vee \neg x \tag{5.13}$$

$$!y \wedge \neg x \Rightarrow x \ll y \tag{5.14}$$

$$\neg x \wedge \neg y \Rightarrow x \approx y \tag{5.15}$$

$$x \prec y \wedge y \ll z \Rightarrow x \ll z \tag{5.16}$$

This set of properties is not a minimal one, as some of the properties could be inferred from the others, but it is not our task to specify the minimal set of properties.

Using predicates $\prec$ and $\ll$, predicates $\succ$ (is more important) and $\gg$ (is much more important) are then defined in the following way:

$$x \succ y \overset{\text{def}}{\Leftrightarrow} y \prec x \tag{5.17}$$

$$x \gg y \overset{\text{def}}{\Leftrightarrow} y \ll x \tag{5.18}$$

5.3.4 Some philosophical aspects on transitivity

Whether preferences should be transitive is a very much discussed question in the Logic of preference field (von Wright 1963, von Wright 1972, Hannson 1970, Martin 1963). Some authors argue that human preferences are not necessary transitive and give some very good examples of non–transitivity of preference relations (Fishburn 1991, Tversky 1969). There is a recent journal issue dedicated to the topic of transitivity of preferences (Gehrlein 1990).

However, in a conceptual design framework, being pragmatic, it could be argued that transitivity reduces the number of questions asked (if A is considered more important to B, and B more important to C, it is automatically inferred that A is more important to C). In most cases transitivity is used to avoid cyclic preferences i.e. $A \prec B$, $B \prec C$ and $C \prec A$ or indifferences such as $A \approx B$, $B \approx C$ and $A \prec C$.

One example for cyclic preferences is cited by Fishburn (1996):

> *Multiattribute comparisons provide a source of cyclic preferences for an individual. May (1954) asked 62 college students to make binary comparisons between hypothetical marriage partners x, y and z characterized by three attributes, intelligence, looks and wealth:*
>
> *x: very intelligent, plain, well off*
> *y: intelligent, very good looking, poor*
> *z: fairly intelligent, good looking, rich*
>
> *Seventeen of the 62 had the 2–to–1 majority cyclic pattern $x \succ y \succ z \succ x$; the other 45 had transitive preferences.*

As it could be seen, intransitivity of preferences could yield to contradictions. Contradiction is something that classical mathematical logic tries to avoid at all costs because starting from contradictions every possible conclusion is derivable. However, there are some branches of mathematical logic that try to avoid the contradiction problem:

The logic of relevance or entailment (Anderson & Belnap 1975, Dunn 1986) where derivation of conclusion from premises require that the premises are relevant to the conclusion i.e. statements such as (which is valid in classical mathematical logic):

$$(2 \neq 3) \Rightarrow \text{Last Fermat theorem is true}$$

are rejected since antecedent is not relevant to the consequent.

Paraconsistent logics (da Costa 1974, Asenjo & Tamburino 1975, Priest, Routley & Norman 1989) where contradictions are in a way localised not to affect the rest of the logic. They distinguish between the *triviality* of the logic (if it contains all formulae) and the *contradictoriness* of the logic (if there is some formula ϕ so that both ϕ and $\neg\phi$ are in contained in that logic). Those two concepts have the identical meaning in the classical case. The idea of paraconsistent logics is, even if there are some contradictions, to work on the largest possible non–contradictory part of it.

Modifications to our system of preferences and their inference rules to be based on one such logic are possible, but this is more a matter of further research.

Paper by Huylenbroeck (1995) suggests some method of conflict analysis and resolution. He uses several tests to decide in conflicting situations is there an indifference, weak or strong preference, or incomparability. This classification however depends on several numerical parameters, some very sensitive.

5.3.5 Group preferences

In the most real world cases, the designer is not just *one* person but a group of people usually with different and sometimes contradictory opinions relating to what is important and what is not. This causes already hard decision making process to be even harder.

Group preferences for accumulating designers preferences could be used, but there are some problems with their application, as stated by Arrow's Impossibility Theorem (Arrow 1951). The following quote from (Keeney, Raifa & Meyer 1976, p. 523) explain the problem of aggregating individuals' preferences:

"...Arrow's problem is roughly as follows: Given the ranking of a set of alternatives by each individual in a decision making group, what should the grouping ranking for these alternatives be? He postulated some very reasonable assumptions concerning the aggregation of individuals' rankings, and then he investigated their composite implications, which turned out to be quite surprising and disturbing. These assumptions are as follows.

Assumption A (Complete Domain) *There are at least two individual members in the group, at least three alternatives, and a group ordering is specified for all possible individual members' orderings.*

Assumption B (Positive Association of Social and Individual Orderings) *If the group ordering indicates alternative A is preferred to alternative B for a certain set of individual rankings, and if*

1. *the individual's paired comparison between alternatives other than A are not changed and*
2. *each individual's paired comparison between A and any other alternative either remains unchanged or is modified in A' favour*

then the group ordering must imply that A is still preferred to B.

Assumption C (Independence of Irrelevant Alternatives) *If an alternative is eliminated from consideration and the preference relations for the remaining alternatives remain invariant for all the group members, then the new group ordering for the remaining alternatives should be identical to the original group ordering for these same alternatives.*

Assumption D (Individual's Sovereignty) *For each pair of alternatives A and B there is some set of individual orderings such that the group prefers A to B.*

Assumption E (Nondictatorship) *There is no individual with the property that whenever he prefers alternative A to B, the group will also prefer A to B regardless of the other individuals' preferences.*

Arrow (1951) proved that there is no rule for combining the individual's rankings that is consistent with the seemingly innocuous Assumptions A through E. More precisely, we have the following theorem:

Theorem 5.1 *(Arrow's Impossibility Theorem) Assumptions A, B, C, D, and E are inconsistent.*

Thus, we conclude that our Decision Maker can find no procedure that can combine several individual's ranking of alternatives to obtain her ranking and that will simultaneously satisfy these five assumptions.

One interpretation of Arrow's Impossibility Theorem is that, in general, there is no procedure for combining individual rankings into a group ranking that does not explicitly address the question of interpersonal comparison of preferences."

In other words (Arrow 1950, p. 24): "If we exclude the possibility of interpersonal comparisons of utility, then the only method of passing from individual tastes to social preferences which will be satisfactory and which will be defined for a wide range of sets of individual orderings are either imposed or dictatorial".

It is difficult to give the exact conditions for Arrow's theorem because even Arrow sometimes mentions four (Arrow 1967) and sometimes five conditions (Arrow 1950, Arrow 1952). The following quote from (Arrow 1952, p. 51) give an explanation about four versus five conditions:

Condition 1 is implicit in the preceding discussion: the social choice function should be capable of defining an ordering for the collectivity, whatever the individual preference scales.

Condition 2 bears on the relation between the individual orderings and the collective ordering which results from them: the collective ordering should reflect positively *the individual welfare judgements. Suppose that for a given set of individual preference scales, society prefers x to y. Suppose that an individual modified his ordering by rising x on his list, which remains otherwise unchanged; it is clear that in this case we should require that society still prefers x to y ...*

The third condition that one should, in my opinion, impose on a social choice function is that the social choice among a set of candidates should depend on the individual preferences for those candidates and those candidates only.

...

There certainly exist trivial methods of aggregation which satisfy these three conditions. The first is the establishment of a collective ordering independent of any individual preference. Another consists in distinguishing one individual in the society and requiring that the society prefer one possibility to another whenever that individual does so: in short, a dictatorship. Neither of these methods answers the problem of aggregation in a truly democratic society. But do there exist other methods for constructing a social choice function? The answer is NO. *There are no methods, other than the two trivial ones just cited, for combining several individual preference systems and making of them a single preference system for society as a whole, at least if the three proposed conditions must be fulfilled.*

In order to avoid trivial cases, Arrow adds the following two conditions (Arrow 1952, p. 56):

Condition 4: It is impossible that the preferences of society be always in agreement with those of a single individual.

Condition 5: For every pair of possibilities x and y, there exists at least one system of individual preference orderings which causes x to be preferred to y.

Condition 4 excludes the case of "dictatorship". Condition 5 excludes the case where the social choice function would impose an ordering a priori, independent of individual preferences.

He further notes that Conditions 2 and 5 are here only to establish the Pareto Unanimity Principle. This principle could therefore have been substituted for the two conditions.

A detailed description of Arrow's theorem could also be found in (French 1986).

Arrow's impossibility theorem also has some opponents, as the following quote from Tullock's *Towards a Mathematics of Politics*, published by University of Michigan Press, 1967, demonstrate (quoted from (Arrow 1969)):

> *A phantom has stalked the classrooms and seminars of economics and political sciences for nearly fifteen years. The phantom, Arrow's General Impossibility Theorem, has been generally interpreted as proving that no sensible method of aggregating preferences exists. The purpose of this essay is to exorcise the phantom, not by disproving the theorem in its strict mathematical form, but by showing that it is insubstantial. I shall show that when a rather simple and probable type of interdependence is assumed among the preference functions of the choosing individuals, the problem becomes trivial if the number of voters is large. Since most cases which require aggregation of preferences involve large number of people, "Arrow problems" will seldom be of much importance.*

Of course, it must not be forgotten that Arrow's theorem deals with independent individual preferences (such as in voting), whereas in the designer group there is always interactivity so the engineers can sort out their differences (although it might cause the possible danger of dictatorship). A recent paper (Scott & Antonsson 1999) concludes that "...engineering design decision–making occupies a middle ground between decision with an idealized decision maker and decision by groups of fully autonomous citizens, and on this middle ground Arrow's Theorem has no detrimental consequences".

5.4 Description of the preference algorithm

Slightly simplified C code of the preference algorithm is given in appendix B, starting on page 131. The algorithm is as follows:

- Let the set of objectives be $O = \{o_1, \ldots, o_k\}$. Construct the equivalence classes $\{C_i \mid 1 \leq i \leq m\}$ of the equivalence relation $\approx$ so that $\cup_{i=1}^{m} C_i = O$ and $C_i \cap C_j = \emptyset$ for $i \neq j$. Choose one element x_i from each class C_i giving set $\mathcal{X} = \{x_1, \ldots, x_m\}$ where $m \leq k$. In the sequel we are going to work on set $\mathcal{X}$.
- Use the following valuation v:
  - If $a \ll b$ then $v(a) = \alpha$ and $v(b) = \beta$
  - If $a \prec b$ then $v(a) = \gamma$ and $v(b) = \delta$
  - If $a \approx b$ then $v(a) = v(b) = \varepsilon$[2]

  Note 1: Taking into account the intended meaning of the relations, it can be assumed that $\alpha < \gamma < \varepsilon = 1/2 < \delta < \beta$. As already mentioned, it is assumed that $\alpha + \beta = \gamma + \delta = 1$. Also note that α, β, γ, δ and ε need not be constant. Their order and property $\alpha + \beta = \gamma + \delta = 1$ is what matters. More discussion about choosing parameters is given in section 5.4.1 below.

  Note 2: Above valuation is the simplest possible case. More complex cases such as $v(x) = g(x, p, t)$ etc. are also possible, where p is some parameter, t is time (i.e. it would be possible to change our

[2]Since we work with class represents, this is only possible if $a = b$.

preferences as the time passes), and g is some real-valued function.

- Initialise two matrices R and R_a of size $m \times m$ to the identity matrix $\mathbf{E}_m$. They will be used in the following way:

$$\left.\begin{array}{lcllcl} x_i \ll x_j & \Leftrightarrow & R(i,j)=\alpha,\ R(j,i)=\beta & \Leftrightarrow & R_a(i,j)=0,\ R_a(j,i)=2 \\ x_i \prec x_j & \Leftrightarrow & R(i,j)=\gamma,\ R(j,i)=\delta & \Leftrightarrow & R_a(i,j)=0,\ R_a(j,i)=1 \\ x_i \approx x_j & \Leftrightarrow & R(i,j)=\varepsilon,\ R(j,i)=\varepsilon & \Leftrightarrow & R_a(i,j)=1,\ R_a(j,i)=1 \end{array}\right\} \tag{5.19}$$

Note: This valuation gives already the idea how to generalise preferences to have s stages instead of only 5 (from "much less important" to "much more important"): if x_i is (say) s' times more important the x_j, simple assignment $R_a(i,j)=s'$ and $R_a(j,i)=0$ etc. is all what is needed.

- Perform the following procedure:

Step1 For all $i \leq m$ and for all $j \leq m$ such that $j \neq i$ do

Step1.1 If $R_a(i,j)+R_a(j,i)=0$ then

- Ask whether $x_i \ll x_j$, $x_i \prec x_j$, $x_j \ll x_i$ or $x_j \prec x_i$
- Using equations (5.19) set $R_a(i,j)$ and $R_a(j,i)$ accordingly:
  - If $x_i \ll x_j$ set $R_a(i,j)=0$, $R_a(j,i)=2$.
  - If $x_i \prec x_j$ set $R_a(i,j)=0$, $R_a(j,i)=1$.
  - If $x_j \prec x_i$ set $R_a(i,j)=1$, $R_a(j,i)=0$.
  - If $x_j \ll x_i$ set $R_a(i,j)=2$, $R_a(j,i)=0$.

Step1.2 Using Warshall's algorithm (Warshall 1962), compute transitive closure of R_a (some modifications are necessary but straightforward):

for $k \in \{1,2,\ldots,m\}$

  for $j \in \{1,2,\ldots,m\}$

    for $i \in \{1,2,\ldots,m\}$

$$R_a(i,j) \leftarrow \min\{2, \max\{R_a(i,j), R_a(i,k) \cdot R_a(k,j)\}\};$$

The rationale behind the formula is the following:

- If one of $R_a(i,k)$, $R_a(k,j)$ is 0 (no path between i and k or between k and j), then $R_a(i,j)$ does not change, otherwise, use transitivity properties of $\prec$ and $\ll$ and (5.16).
- Term $\min(2,\cdot)$ is used so that $R_a(i,j) \in \{0,1,2\}$.

Step2 Using (5.19), calculate matrix R from R_a:

- If $R_a(i,j)=1$ and $R_a(j,i)=1$, set $R(i,j)=\varepsilon$ and $R(j,i)=\varepsilon$.
- If $R_a(i,j)=0$ and $R_a(j,i)=1$, set $R(i,j)=\gamma$ and $R(j,i)=\delta$.
- If $R_a(i,j)=1$ and $R_a(j,i)=0$, set $R(i,j)=\delta$ and $R(j,i)=\gamma$.
- If $R_a(i,j)=0$ and $R_a(j,i)=2$, set $R(i,j)=\alpha$ and $R(j,i)=\beta$.

– If $R_a(i,j) = 2$ and $R_a(j,i) = 0$, set $R(i,j) = \beta$ and $R(j,i) = \alpha$.

Other values are not allowed and indicate an error.

Step3 For each $x_i \in \mathcal{X}$ compute weight as a normalised leaving score:

$$w(x_i) = \frac{S_L(x_i, R)}{\sum_{x_j \in X} S_L(x_j, R)}.$$

and for each $y \in C_i$ set $w(y) = w(x_i)$.

Example 5.2 Let $O = \{o_1, \ldots, o_6, o_7\}$. Suppose that o_7 is a non–important objective and $o_1 \approx o_2$ and $o_3 \approx o_4$.
The classes of equivalence are:

$$C_1 = \{o_1, o_2\},\ C_2 = \{o_3, o_4\},\ C_3 = \{o_5\},\ C_4 = \{o_6\}$$

and $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$ where $x_i \in C_i$ for $1 \leq i \leq 4$. Let $R = R_a = \mathbf{E}_4$ — identity 4×4 matrix.

At the beginning of the run, matrix R_a is an identity matrix:

$$R_a = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}.$$

Suppose that the first question gives answer $x_2 \ll x_1$, than:

$$R_a = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and after transitive closure } R_a = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

the second question gives answer $x_3 \prec x_1$ with the matrix R_a:

$$R_a = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and after transitive closure } R_a = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and the third one $x_1 \prec x_4$. After 3 question situation is the following:

$$R_a = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \text{ and after transitive closure } R_a = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 2 & 1 & 1 \end{bmatrix}$$

Fourth question gives answer $x_2 \ll x_3$ and the following matrix R_a:

$$R_a = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 1 & 2 & 1 & 1 \end{bmatrix} \text{ and after transitive closure } R_a = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 1 & 2 & 1 & 1 \end{bmatrix}$$

Since for each pair (i, j): $R_a(i, j) + R_a(j, i) \neq 0$, there is enough information to construct the preference matrix R (without computing transitive closure 6 questions would be needed and additionally handling of potentially non–consistent answers). If $\alpha = 0.05, \beta = 0.95, \gamma = 0.35, \delta = 0.65$ and $\varepsilon = 0.5$, then

$$R = \begin{bmatrix} \varepsilon & \beta & \delta & \gamma \\ \alpha & \varepsilon & \alpha & \alpha \\ \gamma & \beta & \varepsilon & \gamma \\ \delta & \beta & \delta & \varepsilon \end{bmatrix} = \begin{bmatrix} 0.50 & 0.95 & 0.65 & 0.35 \\ 0.05 & 0.50 & 0.05 & 0.05 \\ 0.35 & 0.95 & 0.50 & 0.35 \\ 0.65 & 0.95 & 0.65 & 0.50 \end{bmatrix}.$$

Further,

$$S_L(x_1, R) = 1.95,\ S_L(x_2, R) = 0.15,\ S_L(x_3, R) = 1.65 \text{ and } S_L(x_4, R) = 2.25$$

and the order of importance is $x_2 \ll x_3 \prec x_1 \prec x_4$.

Weights are further calculated as

$$\begin{aligned} w_1 = w_2 = w(x_1) = w(o_1) = w(o_2) = 1.95/6 &= 0.325 \\ w_3 = w_4 = w(x_2) = w(o_3) = w(o_4) = 0.15/6 &= 0.025 \\ w_5 = w(x_3) = w(o_5) = 1.65/6 &= 0.275 \\ w_6 = w(x_4) = w(o_6) = 2.25/6 &= 0.375 \\ w_7 = w(o_7) &= 0.000 \end{aligned}$$

and after normalisation so that they sum to 1:

$$w_1 = w_2 = 0.2407, w_3 = w_4 = 0.0185, w_5 = 0.2037, w_6 = 0.2778, w_7 = 0.$$

5.4.1 Initial values of parameters and their influence

One valid question is how the valuation:

- If $a \ll b$ then $v(a) = \alpha$ and $v(b) = \beta$
- If $a \prec b$ then $v(a) = \gamma$ and $v(b) = \delta$
- If $a \approx b$ then $v(a) = v(b) = \varepsilon$

influences the obtained weights and if specifying parameters $\{\alpha, \beta, \gamma, \delta.\varepsilon\}$ is necessary, what is the advantage of our preference method in comparison to the original weights specification method?

The first advantage is in reducing the number of parameters: for n objectives using weights, $n-1$ parameters (numbers) need to be specified (since they sum to 1), whereas with preferences, as $\beta = 1-\alpha$, $\delta = 1-\gamma$ and $\varepsilon = 1-\varepsilon = 1/2$, only two need to be specified: α and γ. Further, the idea is that even parameters α and γ don't have to be specified every time and could be considered in the same way as, say, crossover type or mutation probability: once working, they do not need to be changed and could simply be hidden from the designer (user). The designer *can* change them if he is not satisfied with the results, but is not forced to do so.

Further, increasing the number of objectives, reduces the range of each weight so the influence of parameters α and γ becomes even less noticeable. This is the consequence of the following theorem:

Theorem 5.2 *Under the assumption that* $\alpha+\beta = \gamma+\delta = 2\cdot\varepsilon = 1$, *and that there are no equivalent or non–important objectives, the following inequality holds for weight factors* $w(x)$*:*

$$0 < \frac{2\mu}{m} \leq w(x) \leq \frac{2\nu}{m} < \frac{2}{m} \tag{5.20}$$

where m *is the number of objectives and* $\mu = \min\{\alpha, \beta, \gamma, \delta\}$, $\nu = \max\{\alpha, \beta, \gamma, \delta\}$.

Proof: Since $\alpha+\beta = \gamma+\delta = 2\cdot\varepsilon = 1$ and per definition $R(i,j)+R(j,i) = 1$ for all $1 \leq i, j \leq m$, the sum of elements in the preference matrix R (of size $m \times m$) is $m^2/2$, and therefore

$$S = \sum_x S_L(x,R) = \frac{m(m-1)}{2}$$

Further:

$$(m-1)\mu \leq S_L(x,R) \leq (m-1)\nu,$$
$$\frac{(m-1)\mu}{S} \leq w(x) \leq \frac{(m-1)\nu}{S},$$
$$\frac{2\mu}{m} \leq w(x) \leq \frac{2\nu}{m}$$

Since $0 < \mu \leq \alpha, \beta, \gamma, \delta \leq \nu < 1$, the inequality (5.20) is valid, and that completes the proof of the theorem.

For the more general case, with equivalent objectives, the following theorem is true:

Theorem 5.3 *Under the assumption that* $\alpha+\beta=\gamma+\delta=2\cdot\varepsilon=1$, *that there are* k *objectives, all of them important, and that there are* $m\leq k$ *equivalence classes (w.r.t. relation* $\approx$*), the following inequality holds:*

$$\frac{2\mu}{m+2(k-m)\nu}\leq w(x)\leq\frac{2\nu}{m+2(k-m)\mu} \tag{5.21}$$

where $\mu=\min\{\alpha,\beta,\gamma,\delta\}$, $\nu=\max\{\alpha,\beta,\gamma,\delta\}$. *As before,* $w(x)$ *denotes the weight assigned to the objective* x.

Proof: Let O be the original set of objectives of cardinality $|O|=k$, let $C=O/\approx$ be the set of class representatives of cardinality $|C|=m$ and let $C_1,\ldots,C_m$ be the corresponding equivalence classes. Denote R the fuzzy preference matrix (of size $m\times m$). Denote further:

$$S=\sum_{x\in C}S_L(x,R),\qquad \hat{S}=\sum_{x\in O}S_L(x,R)$$

From the proof of the theorem 5.2, the following is true:

$$S=\frac{m(m-1)}{2} \tag{5.22}$$

$$(m-1)\mu\leq S_L(x,R)\leq(m-1)\nu. \tag{5.23}$$

Then

$$\begin{aligned}\hat{S} &= \sum_{x\in O}S_L(x,R)\\ &= \sum_{x\in C}S_L(x,R)+\sum_{x\in(O\setminus C)}S_L(x,R)\\ &= S+\sum_{i=1}^{m}(|C_i|-1)S_L(x_i,R)\end{aligned}$$

where, per definition $x_i\in C_i$. Using (5.22) and (5.23), the following is obtained:

$$\hat{S}\leq\frac{m(m-1)}{2}+(m-1)\nu\sum_{i=1}^{m}(|C_i|-1)=\frac{m(m-1)}{2}+(k-m)(m-1)\nu \tag{5.24}$$

and similarly

$$\hat{S}\geq\frac{m(m-1)}{2}+(m-1)\mu\sum_{i=1}^{m}(|C_i|-1)=\frac{m(m-1)}{2}+(k-m)(m-1)\mu \tag{5.25}$$

Hence, using the property $0\leq a\leq b\wedge 0\leq c\leq d\Rightarrow a\cdot c\leq b\cdot d$, and inequalities (5.24) and (5.25):

$$w(x)=\frac{S_L(x,R)}{\hat{S}}\leq\frac{(m-1)\nu}{\frac{m(m-1)}{2}+(k-m)(m-1)\mu}=\frac{2\nu}{m+2(k-m)\mu} \tag{5.26}$$

and similarly

$$w(x)=\frac{S_L(x,R)}{\hat{S}}\geq\frac{(m-1)\mu}{\frac{m(m-1)}{2}+(k-m)(m-1)\nu}=\frac{2\mu}{m+2(k-m)\nu}. \tag{5.27}$$

Inequalities (5.26) and (5.27) give together (5.21) and that proves the theorem.

Note: Theorem 5.2 is a special case of the Theorem 5.3 for $k = m$.

Example 5.3 In example 5.2 on page 60, $m = 4$, $k = 6$, $\mu = \min\{\alpha, \beta, \gamma, \delta\} = 0.05$ and $\nu = \max\{\alpha, \beta, \gamma. \delta\} = 0.95$. Substituting into (5.21):

$$0.01282 \leq w(x) \leq 0.45238.$$

In the example in Section 6.5 on page 76, the parameters are $k = 13$, $m = 10$, $\mu = 0.05$ and $\nu = 0.95$, and the inequality (5.21) is reduced to:

$$0.00636 \leq w(x) \leq 0.18446.$$

Figure 5.2(a) shows the plot of the upper and lower bound for different ν (using $\mu = 1 - \nu$) for $k = 6$, $m = 4$, and Figure 5.2(b) gives the same plot for $k = 13$, $m = 10$.

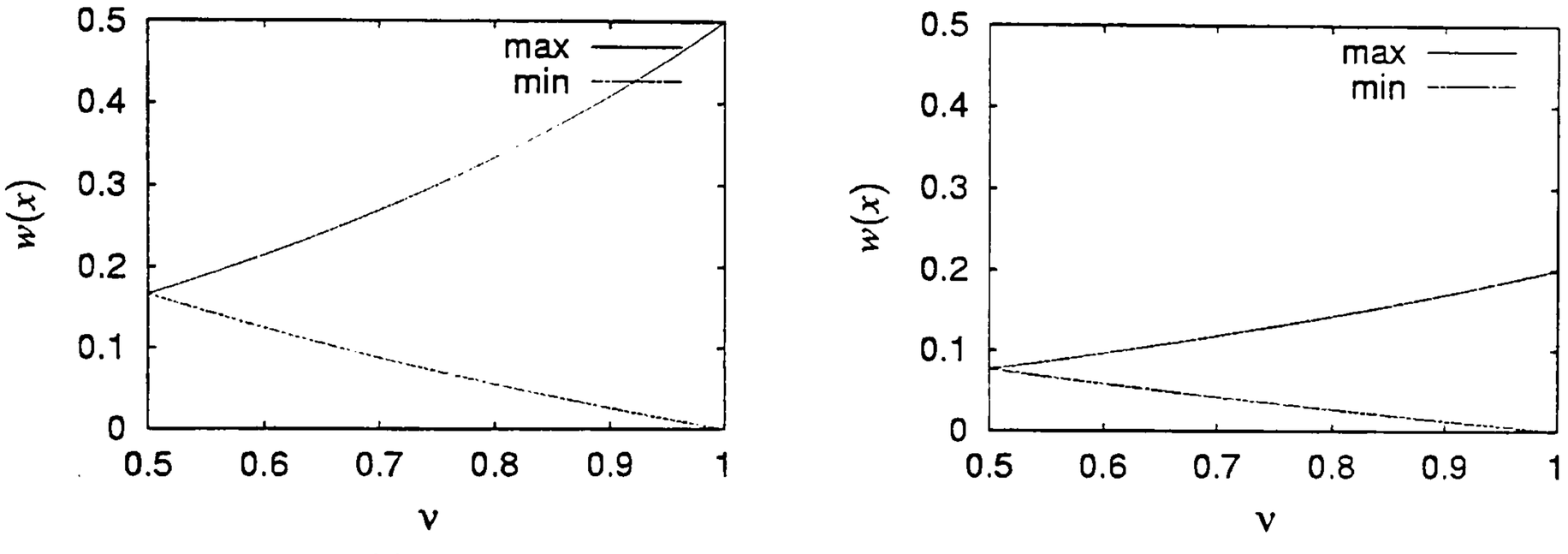


Figure 5.2. Lower and upper bound for (5.21) for different ν and (a) $k = 6$, $m = 4$ and (b) $k = 13$, $m = 10$.

The influence of parameters α and γ on the weights in Example 5.2 (page 60) is given in Table 5.1. Analytically, they can be computed using formulas:

$$w(o_1) = \frac{2-\alpha}{8+2\alpha}, w(o_3) = \frac{3\alpha}{8+2\alpha}, w(o_5) = \frac{1-\alpha+2\gamma}{8+2\alpha}, w(o_6) = \frac{3-\alpha-2\gamma}{8+2\alpha}$$

From the Table 5.1 it can be seen that the weights of objectives change relatively little with variation of preference parameters. Similar analysis could be performed for examples in sections 6.4 (page 73) and 6.5 (page 76) where the number of objectives is 8 and 13. In those cases there is even less variation in weights as predicted by Theorems 5.2 and 5.3.

5.4.2 Scalability and complexity issue

Another important question regarding preferences is "how does the preference method scale with increased number of objectives?". It is easy to see that in the worst case scenario for k objectives $n_q^*(k) = k \cdot (k-1)/2$

α	γ	$w(o_1)$	$w(o_3)$	$w(o_5)$	$w(o_6)$
0.05	0.05	0.2407	0.0185	0.1296	0.3519
0.05	0.15	0.2407	0.0185	0.1543	0.3272
0.05	0.25	0.2407	0.0185	0.1667	0.3148
0.05	0.35	0.2407	0.0185	0.2037	0.2778
0.05	0.45	0.2407	0.0185	0.2284	0.2531
0.10	0.10	0.2317	0.0366	0.1341	0.3293
0.10	0.15	0.2317	0.0366	0.1463	0.3171
0.10	0.25	0.2317	0.0366	0.1707	0.2927
0.10	0.35	0.2317	0.0366	0.1951	0.2683
0.10	0.45	0.2317	0.0366	0.2195	0.2439
0.15	0.15	0.2229	0.0542	0.1386	0.3072
0.15	0.25	0.2229	0.0542	0.1627	0.2831
0.15	0.35	0.2229	0.0542	0.1867	0.2590
0.15	0.45	0.2229	0.0542	0.2108	0.2349
0.25	0.25	0.2059	0.0882	0.1471	0.2647
0.25	0.35	0.2059	0.0882	0.1706	0.2412
0.25	0.45	0.2059	0.0882	0.1941	0.2176
0.35	0.35	0.1897	0.1207	0.1552	0.2241
0.35	0.45	0.1897	0.1207	0.1782	0.2011
0.45	0.45	0.1742	0.1517	0.1629	0.1854

Table 5.1. Influence of parameters α and γ on valuation in Example 5.2.

questions are necessary. However, transitivity and the other preference properties usually reduce that number. In order to find the average number of questions needed, the following simulation has been performed: for a given number of objectives k, give a random answer to each of the question and count the number of questions needed to construct the complete preference matrix. The tests have been performed for $k \in \{4,5,6,7,8,9,10,11,12,15,18,21,25,30,35,40,50,60,75,100\}$, each test repeated 100 times (with different random number seed) and average value $\tilde{n}_q(k)$ and its standard deviation $\sigma_q(k)$ have been calculated. The results are presented in Table 5.2 and plotted in Figure 5.3. For a comparison, the maximal theoretical number of questions n_q^* is presented in column 4 of the table. As it can be seen whereas $n_q^* \sim O(k^2)$, the following approximate formula holds true:

$$\tilde{n}_q(k) \approx (0.85 + \frac{1}{40} \cdot \sqrt{k}) \cdot k \cdot \ln k \tag{5.28}$$

i.e.

$$\tilde{n}_q(k) \sim O(\sqrt{k} \cdot k \cdot \ln k + k \cdot \ln k) \sim O(\sqrt{k} \cdot k \cdot \ln k) \tag{5.29}$$

The value of (5.28) is presented as the last column in Table 5.2.

Note: The worst case scenario (i.e. all n_q^* questions are needed) happens when all answers are in the 'same direction' (e.g. $y_1 \succ y_2, y_1 \gg y_3, \ldots, y_1 \succ y_k, y_2 \gg y_3, \ldots, y_2 \succ y_k, \ldots, y_{k-1} \succ y_k$) and, therefore, the transitivity cannot reduce the number of questions. In some cases asking the questions in different order might help: e.g. for order $x_1 \succ x_2 \succ x_3$ instead of asking $x_1?x_2, x_1?x_3, x_2?x_3$ which all give answer $\succ$ the questions $x_1?x_2, x_2?x_3$ are asked, and since both answers are $\succ$, preference $x_1 \succ x_3$ is automatically inferred. In this case, using 'depth first' traversing instead of 'breadth first' traversing is helpful. Of course, one does not know in advance the

k	$\tilde{n}_q$	$\sigma_q(k)$	n_q^*	(5.28)
4	4.36	0.6745	6	4.99
5	6.37	0.4852	10	7.29
6	8.89	0.5842	15	9.80
7	11.78	0.7047	21	12.48
8	14.83	1.5378	28	15.32
9	17.61	1.0040	36	18.29
10	20.84	2.0137	45	21.39
11	23.80	1.2144	55	24.61
12	26.48	1.1145	66	27.93
15	37.30	1.7552	105	38.46
18	50.04	2.6282	153	49.74
21	62.25	2.8793	210	61.67
25	79.51	2.6723	300	78.46
30	101.48	2.3376	435	100.70
35	125.68	3.0281	595	124.18
40	150.77	3.3900	780	148.75
50	202.42	3.2820	1225	200.84
60	257.78	3.2492	1770	256.38
75	345.49	4.7958	2775	345.35
100	501.31	5.6527	4950	506.57

Table 5.2 The number of questions $\tilde{n}_q(k)$ needed for k objectives with random answers. Average over 100 runs.

designer's opinions and the order of questions cannot be changed accordingly. But, dynamically changing the order of questions as they are being answered is a matter of further research. Also, more precise estimate of number of question needed, maybe using search theory techniques (Pronzato, Wynn & Zhigljavsky 2000), is planned.

5.4.3 Lootsma's work

There is a similar work done by Lootsma (1996, 1997*a*, 1997*b*). The notation from (Lootsma 1997*b*, p. 89) will be used in this subsection. Lootsma considers a finite number of alternatives A_j, $j = 1, \ldots, n$ under a finite number of performance criteria C_i, $i = 1, \ldots, m$ with the respective criterion weights c_i, $i = 1, \ldots, m$, assuming

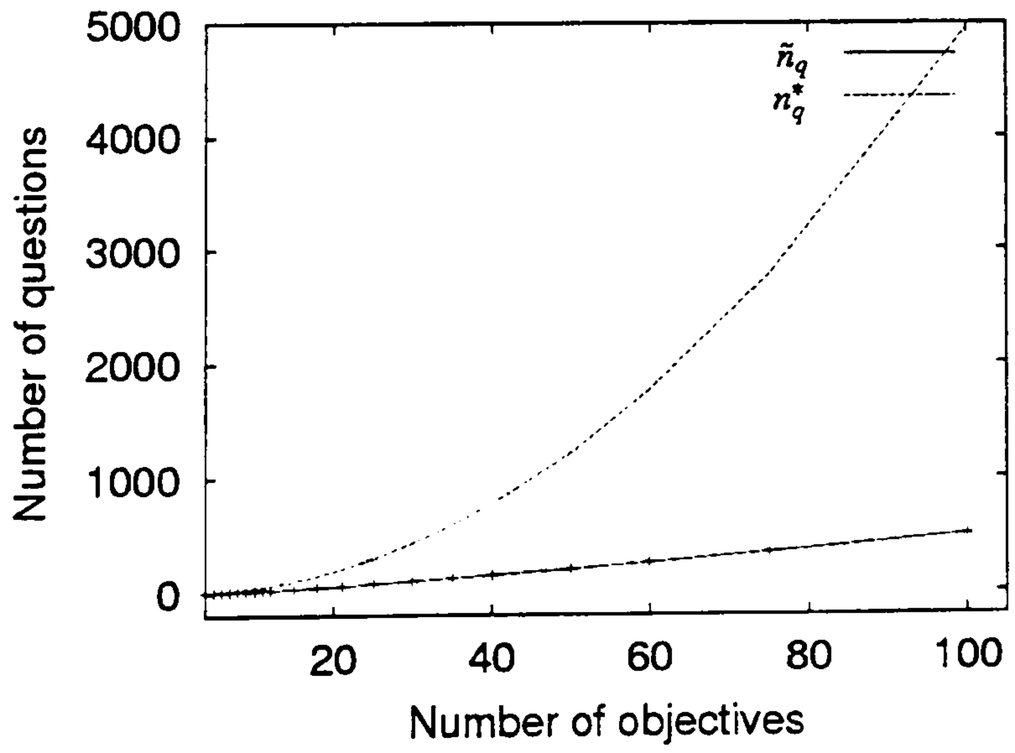


Figure 5.3 The number of questions $\tilde{n}_q(k)$ needed for k objectives with random answers and the maximal number $n_q^*(k)$. Average over 100 runs.

that $\sum_{i=1}^{m} c_i = 1$. The decision maker assesses the alternatives under each of the criteria separately and expresses his/her judgement of alternative A_j under criterion C_i by the assignment of the grade g_{ij}. In order to judge the overall performance of the alternative A_j under all criteria he calculates the final score $s_j = \sum_{i=1}^{m} c_i g_{ij}$.

Lootsma first splits the domain of objective P i.e. $[P_{\min}, P_{\max}]$ into 7 subregions $[P_i, P_{i+1}]$ where $P_i = P_{\min} + (P_{\max} - P_{\min})2^i/64$ for $0 \leq i \leq 6$, and assigns the desirability (price being cheap, somewhat more expensive, more expensive, etc, vastly more expensive) and grades (from 10 for excellent, to 4 for poor) accordingly. Between two objectives C_f and C_s belonging to the same domain, he then assigns the following gradations of relative importance:

16	C_f is vastly more important then C_s
8	C_f is much more important then C_s
4	C_f is more important then C_s
2	C_f is somewhat more important then C_s
1	C_f is as important as C_s
⋮	⋮
1/16	C_f is vastly less important then C_s

If h_i is the grade assigned to criterion C_i, then the ratio of the weights of two criteria C_i and C_j is $\sqrt{2}^{h_1 - h_2}$.

Example 5.4 (From Lootsma 1997*b*, p. 93): Suppose that the customer is buying a car and has the following 4 criteria:

criterion	grade	range
consumer price	9	20000 – 40000 Dfl
maximum speed	5	140–220 km/h
acceleration (0–100km)	7	8–20 s
cargo volume	6	200–2000 dm^3

Then the following weights for criteria have been calculated:

criterion	weight
consumer price	$\sqrt{2}^9/(\sqrt{2}^9 + \sqrt{2}^5 + \sqrt{2}^7 + \sqrt{2}^6) \approx 0.475$
maximum speed	$\sqrt{2}^5/(\sqrt{2}^9 + \sqrt{2}^5 + \sqrt{2}^7 + \sqrt{2}^6) \approx 0.119$
acceleration	$\sqrt{2}^7/(\sqrt{2}^9 + \sqrt{2}^5 + \sqrt{2}^7 + \sqrt{2}^6) \approx 0.238$
cargo volume	$\sqrt{2}^6/(\sqrt{2}^9 + \sqrt{2}^5 + \sqrt{2}^7 + \sqrt{2}^6) \approx 0.168$

For example, for the following table for two different cars and grades:

criterion	Fiesta 1.3 44kW (grade)	Mondeo 1.8 85kW (grade)
consumer price	25000 (6)	40000 (4)
maximum speed	153 (7.5)	195 (9.5)
acceleration	15.3 (4.5)	11.1 (6)
cargo volume	250 (5)	480 (7.5)

judging alternatives (Fiesta vs. Mondeo) the following results are obtained:

$$\begin{aligned} s(\text{Fiesta}) &= 0.475 \cdot 6 + 0.119 \cdot 7.5 + 0.238 \cdot 4.5 + 0.168 \cdot 5 = 5.7 \\ s(\text{Mondeo}) &= 0.475 \cdot 4 + 0.119 \cdot 9.5 + 0.238 \cdot 6 + 0.168 \cdot 7.5 = 5.7 \end{aligned}$$

5.4.4 Greenwood's work

In (Greenwood et al. 1996), the following technique for Pareto ranking is used:

> *"Our approach to using EAs for solving MOPs is unique in two respects. First, rather than asking a decision maker to explicitly rank the attributes, we ask for the ranking of a small number of candidate solutions to the MOP. This ranking of solutions implicitly defines a ranking of attributes.*

They base their method on *imprecisely specified multi–attribute value theory* (ISMAUT) (D'Ambrosio 1996) to perform imprecise ranking of attributes. The method is as follows (Greenwood et al. 1996, p. 442):

An imprecise multi–attribute value function for an alternative x has the following form:

$$V_x = \sum_k w_k v_k(a_k)$$

where w_k are the strictly positive weights (summing to 1), and $v_k(a_k)$ is the attribute value function (utility function) for attribute a_k. Function V_x is called imprecise, because weights w_k do not have specific value, but are constrained by preferences among attributes. By definition:

$$x \succ x' \Rightarrow V_x > V_{x'} \Rightarrow \sum_k w_k[v_k(a_k) - v_k(a'_k)] > 0$$

so, set of preferences define a constraint set W of weights w_k.

It follows that alternatives x and x'' can be compared by solving the following linear programming problem:

$$\text{Minimise (w.r.t. } w_k\text{): } \sum_k w_k[v_k(a''_k) - v_k(a_k)] \quad \text{subject to: } w_k \in W$$

Let

$$z = \min_{w_k} \sum_k w_k[v_k(a''_k) - v_k(a_k)], \qquad \bar{z} = \min_{w_k} \sum_k w_k[v_k(a_k) - v_k(a''_k)]$$

Then:

- If $z > 0$ then $x'' \succ x$;
- If $z \leq 0 \wedge \bar{z} > 0$ then $x \succ x''$;
- If $z \leq 0 \wedge \bar{z} \leq 0$ then $x \approx x''$.

Example 5.5 *(Greenwood et al. 1996, p. 443)* Suppose that the following table concerning attributes is given:

Alternatives	Attributes		
	$v_1(a_1)$	$v_2(a_2)$	$v_3(a_3)$
x_1	0.75	1.0	0.4
x_2	0.5	0.0	0.8
x_3	0.0	1.0	1.0
x_4	1.0	0.0	0.0

and suppose that the user has specified the preference $x_2 \succ x_1$. The question is how to rank x_2 and x_4.

Preference $x_2 \succ x_1$, defines the following set

$$W = \{(w_1, w_2, w_3) \mid -0.25w_1 - w_2 + 0.4w_3 > 0, w_i > 0, w_1 + w_2 + w_3 = 1\}$$

and after solving the linear programming problem: $z > 0$ and $x_2 \succ x_4$.

One of the problems with this approach is mentioned by the authors: so far they assume that it is always possible to find the solution to linear programming problems. However, if the user specify a contradictory set of preferences, there is no solution. To avoid this problem, they use an exponential time algorithm for identifying the inconsistent preference statements, increasing the complexity of the algorithm. Another problem with this approach is that it is not completely obvious when and how often the designer needs to specify his preferences.

5.4.5 Conclusion

In this chapter a new preference method is introduced, some theoretical results proved and various problems, methods and approaches discussed. The fact that there are several different approaches to preferences (not just two mentioned above) proves the general usefulness of preferences in multi–objective design and optimisation. The question which of the methods is the best is however very hard to answer since they are not directly comparable and they serve different objectives. We do believe that our proposed method is simple and elegant and does not suffer from drawbacks of other methods (e.g. inconsistency problem of Greenwood's method). Its usefulness is demonstrated in the next chapters where it is integrated with several optimisation methods, some of which are developed here, some developed elsewhere.

CHAPTER 6

Applications of Preferences

The concept of preferences and of the relative importance of the objectives, developed in the previous chapter, can be integrated with Genetic Algorithms in at least two different situations:

1. weighted sum based optimisation, and
2. weighted Pareto optimisation.

In both cases the preference method is used to calculate the weights as required by the optimisation method.

6.1 Weighted sum based optimisation

Weighted sum based optimisation is described in section 3.2.1, page 21. The preference method integrated with weighted sums has a significant advantage over the traditional weighted sum based optimisation methods since the user doesn't have to express the weights quantitatively but qualitatively (within a few categories) which is much more user friendly. Figure 6.1 shows the optimisation results using different preferences on weighted sum optimisation of BAe function. It demonstrates that the choice of preferences has a great influence on the results obtained: e.g. the preference setting $SEP2 \ll FR$ generates the solution $(0.0014, 7909)$ – the aeroplane with very good ferry range but difficult to steer, whereas the preference setting $SEP2 \gg FR$ generates the solution $(82.89, 4920)$ representing the aeroplane easy to control, but with much shorter ferry range than in the first case.

6.2 Pareto optimisation based methods

Weighted Pareto method is described in section 3.5.1, page 33. Weight vector w in w–Pareto front could either be specified directly by the designer or it can be calculated from his preferences which would help the designer to work in more qualitative terms without the burden to reason if the weight should be set of 0.1 or to 0.09 and how is it going to affect his search.

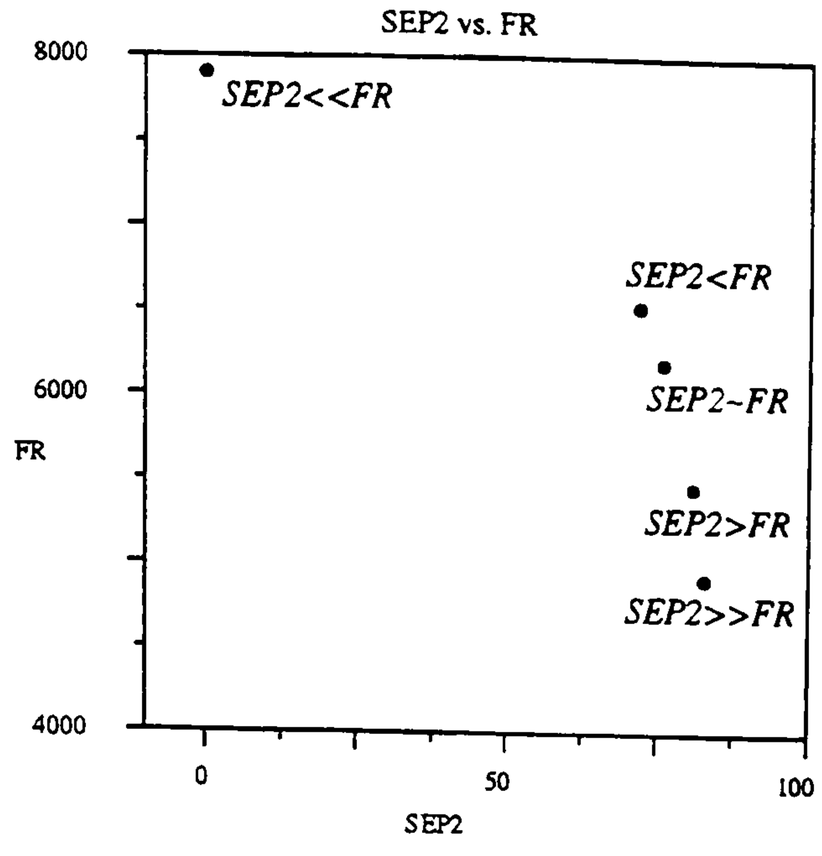


Figure 6.1. The influence of preference settings on the BAe function.

This section describes applications of the weighted Pareto method. First, an example of a simple test function is given, and then, in section 6.3, the real world problem from cooperation with British Aerospace (BAe) is presented.

6.2.1 Simple function example

As a simple test function that can immediately show some features of w–Pareto front, a two–dimensional function $F_s(x_1, x_2) = (f_1(x_1, x_2), f_2(x_1, x_2))$ is maximised (i.e. $n = k = 2$) where:

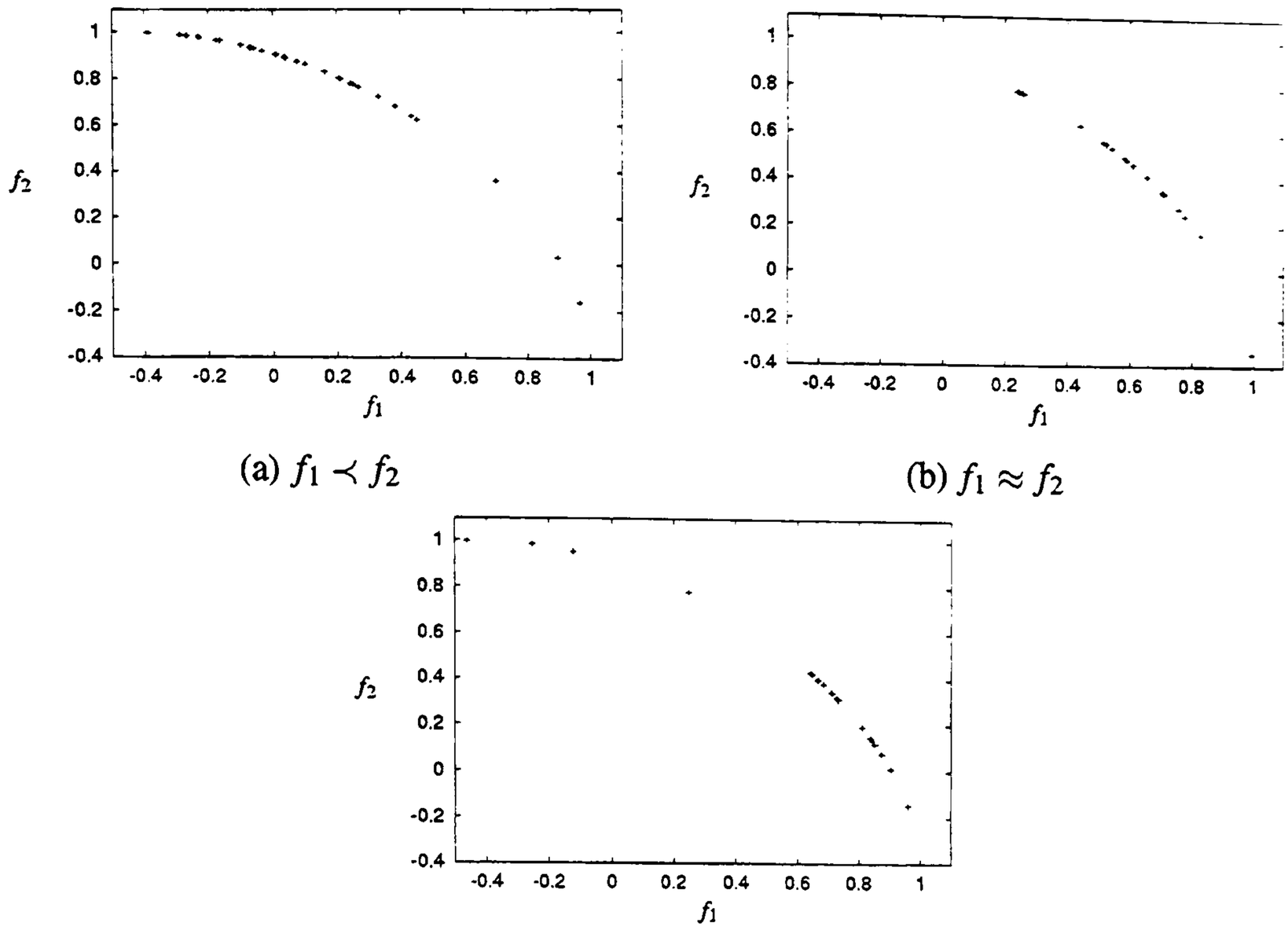
$$\begin{aligned} f_1(x_1, x_2) &= \sin(x_1^2 + x_2^2 - 1) \\ f_2(x_1, x_2) &= \sin(x_1^2 + x_2^2 + 1) \end{aligned} \tag{6.1}$$

for $x_1, x_2 \in [0, 3\pi/4]$. Function f_1 has maximum for $x_1^2 + x_2^2 = \pi/2 + 1 \approx 2.5708$ and f_2 has maximum for $x_1^2 + x_2^2 = \pi/2 - 1 \approx 0.5708$ – so they don't have common maximum. Using different preferences, i.e $f_1 \prec f_2$, $f_1 \approx f_2$ and $f_1 \succ f_2$, three different vectors w are obtained, and three different w–Pareto fronts are showed in Figure 6.2(a)–(c). Results were obtained running GA (described in more details in section 3.3 and with a parameter setting as in section 4.5) for 15 generations. These graphs show very clear separation of Pareto fronts obtained using different preferences.

6.3 BAe function and GA/Pareto optimisation

The British Aerospace (BAe) design problem is introduced in section 3.3, page 27. Briefly (at the moment, as the complexity of the model is constantly increasing), there are 9 input variables $x_1, \ldots, x_9$ and 13 output objectives $y_1, \ldots, y_{13}$ to optimise simultaneously. The interaction between y_4 (specific excess power SEP2) and y_9 (ferry range FR) is specially interesting as they strongly conflict.

However, as already mentioned, the BAe design problem is not only an optimisation problem, actually,



(a) $f_1 \prec f_2$

(b) $f_1 \approx f_2$

(c) $f_1 \succ f_2$

Figure 6.2. Different parts of Pareto front of function (6.1) for different preferences.

optimisation is a rather small part of it. The problems of conceptual design relate to the fuzzy nature of initial design concepts and the many different variants that engineers wish to try. Computers should be able to help them exploration of those variants whilst also suggesting some others (Cvetković et al. 1998, Parmee 1997, Parmee 1998*a*). Therefore, interaction with the designer (team) is very important. Our goal is to assist designers in the preliminary design process phase (more in the sense of MCDA (multiple criteria decision aid), than MCDM (multiple criteria decision making) (Bana e Costa 1990*b*, Carlsson 1996). Figure 6.3 shows a schema of such a system where the designer *is* an actual part of a design system.

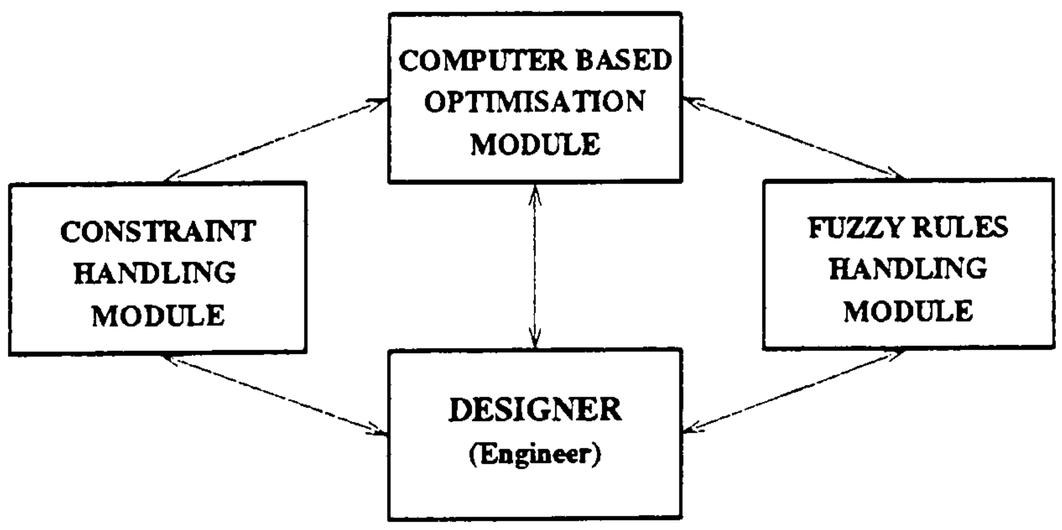


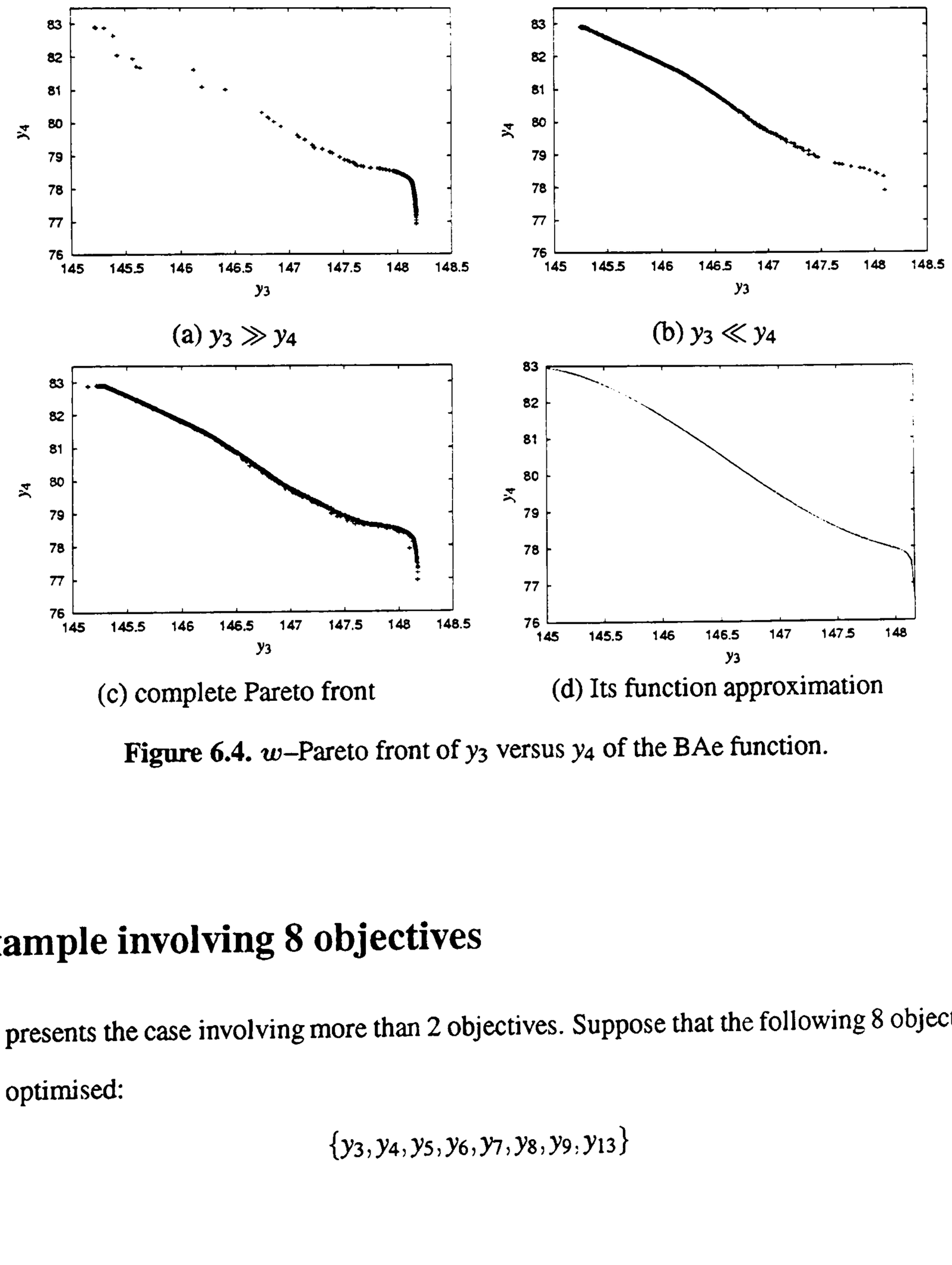
Figure 6.3. Schema of an engineering design system.

Using the results from chapter 4, and as explained in chapter 3, the core of the genetic algorithm used is based on the Breeder Genetic Algorithm (Mühlenbein & Schlierkamp-Voosen 1993*b*). It utilises genetic operators suitable for real valued chromosomes (arithmetic crossover, exponential mutation etc.), and is adapted to use techniques for multi–objective optimisation.

Figure 6.4(a)&(b) shows two w–Pareto fronts of y_3 versus y_4 for different preferences i.e. for different

weights, whereas Figure 6.4(c) shows the shape of the complete Pareto front. This shape was obtained using the least square method of fitting Pareto front data points. The function obtained was $y = -803573 + 0.0158774/(x - 148.185) + 16458.4x - 112.338x^2 + 0.255556x^3$ and is plotted in Figure 6.4(d).

Figure 6.5(a)&(b) show two w–Pareto fronts of y_4 versus y_9 for different preferences i.e. for different weights, whereas Figure 6.5(c) shows the shape of the complete Pareto front. The shape was obtained generating large number of Pareto front points and searching for the best functional approximation which in this case was the function $y = 7216 - 28925/(83 - 0.9x) + 117\sqrt{83 - 0.9x}$. It can be immediately seen that by varying Pareto threshold and the weights of each objective, different Pareto fronts can be obtained. Exact relationship and applicability will be investigated in further research. Knowing the behaviour of the Pareto front with respect to those parameters would enable us to vary the parameters during the genetic algorithm run to identify those parts of the Pareto front that are of special interest (considering the density of points in given regions etc.). Note that during the run some Pareto front points would be generated that are out of these parts, but postprocessing the obtained data by simply removing all points that are too far away from the front's "centre of gravity", the desired front can be obtained. This postprocessing method is explained in section 6.6, page 77.



(a) $y_3 \gg y_4$

(b) $y_3 \ll y_4$

(c) complete Pareto front

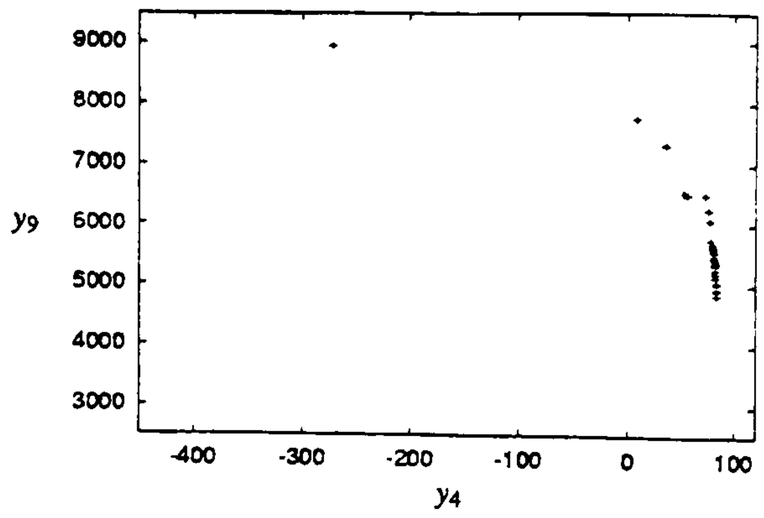
(d) Its function approximation

Figure 6.4. w–Pareto front of y_3 versus y_4 of the BAe function.

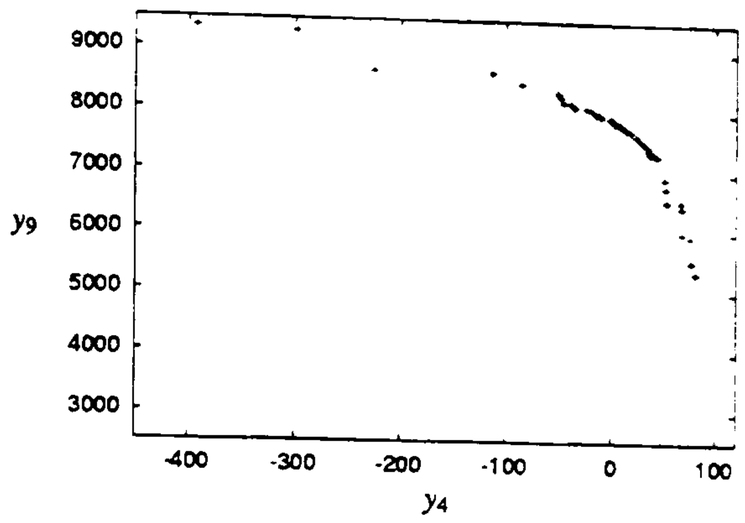
6.4 Example involving 8 objectives

This section presents the case involving more than 2 objectives. Suppose that the following 8 objectives of BAe function are optimised:

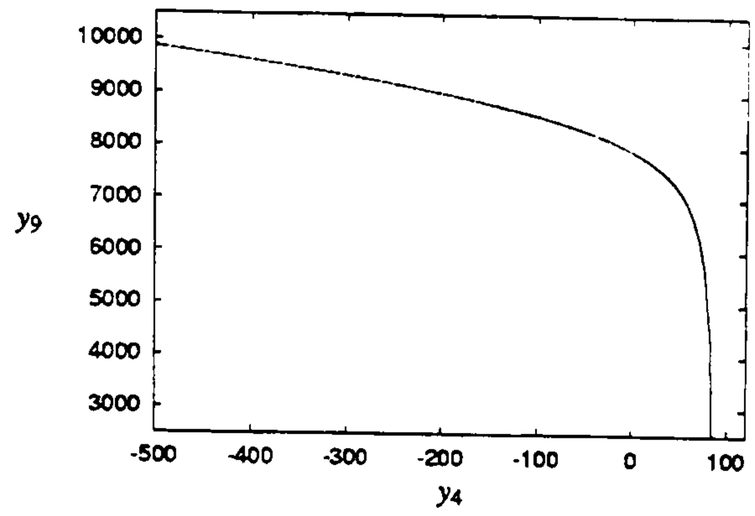
$$\{y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{13}\}$$



(a) $y_4 \succ y_9$



(b) $y_4 \ll y_9$



(c) complete Pareto front

Figure 6.5. w–Pareto front of y_4 versus y_9 of the BAe function for different preferences.

and that the following answers concerning preferences (those are the exact questions the program asks — 6 questions only for 5 distinctive objectives) are specified:

$$y_3 \approx y_4 \quad y_5 \approx y_6 \quad y_7 \approx y_8$$
$$y_3 \succ y_5 \quad y_3 \gg y_7 \quad y_3 \prec y_9 \quad y_3 \succ y_{13}$$
$$y_5 \succ y_7 \quad y_5 \prec y_{13}$$

This results in the following preference order:

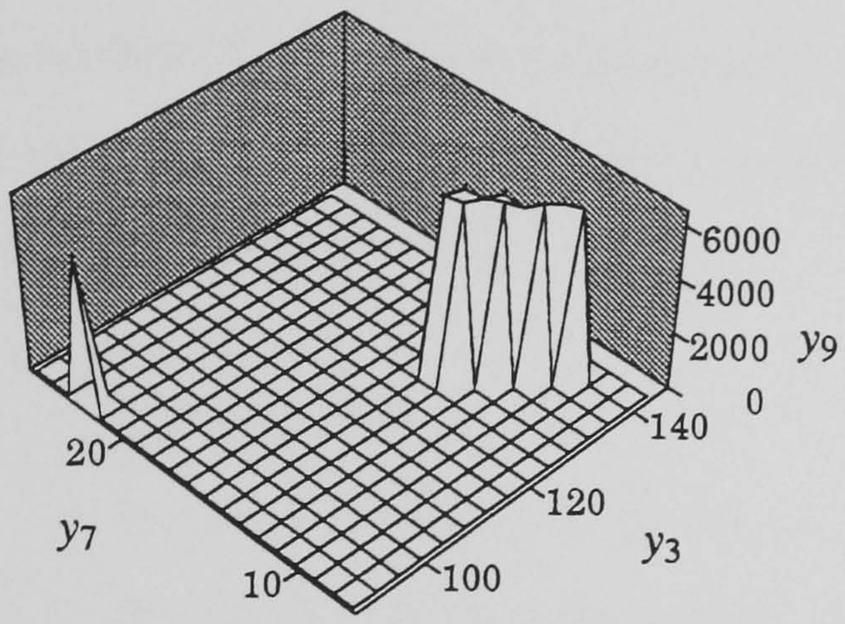
$$y_9 \succ y_3 \approx y_4 \succ y_{13} \succ y_5 \approx y_6 \gg y_7 \approx y_8$$

and, using the same valuation as in Example 5.2 on page 60, the following weights:

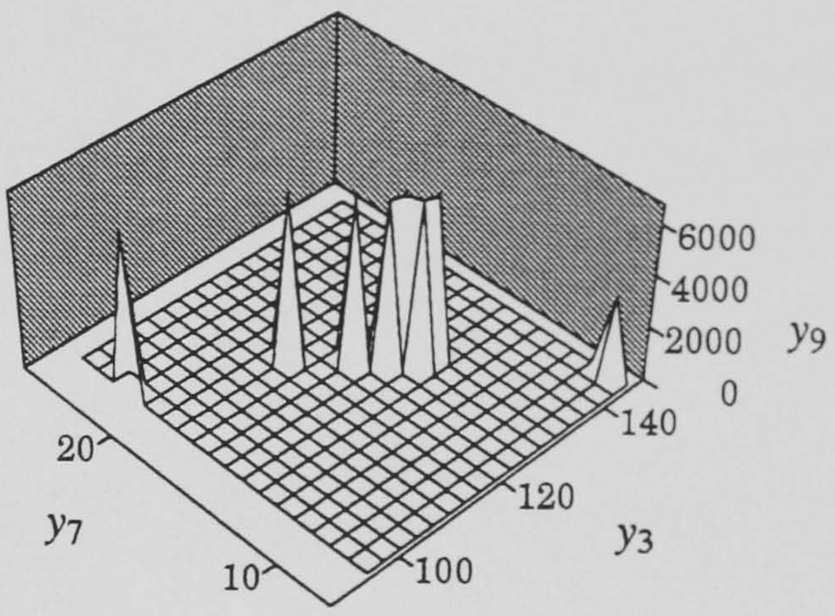
$$w_3 = w_4 = 0.1722, w_5 = w_6 = 0.1126, w_7 = w_8 = 0.053, w_9 = 0.1921, w_{13} = 0.1325$$

Performing weighted Pareto GA optimisation (with Pareto set size limited to 400) gives as a result an 8–dimensional hypersurface. Some of its 3D slices are presented in Figure 6.6. (page 75). For the comparison, right hand side contains the same 3D slice without preferences (i.e. using standard Pareto method).

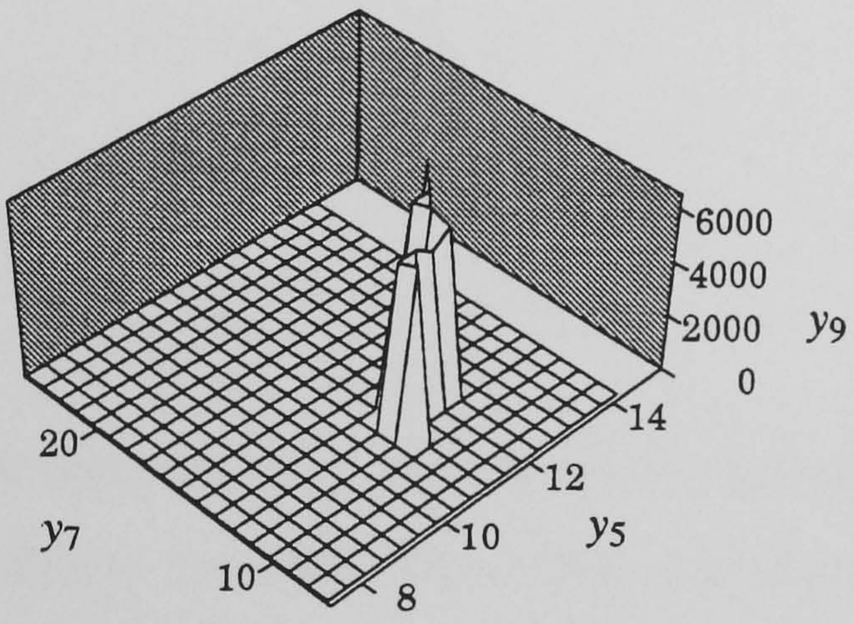
Note that these are just 3–dimensional slices of an 8–dimensional surface. There is still a problem of displaying n–dimensional surface for $n > 4$. Although a very small part of the 8D Pareto front is displayed, the shift towards the smaller values of y_7 in Figure 6.6(a) can nevertheless be noticed as the objective y_7 is the least important of all. Similarly, the values of y_5 are less in Figure 6.6(a) than in Figure 6.6(b). This can be



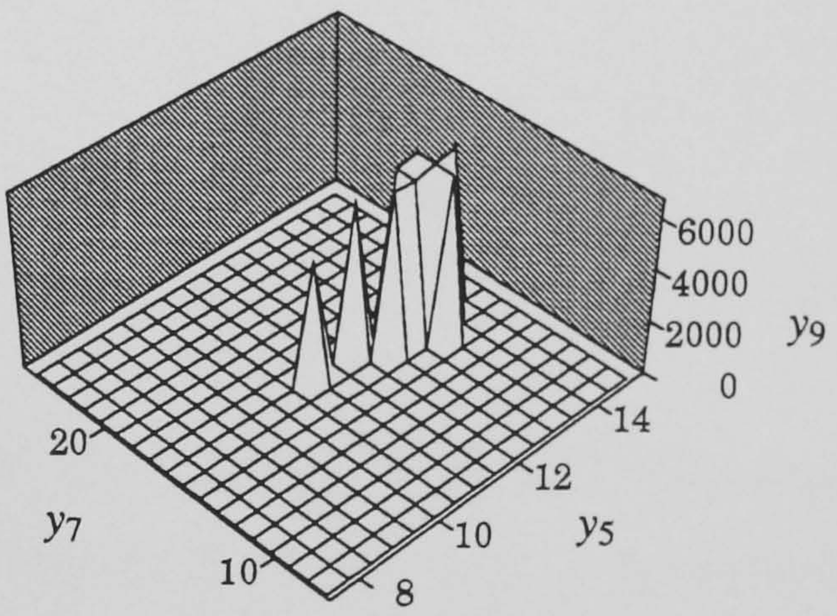
(a) $y_9 \succ y_3 \gg y_7$



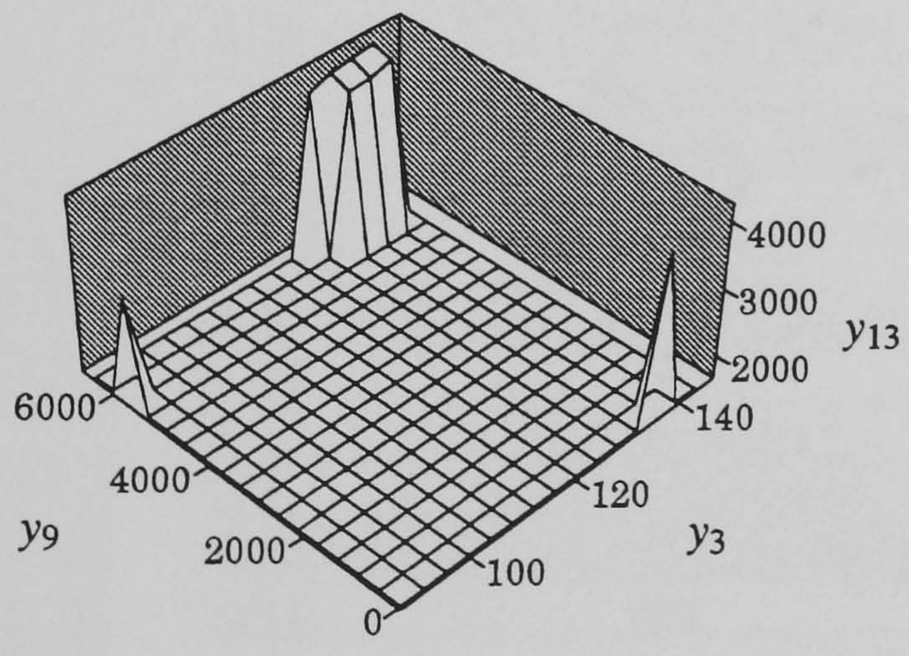
(b) $y_3 \approx y_7 \approx y_9$



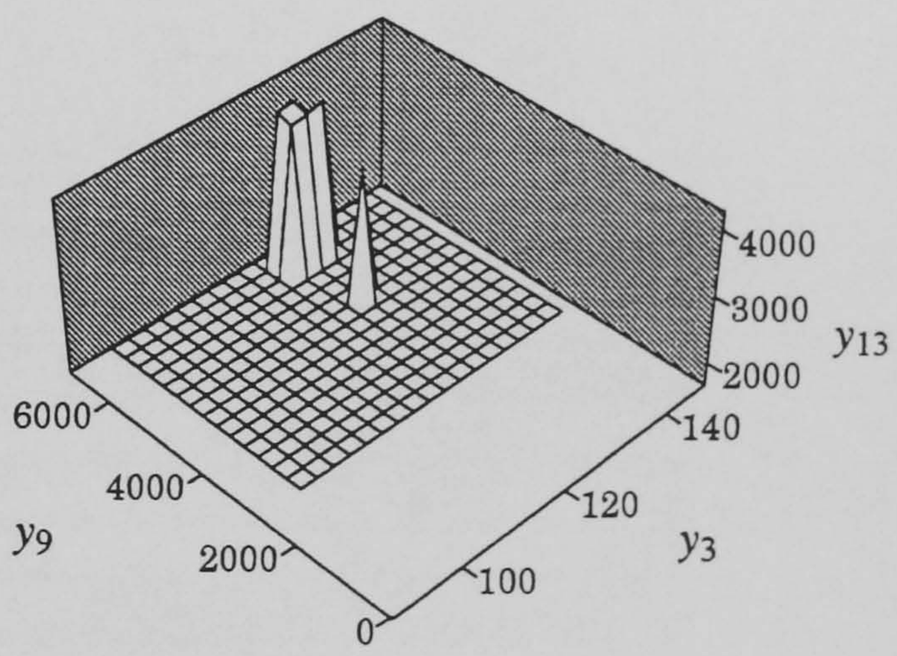
(c) $y_9 \succ y_5 \gg y_7$



(d) $y_5 \approx y_7 \approx y_9$



(e) $y_9 \succ y_3 \succ y_{13}$



(f) $y_3 \approx y_9 \approx y_{13}$

Figure 6.6. 3D slices of Pareto fronts (y_3, y_7, y_9), (y_5, y_7, y_9) and (y_3, y_9, y_{13}) with and without preferences.

easily explained by noticing that the weight factor for y_9 (0.1921) is almost twice the weight of y_5 (0.1126) and almost four times the weight of y_7 (0.053). A similar effect could be noticed in other Pareto front slices.

6.5 Example with 13 objectives

Another large experiment involves *all* 13 current objectives $\{y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10}, y_{11}, y_{12}, y_{13}\}$ of the BAe function. Suppose that the designer has the following preferences (again, as required by the program — 24 questions for 13 objectives (instead of theoretically maximal 78), or 21 question for 10 non–equivalent objectives instead of 45):

$$y_3 \approx y_4,\ y_5 \approx y_6,\ y_7 \approx y_8$$

$$y_1 \succ y_2,\ y_1 \prec y_3,\ y_1 \succ y_5,\ y_1 \succ y_7,\ y_1 \ll y_9,\ y_1 \succ y_{10},\ y_1 \gg y_{11},\ y_1 \succ y_{12},\ y_1 \prec y_{13}$$

$$y_2 \prec y_5,\ y_2 \succ y_7,\ y_2 \prec y_{10},\ y_2 \prec y_{11},\ y_2 \succ y_{12}$$

$$y_3 \ll y_9,\ y_3 \prec y_{13}$$

$$y_5 \succ y_{10},\ y_5 \succ y_{11}$$

$$y_7 \succ y_{12}$$

$$y_9 \succ y_{13}$$

$$y_{10} \gg y_{11}$$

If the objectives are ordered manually, by splitting the set of all objectives into these less than y_i and these greater than y_i, the results obtained (for non–equivalent objectives $\{y_1, y_2, y_3, y_5, y_7, y_9, y_{10}, y_{11}, y_{12}, y_{13}\}$) are presented in Table 6.1.

y_i	More important than y_i	Less important than y_i
y_1	$\{y_3, y_9, y_{13}\}$	$\{y_2, y_5, y_7, y_{10}, y_{11}, y_{12}\}$
y_2	$\{y_1, y_3, y_5, y_9, y_{10}, y_{11}, y_{13}\}$	$\{y_7, y_{12}\}$
y_3	$\{y_9, y_{13}\}$	$\{y_1, y_2, y_5, y_7, y_{10}, y_{11}, y_{12}\}$
y_5	$\{y_1, y_3, y_9, y_{13}\}$	$\{y_2, y_7, y_{10}, y_{11}, y_{12}\}$
y_7	$\{y_1, y_2, y_3, y_5, y_9, y_{10}, y_{11}, y_{13}\}$	$\{y_{12}\}$
y_9	$\emptyset$	$\{y_1, y_2, y_3, y_5, y_7, y_{10}, y_{11}, y_{12}, y_{13}\}$
y_{10}	$\{y_1, y_3, y_5, y_9, y_{13}\}$	$\{y_2, y_7, y_{11}, y_{12}\}$
y_{11}	$\{y_1, y_3, y_5, y_9, y_{10}, y_{13}\}$	$\{y_2, y_7, y_{12}\}$
y_{12}	$\{y_1, y_2, y_3, y_5, y_7, y_9, y_{10}, y_{11}, y_{13}\}$	$\emptyset$
y_{13}	$\{y_9\}$	$\{y_1, y_2, y_3, y_5, y_7, y_{10}, y_{11}, y_{12}\}$

Table 6.1. Ordering of objectives

This gives the following order:

$$y_9 \succ y_{13} \succ y_3 \approx y_4 \succ y_1 \succ y_5 \approx y_6 \succ y_{10} \gg y_{11} \succ y_2 \succ y_7 \approx y_8 \succ y_{12}$$

and the weights vector

$$(0.1003, 0.0334, 0.1054, 0.1054, 0.0951, 0.0951, 0.0283, 0.0283, 0.1414, 0.09, 0.0386, 0.0231, 0.1157).$$

The optimisation run was performed using GA with population size 500 for 500 generations and the size of Pareto front was limited to 1000. Objectives $\{y_1, y_2, y_{10}, y_{11}, y_{12}\}$ are minimised and all other are maximised. Each objective is normalised to $[0, 1]$ range. The obtained graphs are similar to these shown in Figure 6.6, except that the effect of preferences is less visible since the number of objectives has been increased.

6.6 Restricting Pareto front

Even though the Pareto front obtained using preferences is a subset of the whole Pareto front, the search process still finds isolated points further away from the region of interest, as illustrated in Figure 6.8(a) (obtained by performing Pareto optimisation on y_4 and y_9 using $y_4 \ll y_9$ preference). For example the point $(-334.77, 9399.75)$ (the most left point of the front) is very far away from the actual region of interest. The points obtained can be restricted using the concepts of the centre of gravity and the average distance in the following way:

Definition 6.1 *For a given set* $S = \{\boldsymbol{y}^{(i)} = (y_1^{(i)}, \ldots, y_m^{(i)}) \mid 1 \leq i \leq n_S, \} \subseteq \mathbf{R}^m$ *of size* n_S, *its* centre of gravity *is defined as* $\boldsymbol{y}_S^c = (y_1^c, \ldots, y_m^c)$ *where*

$$y_i^c = \frac{1}{n_S} \cdot \sum_{j=1}^{n_S} y_i^{(j)} \tag{6.2}$$

and its average distance *is calculated as* d_S*:*

$$d_S = \frac{1}{n_S} \sum_{i=1}^{n_S} \|\boldsymbol{y}_S^c - \boldsymbol{y}^{(i)}\|_2 = \frac{1}{n_S} \sum_{i=1}^{n_S} \sqrt{\sum_{j=1}^{m} (y_j^{(i)} - y_j^c)^2} \tag{6.3}$$

The restricting algorithm is simple: after computing average distance d_S, and centre of gravity $\boldsymbol{y}_S^c$, all points in the Pareto front with the distance greater than $\alpha_c \cdot d_S$ from the point $\boldsymbol{y}_S^c$ are removed, for some positive real–valued parameter α_c.

Figure 6.8(b)–(d) show the results obtained restricting Pareto front of $y_4 \ll y_9$ (presented in Figure 6.8(a)) for $\alpha_c \in \{1, 2, 3\}$. Figure 6.7 shows the size of restricted Pareto front (i.e. percentage of the unrestricted Pareto front size) as a function of α_c. Our experience shows that the values around $\alpha \approx 2$ for 2D Pareto front give the best results.

6.7 Combining preferences with cooperative optimisation

One of the further applications of preferences is the distributive cooperative Genetic Algorithms described in (Parmee & Watson 1999, Parmee, Cvetković, Watson & Bonham 2000). Ignoring the Taguchi method (Peace 1993) used, and concentrating only on the cooperative part, the main idea of this kind of optimisation is

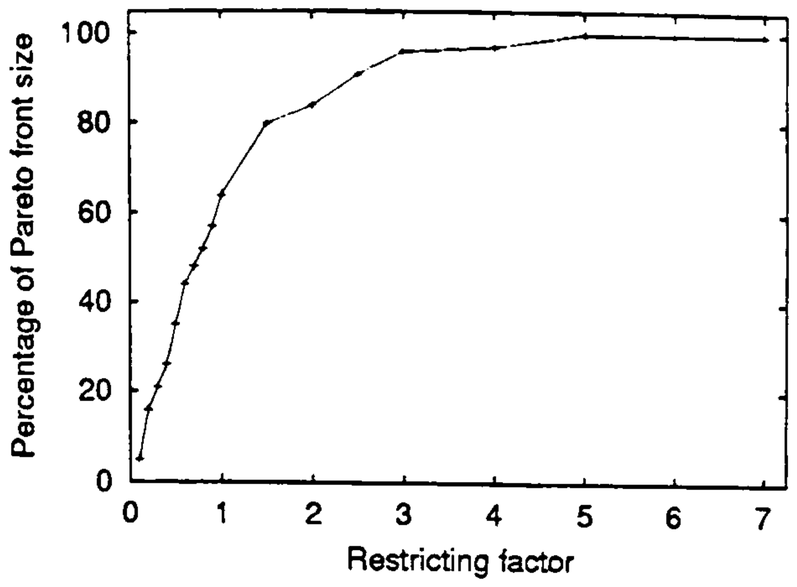
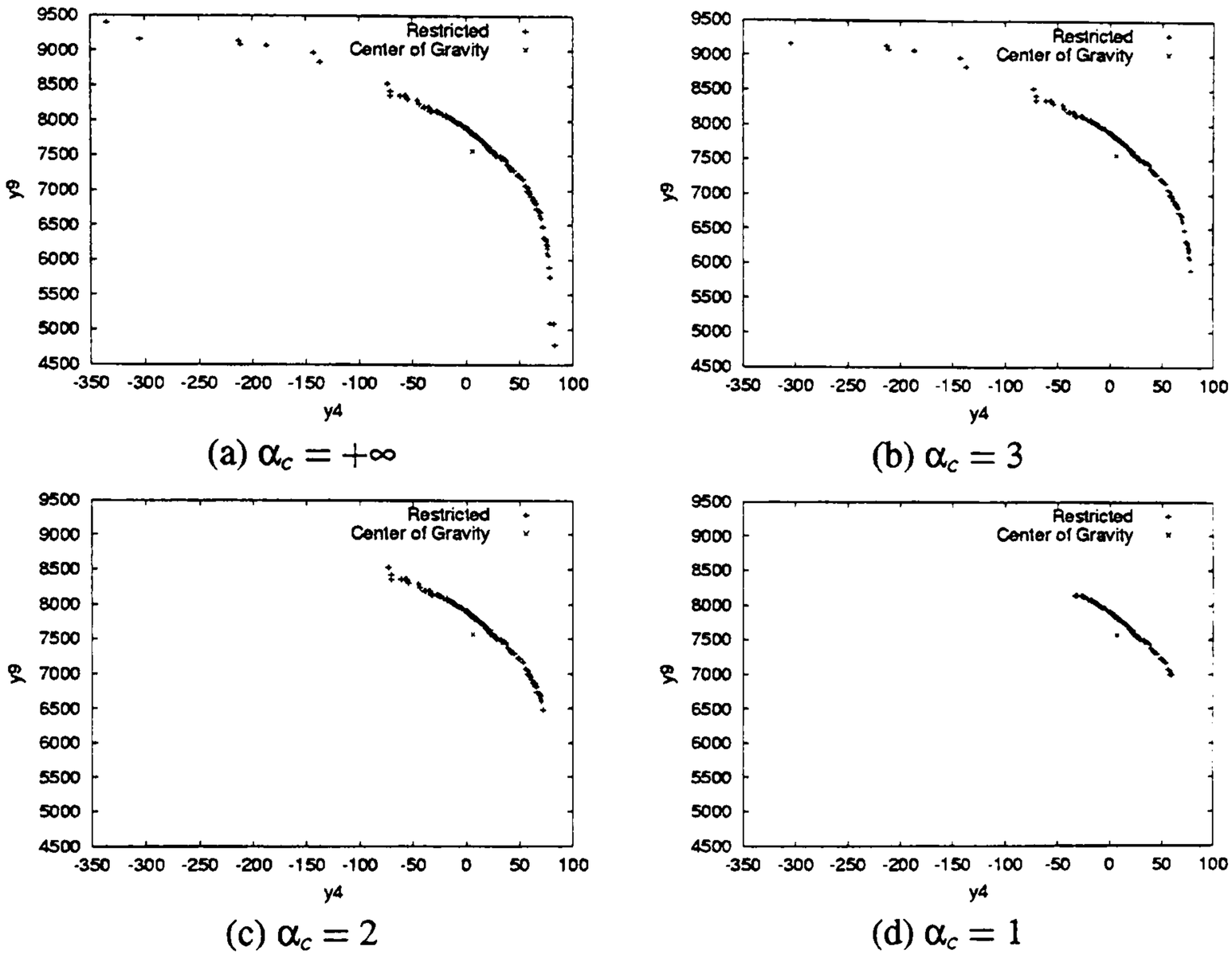


Figure 6.7. Pareto front size as a function of restricting parameter α_c.



(a) $\alpha_c = +\infty$ (b) $\alpha_c = 3$

(c) $\alpha_c = 2$ (d) $\alpha_c = 1$

Figure 6.8. Restricting Pareto front for $y_4 \ll y_9$ preferences. Influence of α_c factor.

the following:

> *A proposed co-evolutionary distributed method utilises individual GAs for the optimisation of each objective. The problem is therefore reduced to a number of concurrent co-evolutionary tasks specific to the overall design domain. PVM software controls the distributed architecture ensuring minimal clock time for these multi–objective problems.*
>
> *The fitness for each objective is normalised relative to the maximum and minimum values found during each GA run with constant adjustment as new upper and lower limits are identified. For each generation, solutions relating to each objective are compared with the best individual from the other GA populations. If a variable is outside a range defined by a range constraint map it is adjusted by a penalty function. (Parmee & Watson 1999, p. 1658)*

The constraint map is defined as a monotonically non–increasing real function $d_p : \mathbf{N} \mapsto [0, 1]$. Parmee & Watson (1999) used the following functions:

1. $d_p(t) = 1$ (no distance constraints);
2. $d_p(t) = \max\{1 - t/n_m, \mu_p\}$ (full linear constraint function);

3. $d_p(t) = \max\{1 - 2t/n_m, \mu_p\}$ (half linear constraint function);

4. $d_p(t) = \mu_p + (1-\mu_p)\cdot(1+\cos(\pi t/n_m))/2$ (full cos–like constraint function);

5. $d_p(t) = \begin{cases} \mu_p + (1-\mu_p)\cdot(1+\cos(2\pi t/n_m))/2, & t \le n_m/2; \\ \mu_p, & t \ge n_m/2. \end{cases}$ (half cos–like constraint function).

where μ_p is the minimal relevant distance value (0.1 in above paper), t is the current generation number, and n_m is the maximal number of generation of the GA (100 in above paper). If the difference between variable x in two individuals from different GAs (optimising different objectives) is greater then $d_p(t)$ of their variable range, their fitness functions are multiplied by the penalty factor $\phi_p < 1$. Formally, fitness function in generation t is defined as

$$f_i(\boldsymbol{x},t) = f_i^0(\boldsymbol{x}) \cdot \prod_{j=1}^{k} \chi_d(x_j, x_i, t) \qquad \text{where } \chi_d(x,y,t) = \begin{cases} \phi_p, & \text{if } |x-y| \ge d_p(t); \\ 1, & \text{otherwise.} \end{cases} \tag{6.4}$$

or, in C:

```
/* compute fitness penalty based on distance and constraint map */
f1 = cmap[gen];      /* maximal allowed distance without penalising */
for(i=0;i<popsize;i++)      /* now apply constraint map */
   for (j = 0; j < nx; j++) {
      f2 = f1*(XRANGE[j][1] - XRANGE[j][0]);
      for (k = 0; k < slaves; k++) {
         if (s ≠ k)
            if (fabs(pop[s][i].decval[j]-pop[k][0].decval[j])>f2)
               pop[s][i].fitness *= fitpen;      /* fitpen = 0.5 */
      }
   }
```

In the case of 3 objectives, ATR1, SEP1 and FR, and three populations, the graph of FR objective is presented in Figure 6.9. It can be seen that at the beginning, where there is no constraint map, each population gives different values for FR, but as the constraint map gets stricter they all converge towards (phenotypically) similar solution.

However, the original algorithm considers all objectives equally important and solutions converge towards the middle solution. The question is: is it possible to use the preference method to make some objective more important (e.g. aeroplane with good ferry range is more important than the aeroplane with good ATR)? This question is discussed in the rest of this section.

The problem with the fitness function (6.4) approach is that the penalty function is *phenotype* based, i.e. the distance is measured in *variable* space, not in *objective* space where the concept of preferences is applied.

One method of overcoming this problem is by modifying the penalty factor ϕ_p in equation (6.4) in the following way: If the objective $f_i(\boldsymbol{x})$ has preference based weight w_i and $f_j(\boldsymbol{x})$ preference based weight w_j, then:

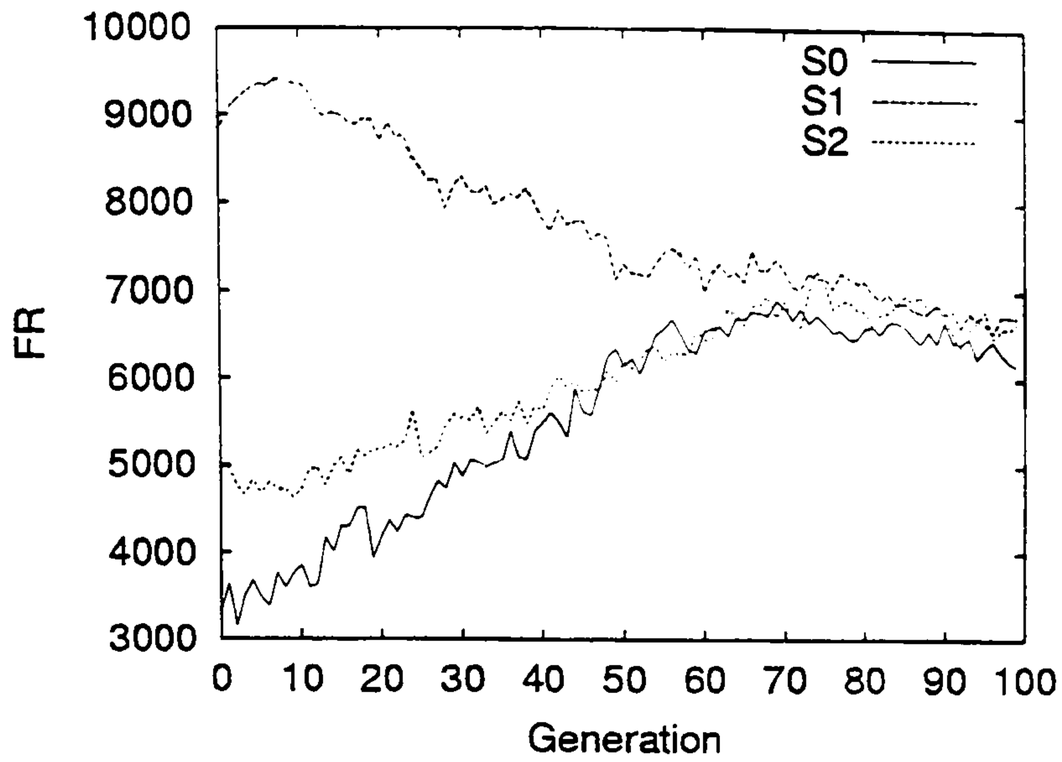


Figure 6.9. Converging towards common solution.

- If the distance between variables is $\geq d_p(t)$, multiply the fitness f_i by $\min\{1, (w_i/w_j) \cdot \phi_p\}$ and multiply the fitness f_j by $\min\{1, (w_j/w_i) \cdot \phi_p\}$.
- If the distance between variables is $< d_p(t)$, do not change fitnesses f_i and f_j.

i.e. mathematically, the modified equation (6.4) is needed:

$$f_i(\boldsymbol{x}, t) = f_i^0(\boldsymbol{x}) \cdot \prod_{j=1}^{k} \chi_d(x_j, x_i, t) \quad \text{where } \chi_d(x, y, t) = \begin{cases} \phi_p \cdot \theta_p(x, y), & \text{if } |x - y| \geq d_p(t); \\ 1, & \text{otherwise.} \end{cases} \tag{6.5}$$

for

$$\theta_p(x, y) = \min\{1, w_x/w_y\} \tag{6.6}$$

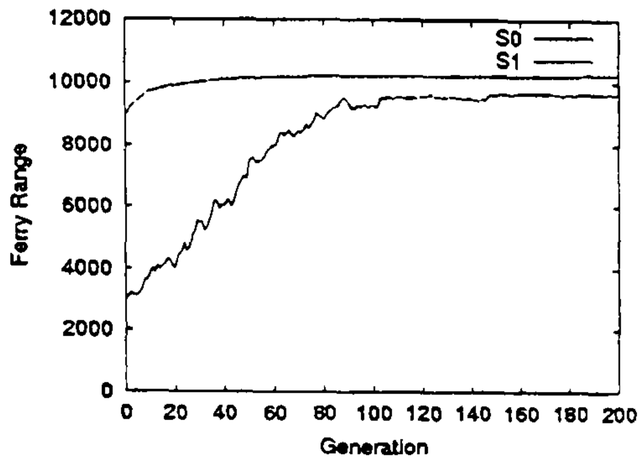
and w_t is the weight of objective t.

In that way, if objective f_i is more important than objective f_j, its fitness gets less penalised and the convergence process is biased towards objective f_i.

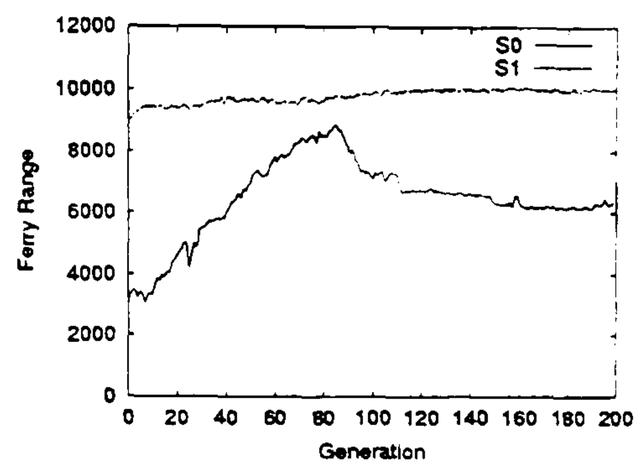
The corresponding C code is the following (comparing with previous version that implements equation (6.4), there is only one line of code added):

```
/* compute biased fitness penalty based on distance and constraint map */
f1 = cmap[gnum];        /* maximal allowed distance without penalising */
for(i=0;i<popsize;i++)      /* now apply constraint map */
   for (j = 0; j < nx; j++) {
      f2 = (XRANGE[j][1] - XRANGE[j][0]) * f1;
      for (k = 0; k < slaves; k++) {
         fp = min(1, fitpen*weight[s]/weight[k]);      /* fitpen = 0.5 */
         if (s ≠ k)
            if (fabs(pop[slave][i].decval[j]-pop[k][0].decval[j])>f2)
               pop[s][i].fitness *= fp;
      }
   }
```

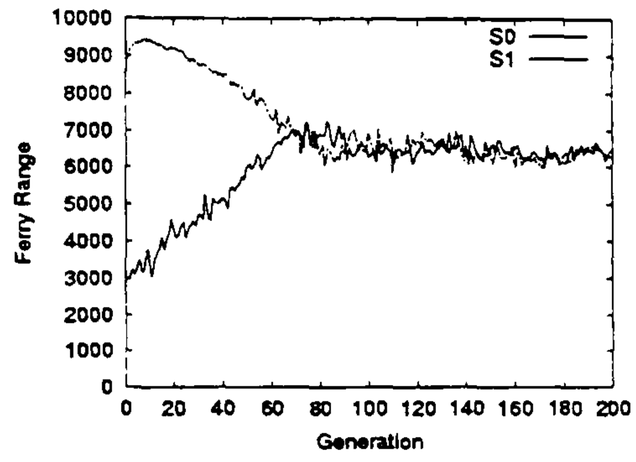
Tests using 2 GA processes S_0 and S_1, S_0 optimising SEP1, S_1 optimising FR, have been run and the results for FR for different preferences are shown in Figure 6.10. Figure 6.11 shows the results for SEP1. All test are averaged over 20 runs. It can be seen that using different preferences, the results are more biased towards regions where the preferred objective has higher values. There is also an interesting behaviour noticeable in Figure 6.10(b) and in Figure 6.11(b) where the processes are diverging. One possible explanation is that the use of preferences decrease the severity of penalty factors, so preferred objective is not "encouraged" towards compromise region as heavily as in the non–preference case.



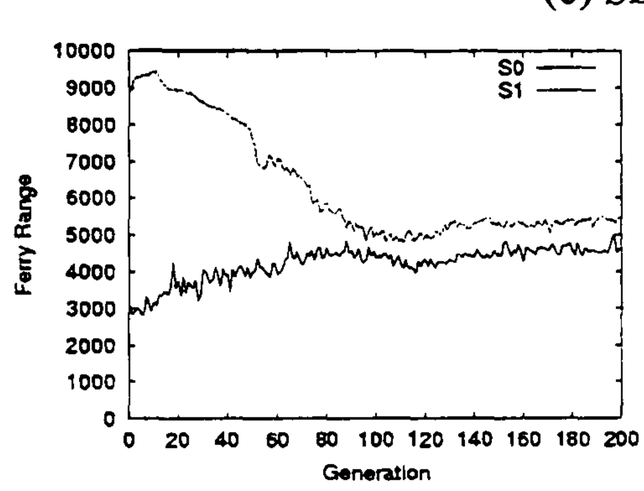
(a) $SEP1 \ll FR$



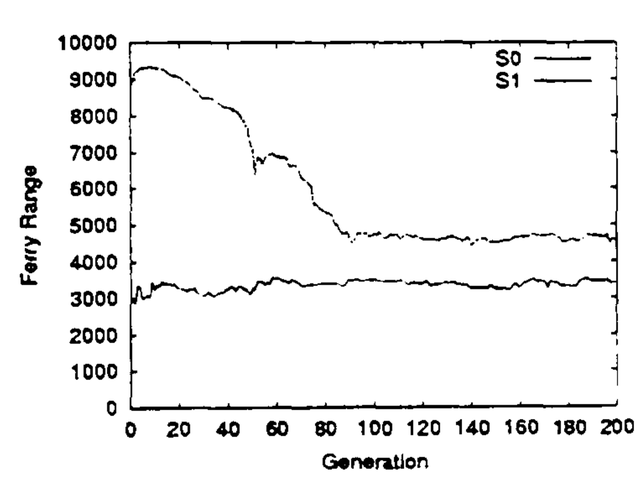
(b) $SEP1 \prec FR$



(c) $SEP1 \approx FR$



(d) $SEP1 \succ FR$

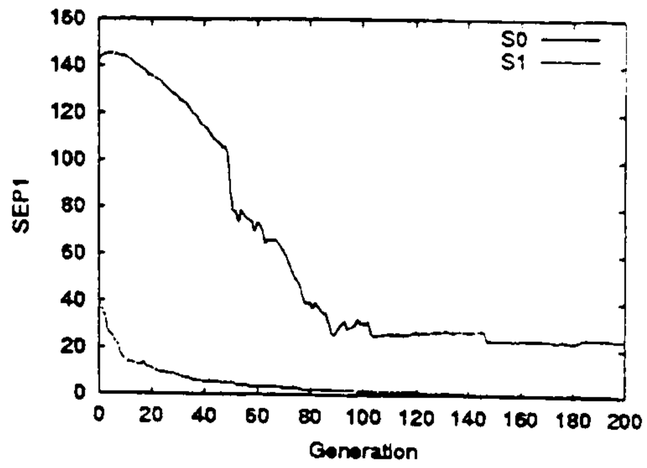


(e) $SEP1 \gg FR$

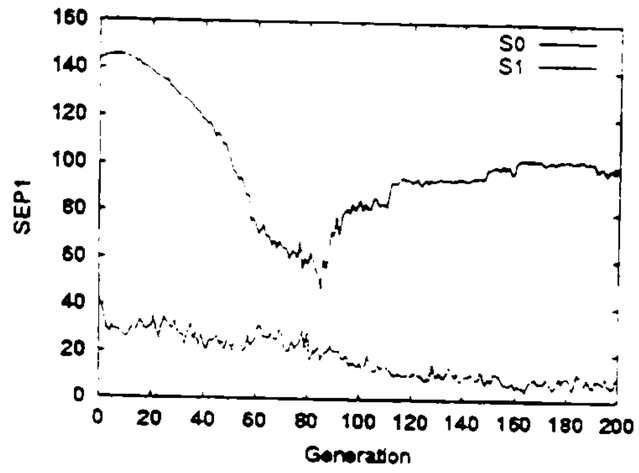
Figure 6.10 Co-evolution and different preferences: results for FR. Process S_0 optimises SEP1 and process S_1 optimises FR.

6.8 Problems, discussion and further research path

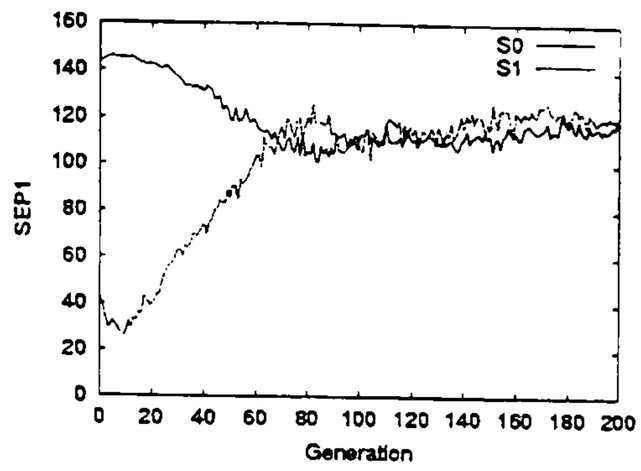
The previous two chapters described a novel method for transforming qualitative characterisation of objective relative importance into quantitative characterisation. An algorithm was given that implements the transformation. Integration with traditional and GA based multi–objective optimisation methods was discussed and a novel Pareto optimisation method integration with weights/preferences was developed. Some applications of preferences in the new Pareto based method were presented. Also, a method for integrating preferences into co-evolutionary optimisation framework was given. In the future work further development of the preference



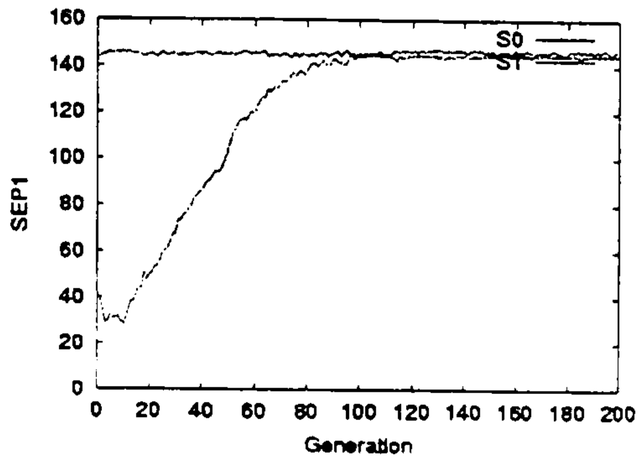
(a) $SEP1 \ll FR$



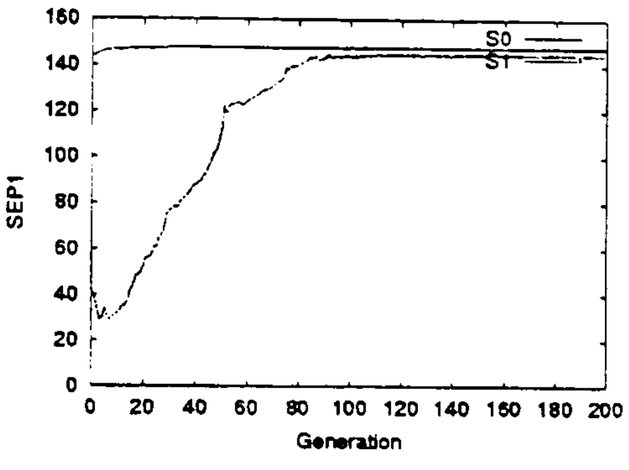
(b) $SEP1 \prec FR$



(c) $SEP1 \approx FR$



(d) $SEP1 \succ FR$



(e) $SEP1 \gg FR$

Figure 6.11 Co-evolution and different preferences: results for SEP1. Process S_0 optimises SEP1 and process S_1 optimises FR.

model is planned and its more tight integration into the real world applications.

To conclude this stage, here is a word of warning from (Keeney et al. 1976, p. ix):

> *"...Now assume you are a harassed decision maker sitting in front of an output device deluged with a mountain of conflicting information. You are confused. What should you do? How can you sort out the issue and start thinking systematically about your choice problem: which policy should you adopt in the real setting?*
>
> ...
>
> *We are convinced of one thing: The decision maker cannot simply plug these incommensurate output performance measures into an objective formula that someone has proposed* ex ante *without any reference to the real–world meaning of the various measures. Instead, our prescriptions lead us in an opposite direction: we advocate that the responsible decision maker force himself to think hard about various value tradeoffs and about his attitudes towards risky choices and we suggest ways that this process can be systematically examined by dividing his complicated choice problem into a host of simpler choice problems."*

Another challenge is the actual realistic number of objectives and preferences one can deal with. There is a difference between the following two design stages:

- detailed engineering design tends to perform the optimisation as far as possible, but rarely deals with more than a couple of objectives, and
- conceptual engineering design where the global idea of a product is created by simultaneously tackling as many issues as possible (cf. section 1.1 and (Pahl & Beitz 1996)).

There is also a point of view expressed by Sen & Yang (1998) that adding user preferences makes sense only up to the certain point. After that, they simply have no further influence. Our situation is similar: with an increased number of objectives, they become less and less sensitive to user specified preferences.

CHAPTER 7

Scenarios in Engineering Design

In section 1.1, page 1, different phases of engineering design has been introduced and it has been established that different phases of design require different levels of creativity: the degree of creativity is the highest in conceptual design and the lowest in the detail design. Further *"...in creative design, knowledge needed to address a problem typically is not available in a form directly applicable to the problem. Instead, at least some of the needed knowledge has to be acquired from other knowledge sources, by analogical transfer from a different problem for example"* (Goel 1997).

Therefore, we can say that the conceptual design phase has the following components:

- Mathematical model;
- Cognitive model.

They are presented in Figure 7.1.

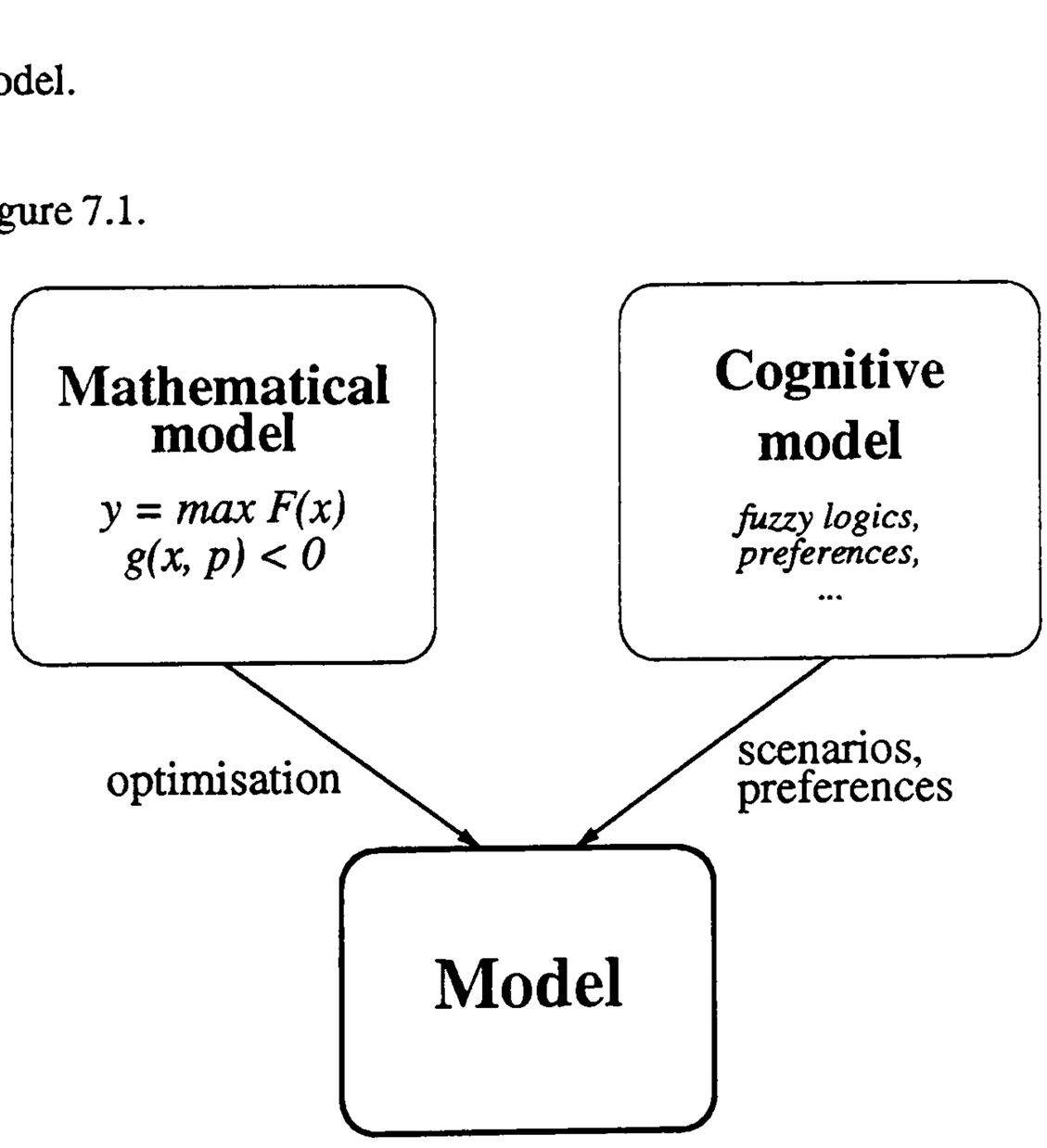


Figure 7.1. The initial phase of computer aided whole system design

For the mathematical model solving, an genetic algorithm has been used, described in chapter 3. The cognitive model component is more subjective and cannot always be expressed in strict mathematical model

terms. Further, different designers working on the same conceptual design problem, although sharing the same mathematical model, may have different ideas and opinions i.e. different cognitive model.

The previous chapters have introduced the preference method that is used to connect the two components of preliminary design. In this chapter another tool, scenarios will be introduced and used.

7.1 Scenarios

Scenario handling is another tool that we want to integrate into our conceptual design system.

In a typical design situation, the designer has one or several preferential situation (scenarios) that he adds to the fitness function such as: "I would like to have $y_5 \in [0,4]$ or, if not possible (or feasible), then I would like to have $y_1 + y_3 > 100$ etc." Therefore, additionally to the mathematical model with its set of objectives and a fixed set of constraints, there is a dynamical, interactive set of additional constraints or goals.

Definition 7.1 Scenario *is formally represented as a conjunction of relations (constraints) in a fairly rich mathematical language, similar to the concept of Horn clause in mathematical logic, Prolog or resolution based automated theorem proving methods (Chang & Lee 1971, Loveland 1978). Each scenario is a function of model's inputs, objectives and (eventually) some additional parameters. Its value is between 0 and 1, representing the percentage of the relations satisfied for given values of (its) inputs.*

Formally, using Backus–Naur Forms (BNF), a scenario grammar is presented in Figure 7.2. Computer implementation is based on the following YACC (Johnson 1975, Aho, Sethi & Ullman 1987) grammar:

```
/* scenarios grammar for the first case */

%token NUM VAR OBJ /* number, variable or objective */
%token NEQ EQ GEQ LEQ LT GT /* relations */
%token ABS SQRT MIN MAX EXP LOG SIN COS ADD SUB MUL DIV /* operators */

%nonassoc NEQ EQ GEQ LEQ LT GT /* nonassociative relations */

%left ADD SUB /* left associative operators */
%left DIV MUL /* left associative operators, higher priority */                    10
%right POW /* exponential sign, right associative */
%left NEG /* sign change */

%%
line    :       /* nothing */
        |       line '\n'
        |       line list '\n'
        |       line list ';'
        ;
list    :       element                                                          20
        |       list '&' element
        ;
element :       expr LT expr
        |       expr EQ expr
        |       expr NEQ expr
        |       expr GEQ expr
        |       expr LEQ expr
        |       expr GT expr
        ;
expr    :       expr ADD expr                                                    30
        |       expr SUB expr
        |       expr MUL expr
```

```
        expr DIV expr
        expr POW expr
        ABS '(' expr ')'
        SQRT '(' expr ')'
        EXP '(' expr ')'
        LOG '(' expr ')'
        SIN '(' expr ')'
        COS '(' expr ')'
        MIN '(' expr ',' expr ')'
        MAX '(' expr ',' expr ')'
        '(' expr ')'
        SUB expr %prec NEG
        VAR
        OBJ
        NUM
;
```

40

$\langle$scenario-l$\rangle$ $\longrightarrow$ $\langle$scenario$\rangle$ [; $\langle$scenario$\rangle$] *

$\langle$scenario$\rangle$ $\longrightarrow$ $\langle$element$\rangle$ [& $\langle$element$\rangle$] *

$\langle$element$\rangle$ $\longrightarrow$ $\langle$expr$\rangle$ $\langle$rel$\rangle$ $\langle$expr$\rangle$

$\langle$rel$\rangle$ $\longrightarrow$ < | = | > | >= | <= | <>

$\langle$expr$\rangle$ $\longrightarrow$ $\langle$expr$\rangle$ $\langle$add-op$\rangle$ $\langle$term$\rangle$ | $\langle$term$\rangle$

$\langle$term$\rangle$ $\longrightarrow$ $\langle$term$\rangle$ $\langle$mult-op$\rangle$ $\langle$efactor$\rangle$ | $\langle$efactor$\rangle$

$\langle$efactor$\rangle$ $\longrightarrow$ $\langle$factor$\rangle$ $\langle$pow-op$\rangle$ $\langle$efactor$\rangle$ | $\langle$factor$\rangle$

$\langle$factor$\rangle$ $\longrightarrow$ ($\langle$expr$\rangle$) | - $\langle$expr$\rangle$ | $\langle$op$\rangle_P$ ($\langle$expr$\rangle$) | $\langle$var$\rangle$ | $\langle$const$\rangle$

$\langle$add-op$\rangle$ $\longrightarrow$ + | -

$\langle$mult-op$\rangle$ $\longrightarrow$ * | /

$\langle$pow-op$\rangle$ $\longrightarrow$ ^

$\langle$op$\rangle_P$ $\longrightarrow$ sqrt | abs | max | min | sin | cos | log | exp

$\langle$var$\rangle$ $\longrightarrow$ X$\langle$const$\rangle$ | Y$\langle$const$\rangle$

$\langle$const$\rangle$ $\longrightarrow$ $\langle$number$\rangle$

Figure 7.2. Formal Backus–Naur Form (BNF) grammar of scenarios.

Example 7.1 Consider a following set of 3 scenarios:

$S_1 : (y_3 > 100) \wedge (y_2 > 7000)$

$S_2 : (y_4 \cdot e^{-x_1} \leq 2x_2 + \sqrt{7.3y_5}) \wedge (y_1 < 12.42x_4) \wedge (x_1 + x_2 > 20)$

$S_3 : \sin(x_2/100) < 0.54$

Those scenarios can be specified in the following (C–like) notation:

```
y3 > 100 & y2 > 7000
y4*exp(-x1) <= 2*x2 + sqrt(7.3*y5) & y1 < 12.42*x4 & x1 + x2 > 20
sin(x2/100) < 0.54
```

If $x_1 = 1.0, x_2 = 17.9, x_4 = 5.12, y_1 = 59.4, y_2 = 7122, y_3 = 95, y_4 = 360$ and $y_5 = 1400$, then

$$v(S_1) = \frac{1}{2}, \quad v(S_2) = \frac{2}{3}, \quad v(S_3) = 1.$$

7.1.1 Combinations of scenarios

The evaluation of scenarios S_i (for $1 \leq i \leq M_s$) gives the set of (0,1)–real values $\{v(S_i) \mid 1 \leq i \leq M_s\}$ but the question is how to interpret the set of scenarios $S = \{S_i \mid 1 \leq i \leq M_s\}$ and what is its value $v(S)$? This is a problem in itself and there are several possible approaches:

1. Let the designer specify the relative importance of each scenario, using weights w_i or using our preference based method for specifying importance. Then

$$v(S) = \sum_i w_i \cdot v(S_i).$$

2. The following geometric progression can be applied: importance factor of scenario S_1 is α, scenario S_2 is a factor α less important than S_1 (i.e. its importance factor is α^2), scenario S_3 is a factor α less important than S_2 etc., where α is a solution of the equation

$$\sum_{i=1}^{M_s} x^i = 1, \quad 0 < x \leq 1.$$

   In that case $v(S)$ is computed as

$$v(S) = \sum_{i=1}^{M_s} \alpha^i \cdot v(S_i) \tag{7.1}$$

   Continuing Example 7.1, using this method: $M_s = 3$, $\alpha = 0.5437$ and $v(S) = 0.6296$, i.e. the set of scenarios is accomplished by approximately 63%.

   The advantage of this method is that the designer does not have to worry about specifying the importance of scenarios, since they have been calculated automatically. However, this model is rather limited, and depends heavily on the scenario order, so that the designer has to know what is the most preferable scenario, what is the second one etc. (as in the lexicographic ordering described in section 3.2.2, page 23).

3. Third method considers each scenario independently of the others and performs a parallel, distributed, island–based optimisation where each island optimises the original problem plus one scenario. In this case, agents or some other methods can be used that can monitor the fulfilment of the scenarios. The advantage of this method is that there is no need to fulfil all scenarios (they could be conflicting) and that the system (through agents) can signal back the possibility or impossibility of the fulfilment of scenarios.

   Obviously, this method has the broadest potential and our future work will mainly be in that area. Some work is presented in chapter 8 describing agents in the conceptual design context.

4. Fourth method is the classical one: use penalty functions (Michalewicz 1995) and penalise the solutions that violate some of those scenarios. However, "...there is no general guideline on designing penalty functions, and constructing an efficient penalty function is quite problem–dependent" (Gen & Cheng 1997, p. 52).

7.2 Different scenarios scenario

The previous section describes a scenario grammar as defined in Definition 7.1 and shown in Figure 7.2. It gives a level of fulfilment of scenarios (as a real number between 0 and 1), but which has very limited expression power (i.e. no disjunction nor negation within the scenario). In this section a new, more powerful scenario grammar is created, that includes these two additional logical operators (using symbol \/ for disjunction and ~ for negation). The values of scenarios are limited to true (1) or false (0) i.e. a scenario is true or a scenario is false. A combination of those two approaches is possible, but this would require some form of multi–valued logic (Urquhart 1986). However, that is a matter of further research. One possible approach is the following interpretation:

$$
\begin{aligned}
v(\bigwedge_{i=1}^{n} \phi_i) &= \frac{k}{n}, \qquad k \text{ is the number of valid formulas} \\
v(\bigvee_{i=1}^{n} \phi_i) &= \begin{cases} 1, & \text{at least one of } \phi_i \text{ is true} \\ 0, & \text{otherwise.} \end{cases} \\
v(\neg\phi) &= 1 - v(\phi)
\end{aligned}
$$

This interpretation uses the same conjunction operator as the first grammar, but unfortunately not all of the usual properties of the logical operators are fulfilled (e.g. de Morgan's laws).

The complete specification of the modified grammar is given in Figure 7.3.

$$
\begin{aligned}
\langle \text{scenario-l} \rangle &\longrightarrow \langle \text{scenario} \rangle \; [; \langle \text{scenario} \rangle]^* \\
\langle \text{scenario} \rangle &\longrightarrow \langle \text{element} \rangle \; [\langle \text{log-op} \rangle \langle \text{scenario} \rangle] \mid (\; \langle \text{scenario} \rangle \;) \\
\langle \text{element} \rangle &\longrightarrow \langle \text{expr} \rangle \langle \text{rel} \rangle \langle \text{expr} \rangle \mid \tilde{}\, \langle \text{scenario} \rangle \mid (\; \langle \text{scenario} \rangle \;) \\
\langle \text{log-op} \rangle &\longrightarrow \&\mid \backslash/ \\
\langle \text{rel} \rangle &\longrightarrow <\mid =\mid >\mid >=\mid <=\mid <> \\
\langle \text{expr} \rangle &\longrightarrow \langle \text{expr} \rangle \langle \text{add-op} \rangle \langle \text{term} \rangle \mid \langle \text{term} \rangle \\
\langle \text{term} \rangle &\longrightarrow \langle \text{term} \rangle \langle \text{mult-op} \rangle \langle \text{efactor} \rangle \mid \langle \text{efactor} \rangle \\
\langle \text{efactor} \rangle &\longrightarrow \langle \text{factor} \rangle \langle \text{pow-op} \rangle \langle \text{efactor} \rangle \mid \langle \text{factor} \rangle \\
\langle \text{factor} \rangle &\longrightarrow (\; \langle \text{expr} \rangle \;) \mid -\langle \text{expr} \rangle \mid \langle \text{op} \rangle_P \; (\; \langle \text{expr} \rangle \;) \mid \langle \text{var} \rangle \mid \langle \text{const} \rangle \\
\langle \text{add-op} \rangle &\longrightarrow +\mid - \\
\langle \text{mult-op} \rangle &\longrightarrow *\mid / \\
\langle \text{pow-op} \rangle &\longrightarrow \hat{} \\
\langle \text{op} \rangle_P &\longrightarrow \texttt{sqrt} \mid \texttt{abs} \mid \texttt{max} \mid \texttt{min} \mid \texttt{sin} \mid \texttt{cos} \mid \texttt{log} \mid \texttt{exp} \\
\langle \text{var} \rangle &\longrightarrow \texttt{X} \langle \text{const} \rangle \mid \texttt{Y} \langle \text{const} \rangle \\
\langle \text{const} \rangle &\longrightarrow \langle \text{number} \rangle
\end{aligned}
$$

Figure 7.3. Formal Backus–Naur Form (BNF) grammar for scenarios.

The corresponding YACC grammar contains the following specification (this is not a complete YACC

grammar, some details are omitted for brevity as they do not contribute to its understanding):

```
/* scenarios grammar for the second case */
%token NUM VAR OBJ /* number, variable or objective */
%token NEQ EQ GEQ LEQ LT GT NOT /* relations */
%token AND OR ABS SQRT MIN MAX EXP LOG SIN COS ADD SUB MUL DIV /* opersators */
%nonassoc NEQ EQ GEQ LEQ LT GT
%left AND OR /* left associative relations */
%left ADD SUB /* left associative operators */
%left DIV MUL /* left associative operators, higher priority */
%right POW /* exponential sign, right associative */
%left NOT NEG /* sign change and negation */
%%
line    :       /* nothing */
        |       line '\n'
        |       line list '\n'
        |       line list ';'
        ;
list    :       element
        |       list AND element
        |       list OR element
        |       '(' list ')'
        ;
element :       expr LT expr
        |       expr EQ expr
        |       expr NEQ expr
        |       expr GEQ expr
        |       expr LEQ expr
        |       expr GT expr
        |       expr
        |       '(' list ')'
        |       NOT list
        ;
expr    :       expr ADD expr
        |       expr SUB expr
        |       expr MUL expr
        |       expr DIV expr
        |       expr POW expr
        |       ABS '(' expr ')'
        |       SQRT '(' expr ')'
        |       EXP '(' expr ')'
        |       LOG '(' expr ')'
        |       SIN '(' expr ')'
        |       COS '(' expr ')'
        |       MIN '(' expr ',' expr ')'
        |       MAX '(' expr ',' expr ')'
        |       '(' expr ')'
        |       SUB expr %prec NEG
        |       VAR
        |       OBJ
        |       NUM
        ;
```

Example 7.2 Returning to the example 7.1 (page 86), using new interpretation, scenarios are specified in the same way as before, but their values are different:

$$v(S_1) = 0, \quad v(S_2) = 0, \quad v(S_3) = 1$$

instead of the previous:

$$v(S_1) = \frac{1}{2}, \quad v(S_2) = \frac{2}{3}, \quad v(S_3) = 1.$$

This is not as fine as before, but it allows to also express scenarios such as

1. $S_4 : ((x_1 > 0) \wedge (x_1 < 5)) \vee ((x_1 > 10) \wedge (x_1 < 15))$

2. $S_5 : \neg((y_2 < 0) \vee (y_4 + y_5 > 66))$

in the following way:

```
(x1 > 0 & x1 < 5) \/ (x1 > 10 & x1 < 15)
~ (y2 < 0 \/ y4 + y5 > 66)
```

Using the same variable valuations as in Example 7.1, their values are:

$$v(S_4) = 1,\ v(S_5) = 0.$$

In the further text the second version of scenarios will be used.

7.3 Applications of scenarios

The main application of scenarios is to give additional constraints and objectives that are not built–in into the basic (miniCAPS) model. That gives the designer a lot of flexibility as it enables him to dynamically change those additional requirements. Each of scenarios can have a preference–based weight assigned. Fitness function is calculated as:

$$f(\boldsymbol{x}) = f_o(\boldsymbol{x}) \times \sum_i w_i \cdot v(S_i) \tag{7.2}$$

where w_i are preference weights summing to 1, and $f_o(\boldsymbol{x})$ is the original fitness function.

Example 7.3

Data for all aircrafts are from (Sharpe 1999, Jane's Information Group 1999) as well as from (Chandler 1999).

(a) Suppose that the following set of scenarios is given :

```
# constraints for F-117A Stealth Fighter
y11 <=13.20    # Wing span
x4 <=105.9     # Wing plan area
y9 >=2327      # Ferry range estimated as combat radius * 2.7
y10 >= 24894   # take-off mass
y1 <= 1890     # take-off run
```

Running the program with this set of scenarios and maximising y_9 and y_{10} and minimising y_1 and y_{11} gives the following solution:

```
input parameters: 0.9 7383.3 0.8969 80 2.178 0.6 60 0.12 0.8738
relevant outputs: y1=602.45, y9=7691.98, y10=29288.96, y11=13.2
```

whereas running without any scenarios, the following solution is obtained:

```
input parameters: 0.8978 8089.48 0.9 80 6 0.1 60 0.12 0.7416
relevant outputs: y1=969.33, y9=7448.92, y10=37151.64, y11=21.9
```

i.e. since wing span is not limited any more, much better take-off mass is obtained, but also causing longer take-off run.

(b) With the following set of scenarios:

```
# constraints for Grumman F-14D Tomcat
y11 >= 11.65  & y11<=19.55  # Wing span
x4 <=52.49                  # Wing plan area
y9 >=3200                   # Ferry range
y10 >= 33724                # take-off mass
x3 >= 0.83                  # max cruise speed 1019km
y1 <= 427                   # take-off run
```

the solution that is obtained violates scenario S_4 (take-off mass constraint):

```
input parameters:0.8492 6856.77 0.79 52.1 5.5 0.1 60 0.03 0.9154
relevant outputs: y1=425.45, y9=5359.99, y10=25762.06, y11=16.93
```

Here the take–off mass parameter is violated, but all other are fulfilled.

Using preferences, scenario S_4 can be fulfilled:

By setting the following set of preferences for above set of scenarios $\{S_1, \ldots, S_6\}$: $S_4 \succ S_1 \approx S_2 \approx S_3 \approx S_5 \approx S_6$, the solution is obtained where S_4 is fulfilled, but scenarios S_1, S_2 and S_6 are not:

```
input parameters:0.8079 8089.5 0.8597 8 6 0.1 60 0.12 0.3836
relevant outputs: y1=969.33, y9=7448.92, y10=37151.64, y11=21.9
```

This certainly gives the designer possibilities to balance different constraints and objectives: if the take–off mass is more important than take–off run (as in the case of bombers and transporters), the search can be biased in that direction. If however, take–off run and wing span are more important (for fighters), it will set the preferences accordingly, taking into account lesser take-off mass etc.

(c) A similar example is shown below, with the following set of scenarios:

```
# constraints for F-111 Aardvark Bomber
y11 >= 9.74  & y11<=19.20  # Wing span
x4 <=61.07 & x4 >=48.77    # Wing plan area
y9 >=4707                  # Ferry range
y10 >= 45360               # take-off mass
x3 >= 0.75                 # max cruise speed 919km/h
y1 <= 951                  # take-off run
```

If the program is in its original form, regardless of preferences, the take–off mass scenario cannot be fulfilled. The limitation is in the input range of variable $x_4 \in [20, 80]$ (from the input file `caps.dat`). However, increasing wing plan area (x_4) range to 110, using preference setting $S_4 \succ S_1 \approx S_2 \approx S_3 \approx S_5 \approx S_6$, a solution is obtained where S_4 is fulfilled, but scenarios S_1 and S_2 are not:

```
inputs:0.8953 9077.8174 0.8924 110 6 0.1 60 0.1064 0.4961
relevant outputs: y1=950.9987, y9=7817.30, y10=46760.24, y11=25.7
```

This example will be expanded later on in section 8.7.2 on page 110.

This example demonstrates the power of the developed system for interactive analysis and change of parameters in the run and provides almost immediate feedback about the objectives, constraints and preferences. Some further scenario examples for different aeroplanes are given in appendix C, page 138.

7.4 Scenarios in BAe system for conceptual design

Our system developed so far that applies scenarios together with designer interaction and genetic algorithms is simply represented as in Figure 7.4. It incorporates multi–objective optimisation methods, preferences and scenarios.

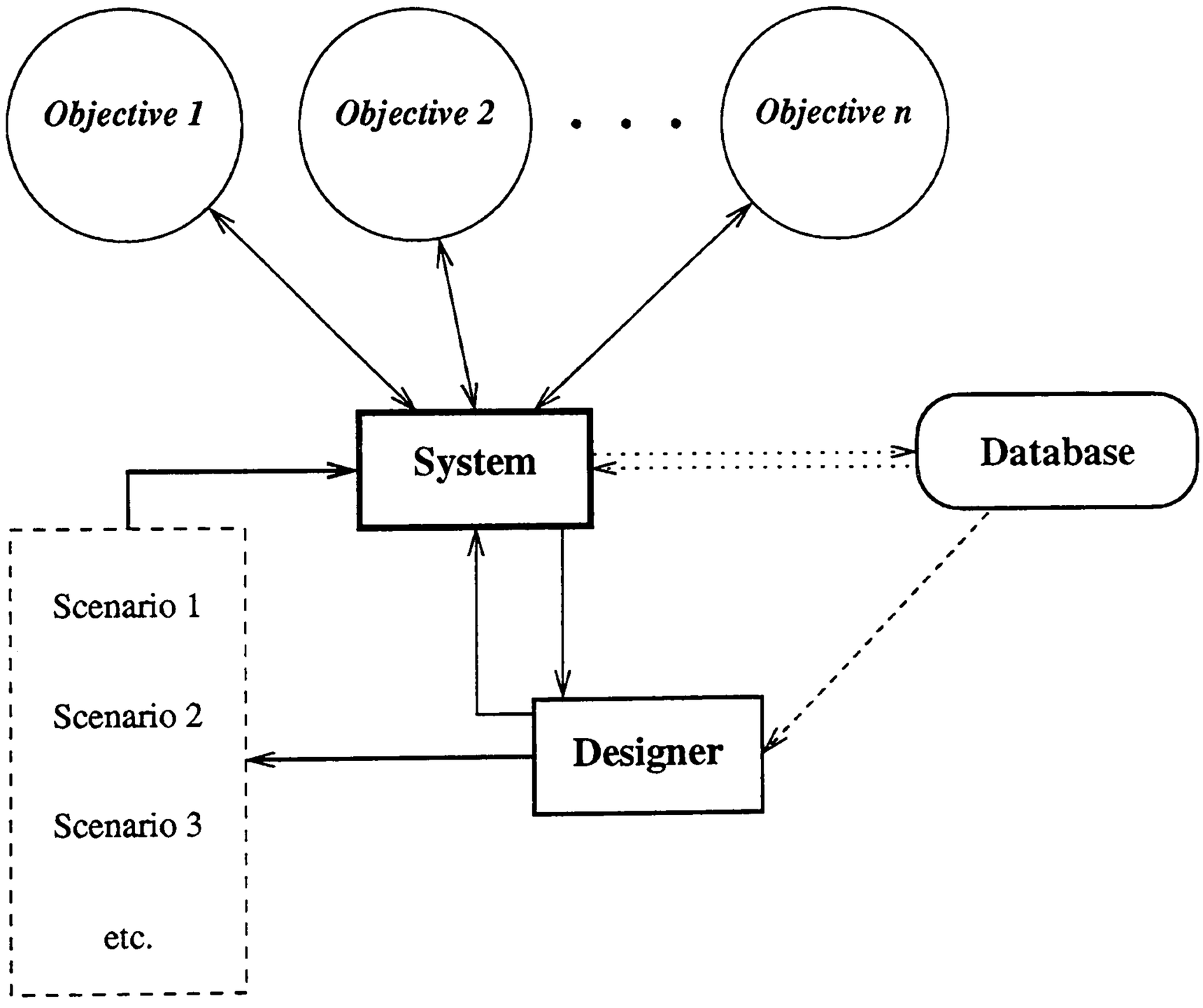


Figure 7.4. Schema of the computer/human design system.

The fitness function for our engineering design system is schematically presented in Figure 7.5. For the simplicity reasons, the intermediate fitness function is illustrated for weighted sum based transformation, but any other multi–objective optimisation method can be equally well used. It can be seen that the model of our system (reflected, as required for GAs by fitness function), goes through several transformations: first the set of objectives is transformed into a (usually but not necessary one–dimensional) fitness function, which is then, by means of scenarios, further transformed into another fitness function which better reflects and refines the

designer's needs and ideas. This transformation is at any time interactively changeable by the user (designer).

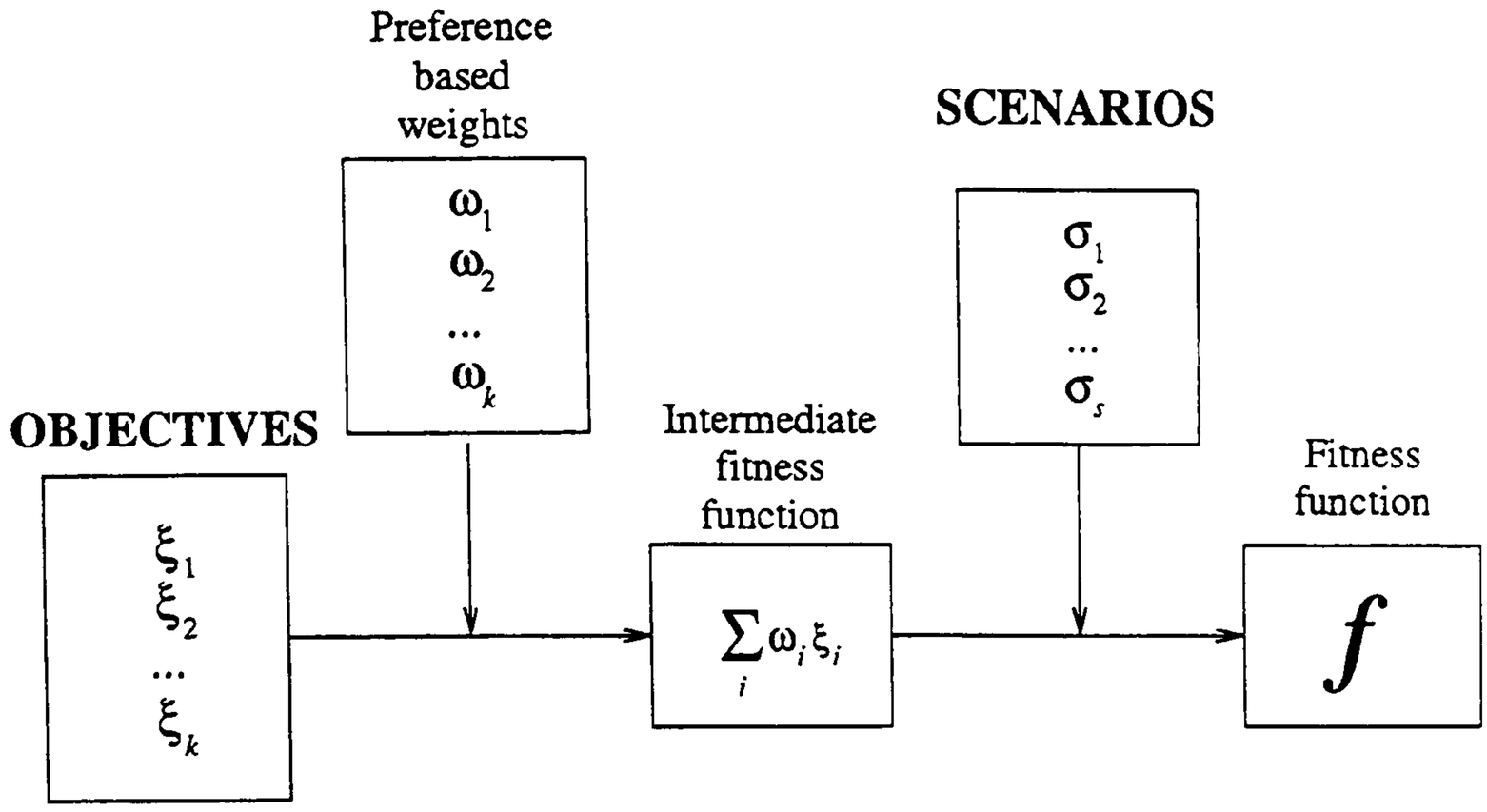


Figure 7.5. Transformation of fitness function

CHAPTER 8

Agents and their Use in Conceptual Design

The chapter discuss the use of agents generally, and in the conceptual design context in particular.

8.1 Introduction and a general framework

The first question one asks is "What is an agent". There is a nice quotation about it in (Wooldridge & Jennings 1995):

> *Carl Hewitt recently remarked (at the 13th international workshop on distributed AI) that the question* what is an agent? *is embarrassing for the agent–based computing community in the same way that the question* what is intelligence? *is embarrassing for the mainstream AI community. The problem is that although the term is widely used, by many people working in the closely related areas, it defies attempts to produce a single universally accepted definition. This need not be a problem: after all, if many people are successfully developing interesting and useful applications, then it hardly matters that they do not agree on potentially trivial terminological details. However, there is also the danger that unless the issue is discussed, 'agent' might become a 'noise' term, subject to both abuse and misuse, to the potential confusion of the research community.*

According to (Stenmark 1999):

> *An agent can be anything from a human being to a thermostat! Since this is a too broad definition to be useful, we shall try to narrow it down by restricting us to software agents and looking at some previous definitions. A personal agent is someone who acts on someone else's behalf. One way of defining a software agent is thus as a piece of software that assists people and acts on their behalf. A slightly more complicated definition: An autonomous agent is a system situated within and part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future. Though the definitions may vary between people there are a number of aspects that most people agree upon a program must posses to qualify as an agent. An agent should be:*
>
> - *Autonomous. The agent must have control over its own actions and be able to work and launch actions independent of the user or other actors.*
> - *Reactive. The agents can detect changes in its environment and react to those in a timely matter by answering to events and initiate actions.*
> - *Communicative. The agent is able to interact and communicate with users and other agents.*
> - *Goal-driven. Agents have a purpose and act in accordance with that purpose until it is fulfilled.*
>
> *Other aspects also often mentioned are dynamic (agents should be able to operate depending on time and space), adaptive (agents learn and change their behaviour based on previous experiences), temporal continuous (agents should not be started or stopped for explicit tasks but rather*

be a continuously running process), and/or mobile (agents should be able to move themselves from one machine to another, and across different architectures and platforms).

He further classifies agents into the following categories:

Interface agents Interface agents are used to decrease the complexity of the more and more sophisticated and overloaded information systems available. They may add speech and natural language understanding to otherwise dumb interfaces, or add presentation ability to systems.

System agents System agents run as integrated parts of operating systems or network protocol devices. They help managing complex distributed computing environments by doing hardware inventory, interpreting network events, managing backup and storage devices, and performing virus detection. These agents do not primarily work with end-user information.

Advisory agents Advisory agents are used in (complex) help or diagnostics systems.

Filtering agents Filtering agents are used to reduce information overload by removing unwanted data, i.e. data that does not match the user's profile, from the input stream.

Retrieval agents Retrieval agents search and retrieve information and serves as information brokers or documents managers.

Navigation agents Navigation agents are used to navigate through external and internal networks, remembering short-cuts, pre-load caching information, automatically bookmarking interesting sites.

Monitoring agents Monitoring agents provides the user with information when particular events occur, such as information being updated, moved, or erased.

Recommender agents Recommender agents are usually collaborative; they need many profiles to be available before an accurate recommendation can be made.

Profiling agents Profiling agents are used to build dynamic sites with information and recommendations tailored to match each visitor's individual taste and need. The main purpose is to build customer loyalty and profitable one-to-one relationships.

Watt (1996, p. 89) also discusses the definition issue:

"... 'Agent' is a difficult word for a difficult concept; covering a rag-bag of concepts that span a whole gamut of different kinds of behaviour, including, for example, autonomy, learning and social interaction; but there is a common ground. An agent will set out to do something, and do it; therefore it has competences for intending to act, for action in an environment, and for monitoring and achieving its goals. Of course, the adequate performance of these, other competences, such as learning, negotiation, and planning, may be helpful or even necessary.

This is not the whole story. Agency is a lot more than action in an environment, or, rather, the environment is not just a simple passive system. Often the environment will contain other agents, which is why social interaction and collaboration are so often stressed as a feature of agency. More interesting, perhaps, the environment may even contain people, leading to the human kind of agency — the kind we talk about in terms like 'estate agent'. Agents are embedded in an environment, but this environment is social as well as physical — social not only in that an agent is working with other agents, but also in terms that an agent must work with people as well. The environment, therefore, and the social rules that apply, are those of human social behaviour."

There is a large research area describing the logical background of agent theory (including topic such as belief, intension, default reasoning, possible world semantics etc.). An overview of theoretical aspects of agents is given in (Wooldridge & Jennings 1995).

8.1.1 Agent hierarchy

From a higher level view, there are the following types of agents:

Interface agents that help the designer deal with a system and which (if designer wishes it) can hide some low–level non-interesting details from the designer;

Search agents that cover the process of optimisation, cooperation, population monitoring, jumping out of regions, constraint questioning etc.

Information agents that deal with information obtained, that can look for interesting solution, filtering uninteresting ones, making decision what and where to explore, resolving conflicts etc.

There is a similar classification given in (Sycara, Decker et al. 1996) where the agent system RETSINA has 3 types of agents: *interface agents* (interacts with user receiving user specifications and delivering results), *task agents* (which carries out the plan and exchanges information with other agents) and *information agents* (which provide intelligent access to a heterogeneous collection of information).

8.1.2 Negotiations

According to Sycara (1991), there are four conflict situations where negotiation is used in design. These conflicts are (Berker 1995):

- Different agents make conflict recommendations for a parameter value;
- A value proposed by one agent makes it impossible for another agent to offer consistent values for other attributes;
- A decision of one agent adversely affects optimality of other agents;
- Alternate approaches achieve similar functional results.

The negotiation process proceeds as follows:

1. Generation of proposal;
2. Generation of counter proposal based on feedback from dissenting agents;
3. Communication of justifications and supporting evidence.

A paper by Nwana, Lee & Jennings (1996) also gives an overview of different coordination techniques. One negotiation process, tailored for conceptual design multi–objective process will be presented in section 8.5.1. page 106.

8.1.3 Agent communication

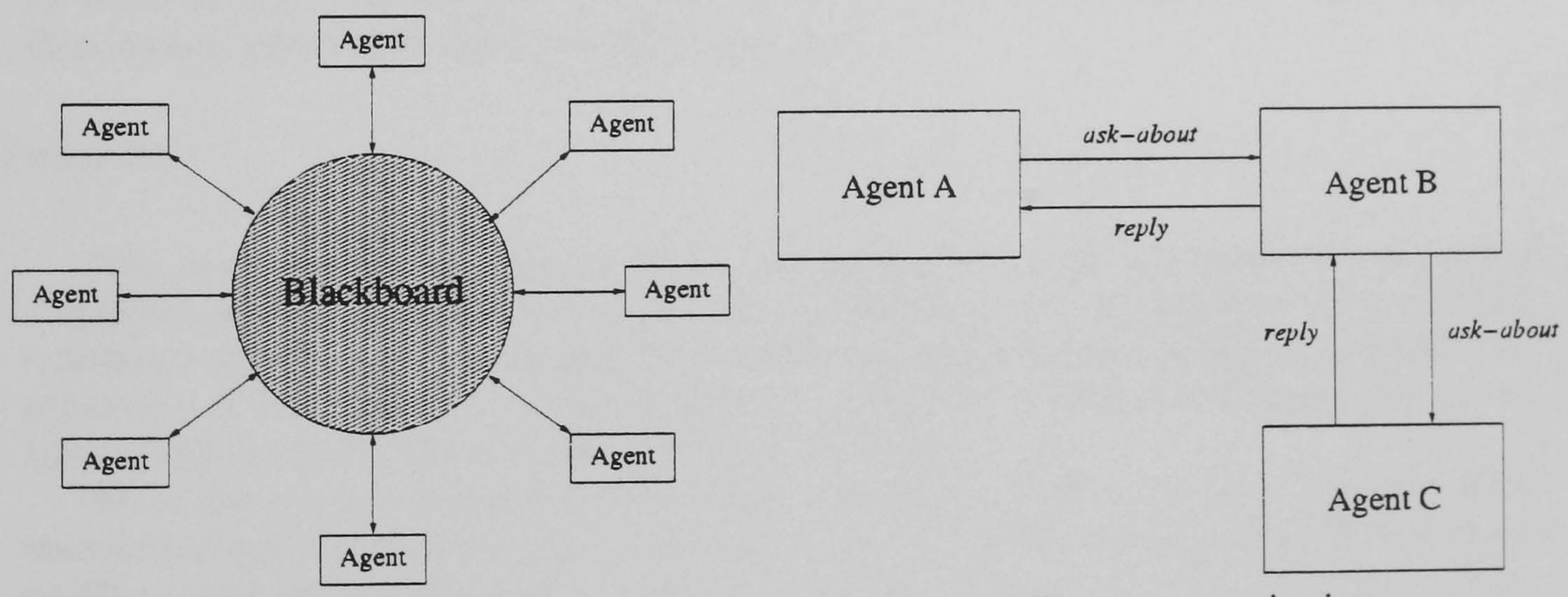
Agents need a common language in order to be able to communicate. There are several agent languages developed: KQML (Finis, Weber et al. 1993, Labrou & Finin 1997), AKL (Franzen, Ilaridi & Janson 1992, Janson & Ilaridi 1993) ...

Nwana & Wooldridge (1996) argue that agents need ontology, i.e. fundamental knowledge shared between agents, in order to communicate meaningfully.

Therefore, as per (Genesereth & Ketchpel 1994), " ...an agent communication language (ACL) can best be thought of as it consists of three parts: its vocabulary, an inner language called KIF (Knowledge Interchange Format) and an outer language called KQML (Knowledge Query and Manipulation Language). An ACL message is a KQML expression in which the 'arguments' are terms or sentences in KIF formed from words in the ACL vocabulary."

However, in the conceptual design context, it can be argued that a "full–blown" language like KQML is an "overkill" since its language is too rich for the kind of communication we need. A much simpler language based on (simple) message passing using PVM or using shared memory is sufficient. The advantage of KQML is more apparent in a context where agents need to communicate with "foreign" agents in mobile and Internet based communication.

Concerning communication between agents, the first developed was one that uses blackboard architecture (Hayes-Roth 1985, Brenner, Zarnekow & Wittig 1998), as presented in Figure 8.1(a), where all agents are able to read from and to write to a shared memory area. The other method is directed message passing from agent to agent, as shown in Figure 8.1(b) using message transport methods (e.g. PVM, MPI, etc).



(a) Blackboard system

(b) Direct communication

Figure 8.1. Agent communication methods.

8.1.4 The use of reinforcement learning

According to (Moriarty, Schultz & Grefenstette 1999), all reinforcement learning (RFL) methods share the same goal: to solve *sequential decision tasks* through trial and error interactions with the environment. In a sequential decision task, agents interact with a dynamical system by selecting actions that affect state transi-

tions to optimise some reward function. More details can be found in (Sutton & Barto 1998). The following paradigms can be used for the agent design (Moriarty et al. 1999):

- Reinforcement learning (RFL);
- Planning;
- Supervised learning.

They argue that RFL provides a flexible approach to the design of intelligent agents in situations where the significant domain knowledge is either unavailable or costly to obtain. Their discussion is however more in the domain of intelligent robots.

There is also an approach mentioned by Glover & Laguna (1997, p. 14): Fisher & Thompson (1963) introduced an innovation of altering between multiple rules at each decision node by a probabilistic strategy, which used reinforcement learning to amend the probabilities of choosing the rules according to the quality of schedules produced over multiple solution runs.

8.1.5 Agents, yes or no?

Talking about agents in the context of legacy systems, Jennings & Wooldridge (1995) say the following:

> *"Although agent based technology clearly has an important role to play in the development of leading edge compound applications, it should not be regarded as a panacea. The majority of applications which currently use agents could be solved using non–agent techniques (in most cases not as well, but in some cases better!). Thus the mere fact that a particular problem domain is open or involves legacy systems does not necessarily imply that an agent based solution is the best one (or even that it is a feasible one). As with all system designs, the ultimate choice depends upon a large number of technical and non–technical factors ..."*

And further on:

> *"The above systems, in common with the majority of other agent applications, herald a fundamentally new paradigm for developing and implementing complex systems. The traditional (idealised) software engineering model of providing a complete system specification and then implementing it in a number of rigid, deterministic components (modules) is inappropriate for the types of application for which agents are being considered. ...*
>
> *Whilst this new system paradigm offers many exciting opportunities, it has a down side which invariably places a limit on the types of application to which agents can be applied. The first major problem is that the overall system is unpredictable and non–deterministic: which agents will interact with which other is which way to achieve what cannot be predicted in advance. Even worse, there is no guarantee that dependencies between the agents can be managed effectively, since the agents are autonomous and free to make their own decisions. ... The second main disadvantage is that the behaviour and properties of the overall system cannot be fixed at design time. While a specification of the behaviour of an individual agent can be given, a corresponding specification of the system in its entirety cannot, since global behaviour necessarily emerges at run time."*

A very successful agent–based application is described in (Ygge & Akkermans 1999): they describe climate control of large buildings with many office rooms using "marked based agent approach". Agents buy and sell cooling power resources (Huberman & Clearwater 1995). A very comprehensive review of computer supported cooperative environments for engineering design is given in (Shen & Barthès 1996, Shen & Norrie 1999).

Maes (1994) presents a parallel between traditional Artificial Intelligence (AI) approaches and agents based approaches:

1. Traditional AI has focused on systems that demonstrate isolated and often advanced competences. Traditional AI provides "depth" rather than "width" in their competence. In contrast, an autonomous agent has multiple integrated competences. Typically, the competences are lower–level competences.

2. Traditional AI has focused on "closed" systems that have no direct interaction with the problem domain. Their connection with the environment is very controlled and indirect through a human operator. In contrast, an autonomous agent is an "open" system. An agent is "situated" in its environment. It is directly connected to its problem domain. It can affect or change this domain. The problem domain is typically very dynamic which means that the system has a limited amount of time to act and that unpredictable events can happen.

3. Most traditional AI systems deal with one problem at a time. Often the system does not have time constraints for solving the problem and does not have to deal with interrupts. From the system's point of view the problem domain does not change while the system is computing. In contrast, an agent is autonomous: the system is completely self–contained. It has to monitor the environment and figure out by itself what the next problem or goal to be addressed is. It has to deal with problems in a timely fashion. Typically an agent has to deal with many conflicting goals simultaneously.

4. Traditional AI focuses on the question of what knowledge a system has. AI systems have declarative "knowledge structures" that model aspects of the domain of expertise. All of the internal structures are static. In contrast, the emphasis in autonomous agent research is on what behaviour a system demonstrates when put into its environment. The internal structures of an agent are dynamic "behaviour producing" modules.

5. Finally, traditional AI is not usually concerned with the developmental aspect or the question of how the knowledge structures got there in the first place and how they should change over time. In contrast, in autonomous agent research there is a strong emphasis on "adaptation" and on a "developmental approach". This means that the system improves its own internal structures over time, based on its experience in the environment. The agent actively explores and updates its structures using an incremental, inductive learning method. The user can gradually evolve a more sophisticated system by adding structure to an already existing "working" system.

8.2 Agents developed for BAe conceptual design process

Previous sections present a fairly general framework for agents and their use. The following pages describe agents we have developed for the BAe conceptual design process. In most of the cases, the philosophy of

simple agents was followed: an agent performs only one function (similar to SIFA (Brown, Dunskus et al. 1995, Berker 1995) – single function agent). Agents in SIFA are designed to perform one function only and they have the following parameters:

Function defines what kind of work it performs;

Target defines on what parameter or object the agent has an immediate effect

Point of view specifies the perspective that the agent takes in performing its function on its target. The point of view can be cost, strength etc.

(Brown et al. 1995) argue that in this way it is much easier to construct new agents and (equally important) it is also much easier to debug agents.

The agents developed for the BAe conceptual engineering design system are schematically presented in Figure 8.2.

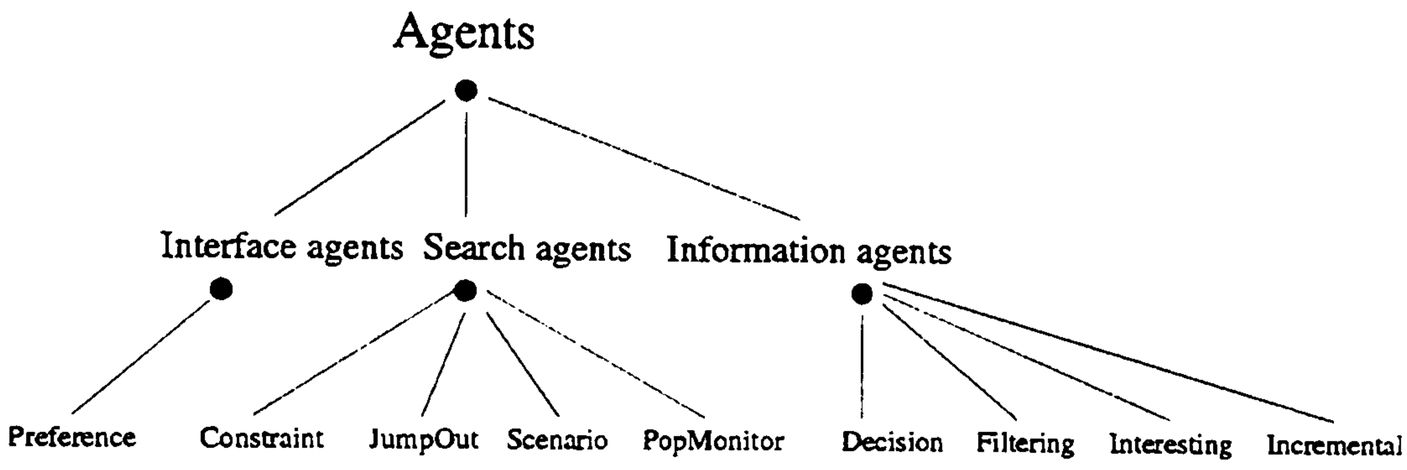


Figure 8.2. Agent hierarchy

They are going to be described in more details in the following sections.

8.3 Interface agents

Interface agents are used to decrease the complexity of the more and more sophisticated and overloaded conceptual design systems. They build an (user friendly) interface between the designer and the computer, as presented in Figure 8.3. The designer can specify:

- Quality threshold of solutions;
- If some specific situation occurs, what should an agent do;
- etc.

The idea is that the agent(s) should simplify the designer's task in the following sense:

- Look for the interesting solutions. Here the notion of "being interesting" is defined by the designer, or e.g. good solution with large Hamming (or Euclidean) distance from the majority of the population etc.

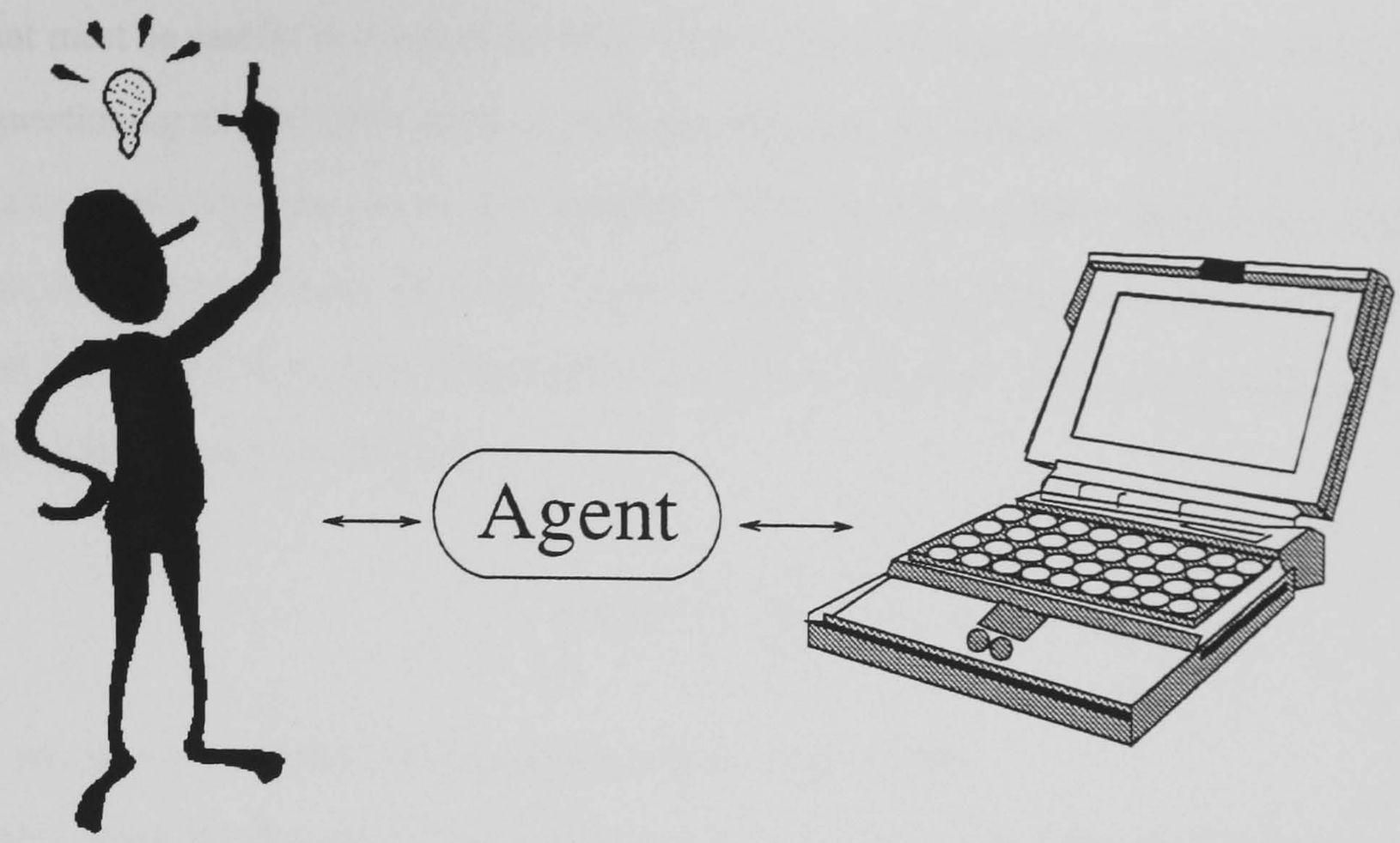


Figure 8.3. An interface agent sitting between designer and computer.

- Help the designer in a (boring) Q&A preference estimation (section 5.4, page 58) procedure (e.g. "Statement: *A* is the most important to me and *B* the least important of all the objectives" that the agent needs to transform into a series of answers to *x*?*y* questions)
- etc.

An agent has been implemented in the system that helps the designer in the preference estimation procedure. It allows the designer to specify for example the following preference order on the command line or in a file:

$$y_9 \approx y_{10} \gg y_1 \approx y_2 \approx y_6 \succ y_3 \approx y_4 \succ y_5 \gg y_7 \approx y_{11} \succ y_8$$

instead of answering the following sequence of questions:

$$y_9 \approx y_{10}$$

$$y_1 \approx y_2 \approx y_6$$

$$y_3 \approx y_4$$

$$y_7 \approx y_{11}$$

$$y_1 \succ y_2,\ y_1 \succ y_5,\ y_1 \gg y_7,\ y_1 \gg y_8,\ y_1 \ll y_9$$

$$y_3 \succ y_5,\ y_3 \gg y_7,\ y_3 \gg y_8$$

$$y_5 \gg y_7$$

$$y_5 \gg y_8$$

$$y_7 \succ y_8$$

(11 questions for 6 different classes of objectives – this is an unusually high number of questions (average from Table 5.2 on page 66 is 8.89) caused by all preferences except one going into the same direction).

The agent must be careful in interpreting given order to generate exactly those queries that it would ask in a pairwise questioning method since its set of generated questions and answers should also be used by machine based agents to establish preferences reading them from a file. In order to create exactly these question it would normally ask, it needs to simulate the whole question–answer process including computing transitive closure (as described in section 5.4) to generate the minimal number of questions. The process is simplified by the fact that the order given is always of the form:

$$y_{i_1}\rho_1 y_{i_2}\rho_2 \cdots \rho_{k-1} y_{i_k} \tag{8.1}$$

where $\rho_i \in \{\approx, \gg, \succ\}$, i.e., elements are sorted in non–increasing order.

In the initial stage, the designer will probably prefer the second method (pair–wise comparisons), but as the process goes on, specifying the complete order is easier, especially if the changes are incremental (like in the case of the incremental agent described in section 8.7.1 on page 109 below).

8.4 Search agents

Among the search agents we have the following classes of agents:

Jump out agent that searches exclusively out of boundaries;

Quality monitoring agent that monitors the quality of solutions;

Constraint agent that tries to find out what solutions can be obtained by breaking one of the constraints;

Scenario agent that solves the original problem without one of the scenarios;

Population monitoring agent that monitors the convergence of the population;

These agents are described in subsequent sections.

The following need to be considered in applying agents:

- Where to search?
- Variable 'variable' ranges;
- Constraint vs. objective space (i.e. shall we use penalty functions to transform constraints into objectives etc.);

8.4.1 Jump–out agent

This agent searches exclusively out of boundaries. It can start a new GA that works in parallel with the main one, or it can be just a quick hill–climber that starts from one of the solution, changes a randomly chosen variable to be out of defined range and then perform hill–climbing, as in Figure 8.4. Eventually, we could start modifying good solutions, not just any random population member. The idea is: Suppose $\vec{x} = (x_1, \ldots, x_n) \in \mathcal{D}$

is a population member. Create $\vec{x'} = (x'_1, \ldots, x'_n) \notin \mathcal{D}$ where at least one of $x'_i \notin [x_i^L, x_i^R]$. Start optimising $\vec{x'}$ for certain number of steps and see if anything meaningful could be obtained. Parameters of the agents limit how far outside domain can the agent go and for how many generations. It could also be specified how many individuals to create. The method used could be hill climbing, Simulated Annealing (Laarhoven & Aarts 1987), Scatter Search (Glover 1998, Laguna in press, Glover 1999) etc. This could be combined with tabu lists (Glover & Laguna 1997, Glover & Laguna in press) that remembers what regions have already been explored.

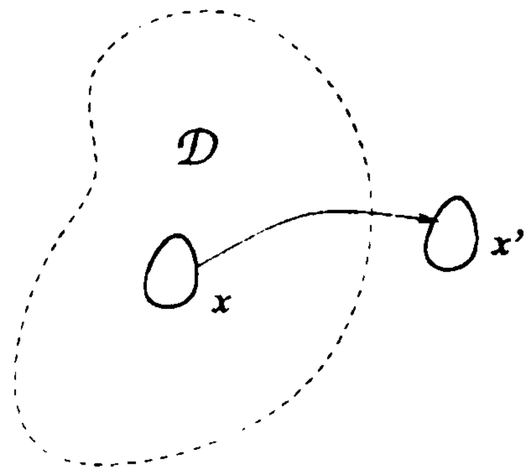


Figure 8.4. Jump out of domain

8.4.2 Quality monitoring agent

If the solution fitness is at least 90% of the best solution, this agent will notify the designer about it. The percentage of the best solution is a configurable parameter . Also, the engineer should be able to configure the system when it wants that information:

- Immediately, so that he can lead the search in that direction, or
- Afterwards, for off–line analysis;

8.4.3 Constraint agent

This agent tries to break some of the constraints, and if a good solution is obtained, it informs the designer. See Figure 8.5. Constraints can have different levels of 'breakability' assigned (say from 0 to 1, or from 'absolutely unchangeable' to 'you can do whatever you want with it'). For each solution the information how many constraints did it break (if using penalty functions for resolving constraints) need to be stored.

It is also possible to have an agent that monitors the best solutions and for each of them, the agent tries further optimisation of the solution, ignoring one of the constraints. If the obtained solution is significantly better, present it to the designer and let him decide if that constraint is necessary. The designer can mark some of the constraints as non–questionable (as before).

8.4.4 Scenario agent

Let each agent solve the original problem minus one of the scenarios (scenarios as defined in section 7.2, page 88). For m scenarios, that would mean $m+1$ parallel GAs (one eventually without any scenarios), as in

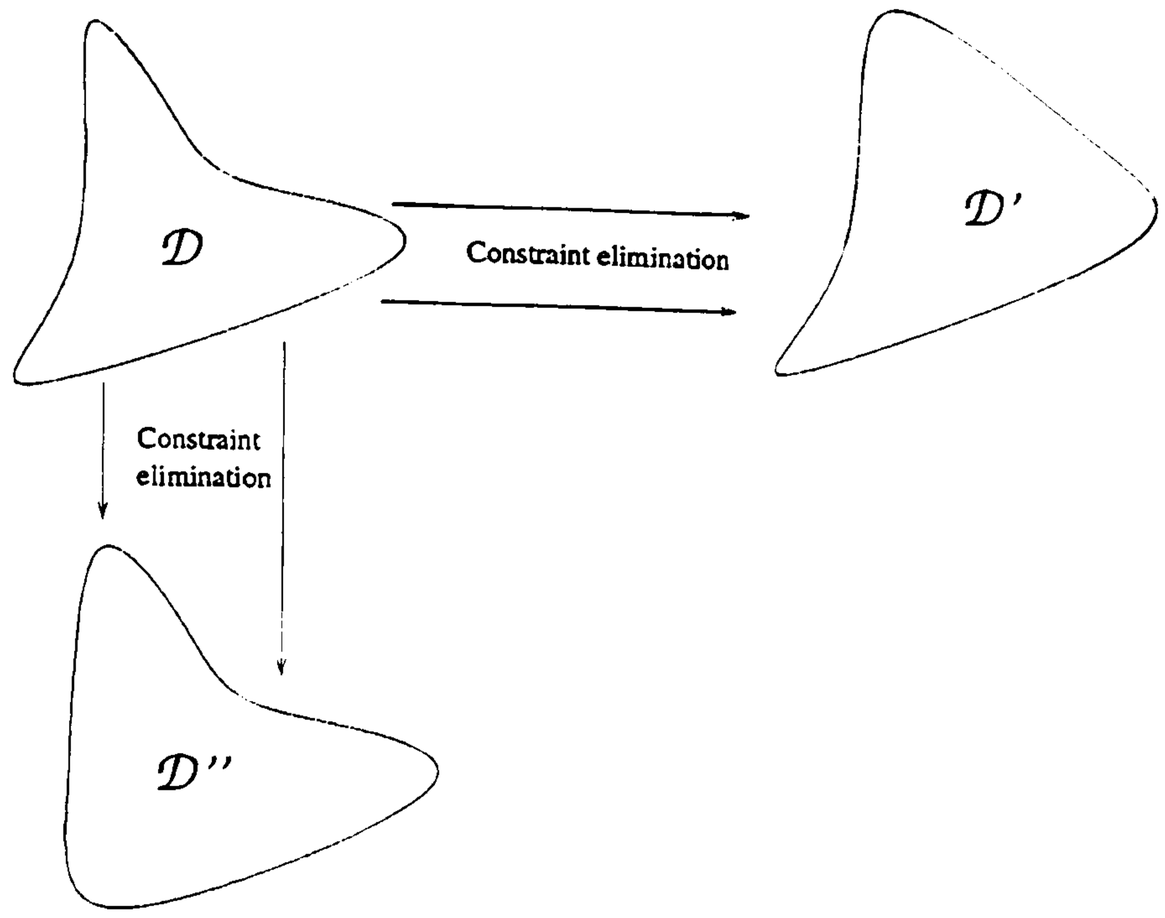


Figure 8.5. Elimination one of constraints.

Figure 8.6. This could be very costly since parallel search processes are required. However, the increased use of parallel and more and more powerful computers makes such parallel methods increasingly feasible.

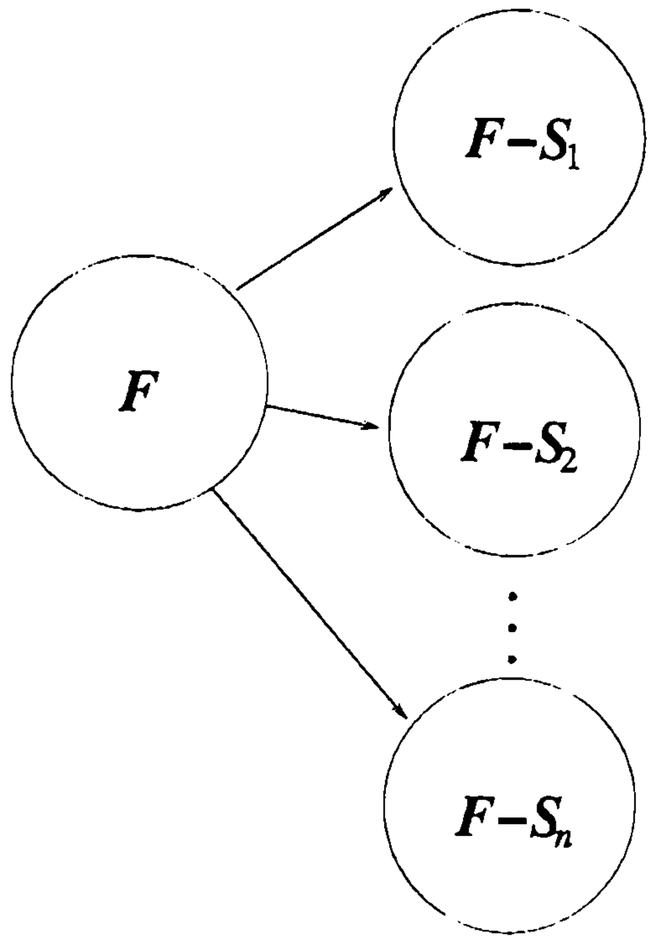


Figure 8.6. Solving original problem F minus one scenario S_i.

8.4.5 Population monitoring agent

If the GA search is too concentrated in one part of the search space, try to "jump" far away (but still within the domain $\mathcal{D}$) and start a new search there. This is illustrated in Figure 8.7. Bookkeeping about already explored regions is needed in order to avoid visiting the same region many times.

Note: There are already agents implemented that monitor the state of the population and perform an action accordingly:

- If using CHC (Eshelman 1990)[1], and if average distance in the population is less than 0.1, then

[1]According to (Whitley 1993, p. 31), CHC stands for *Cross generational elitist selection, Heterogeneous recombination and*

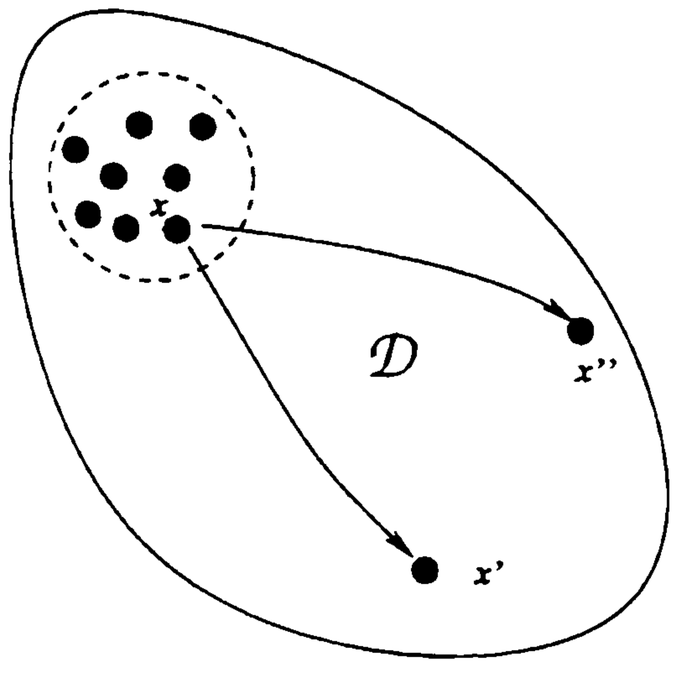


Figure 8.7. Changing region of the search space

re–initialise one part of the population. Also increase the mutation rate to keep the versatility of the population;

- When adding elements to the Pareto front, elements that are too similar to already existing elements (phenotypically or genotypically) are rejected;

Three levels of spontaneous behaviour are available:

- Machine–based agent automatically decides to try jumping out of regions, breaking constraints etc.;
- The designer only decides on the action taken;
- Interactively: agent suggests and the designer declines or accepts.

8.5 Agent cooperation

Let us consider a system with several agents, each with a task to optimise a single objective. The question is how to make them collaborate, negotiate and cooperate. Each agent is aware of the quality of its own solution. If the quality of one's solution is inferior to the quality of solution of some other agents and their solutions are contradicting, that agent should compromise and accept a worse solution from the point of view of that agent, for the benefit of other agents. In the case they cannot decide (e.g. both agents think that they have quality solutions), the designer will be asked to decide, as illustrated in Figure 8.8. Once the designer resolves a conflict, the agents need to remember the decision and try to learn from it so that the next time a similar situation happens, they can resolve the conflict among themselves without designer's intervention. Some form of voting system, where the importance of each agent also plays a certain role, can be used for resolving conflicts.

If an agent is successful, it is made more important than the others (so that good solutions usually count as more important in the negotiation process). For each "best solution" increase the value of the solution. If an agent is less successful, reduce the agent's importance or the quality of its solutions. Limits to both maximal and minimal possible importance of an agent are needed in order to keep diversity of solutions generated.

Cataclysmic mutation.

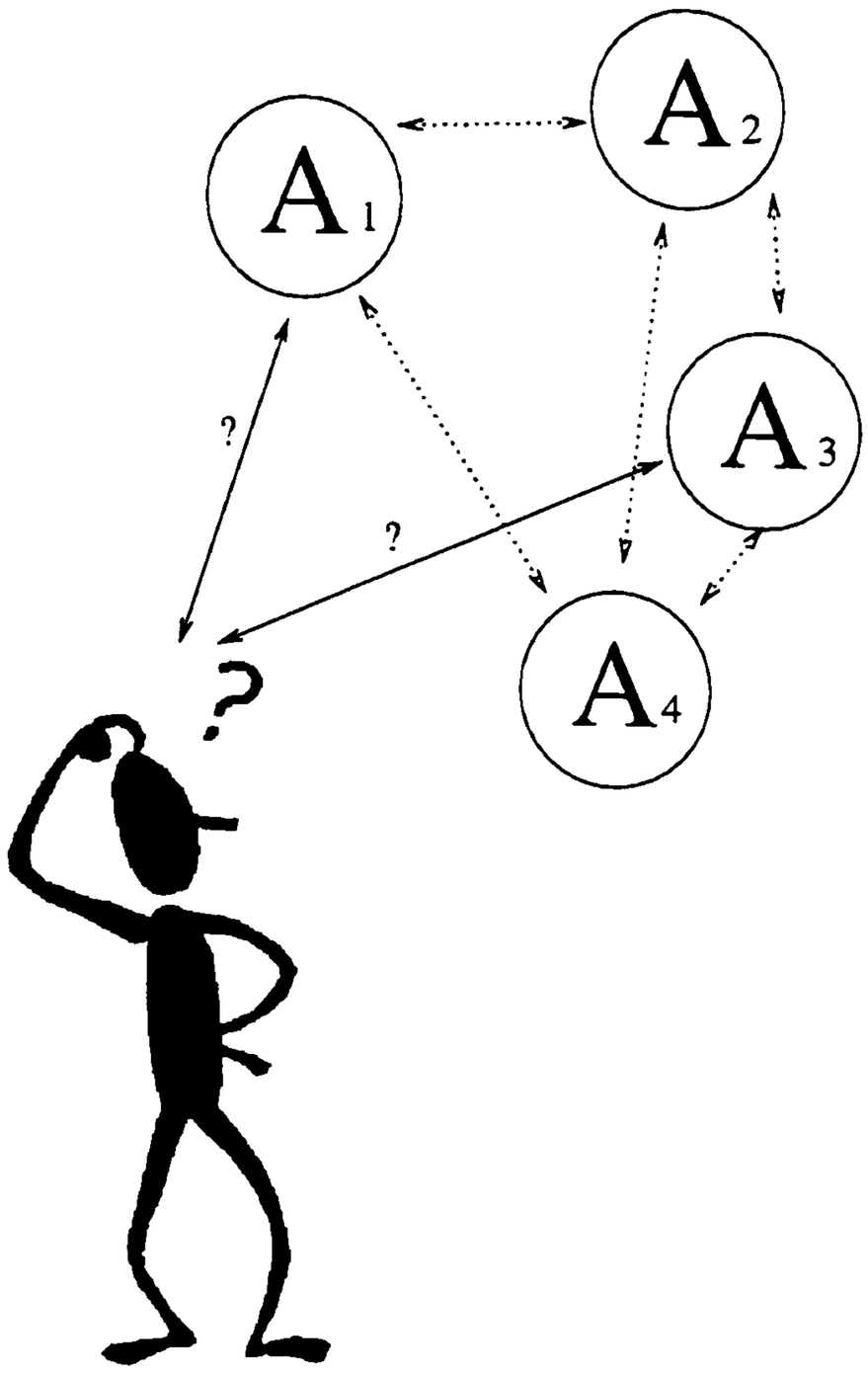


Figure 8.8. Agents cooperation and designer interaction in resolving $A_1 - A_3$ conflict.

8.5.1 Our idea of compromise

Suppose that the function to optimise is

$$(f_1(\boldsymbol{x}), \ldots, f_k(\boldsymbol{x}))$$

where each $f_i : \mathbf{R}^k \mapsto \mathbf{R}$, and suppose that there are l agents $A_1, \ldots, A_l$ optimising it. That means that each agent A_i is responsible for an objective or a set of objectives to optimise, and it returns its fitness function $\alpha_i(\boldsymbol{x})$. Each agent then computes the penalty of a given point $\boldsymbol{x}$ as presented in Figure 8.9. Penalty $p_i(\boldsymbol{x})$ is calculated as a difference of $\alpha_i(\boldsymbol{x})$ from the best solution found so far $\alpha_i(\boldsymbol{x}_*^{(i)})$ by that agent:

$$p_i(\boldsymbol{x}) = ||\alpha_i(\boldsymbol{x}) - \alpha_i(\boldsymbol{x}_*^{(i)})|| \tag{8.2}$$

for some norm $||\cdot||$. Then $p(\boldsymbol{x})$ can be computed as

$$p(\boldsymbol{x}) = \sum_{i=1}^{l} w_i \cdot p_i(\boldsymbol{x}) \tag{8.3}$$

where $\sum_i w_i = 1$ and w_i is the importance factor of each agent. The agent which has found a solutions with minimal penalty gets a reward (i.e. its weight w_i get increased by some amount δ), whereas the worst one get its weight decreased by δ.

Here is the algorithm in more details:

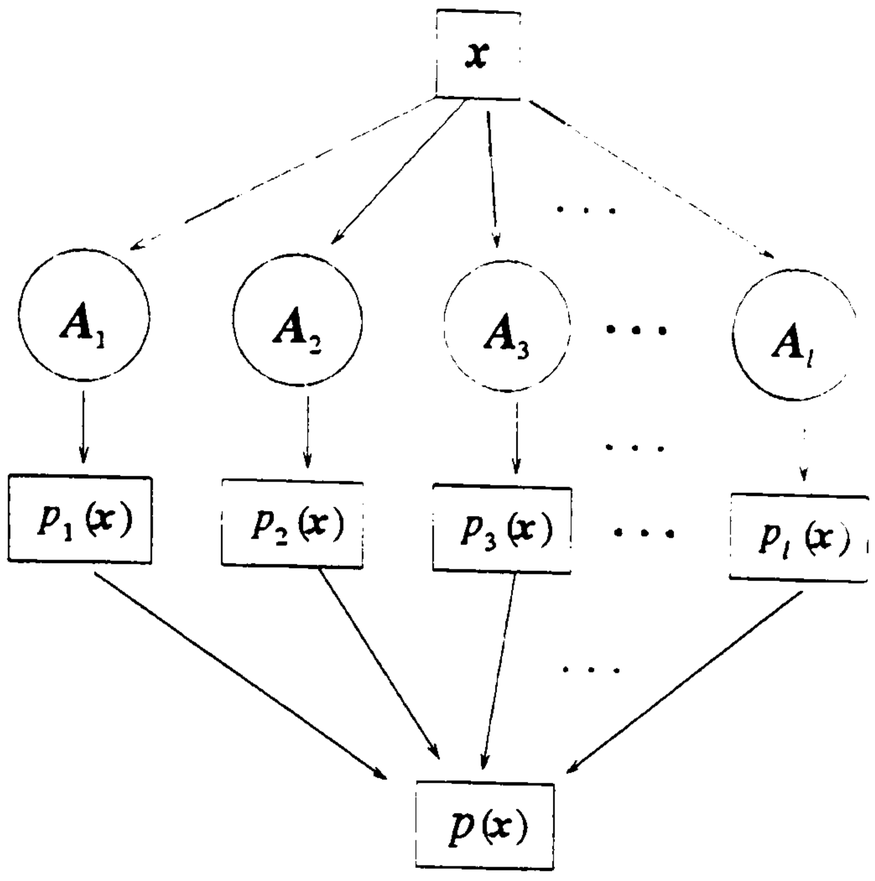


Figure 8.9. Calculating penalty of a solution.

1. Give agents a *starting point* $\boldsymbol{x}_0$ and let them optimise their fitness functions $\alpha(\boldsymbol{x})$ starting from $\boldsymbol{x}_0$.
2. For each agent's A_i returned vector $\boldsymbol{x}_*^{(i)}$, calculate a vector $(p_1(\boldsymbol{x}_*^{(i)}), \ldots, p_l(\boldsymbol{x}_*^{(i)}))$ and the function $p(\boldsymbol{x}_*^{(i)})$ using equations (8.2) and (8.3).
3. Find

$$\boldsymbol{x}_* = \arg\min_{1 \le i \le l} p(\boldsymbol{x}_*^{(i)})$$

$$\boldsymbol{x}^* = \arg\max_{1 \le i \le l} p(\boldsymbol{x}_*^{(i)})$$

   and the corresponding indices i_1 of the agent with the best solution $\boldsymbol{x}_*$ and i_2 of the agent with the worst solution $\boldsymbol{x}^*$.
4. Increase w_{i_1} by δ and decrease w_{i_2} by δ (taking into consideration the upper and lower limits);
5. Set $\boldsymbol{x}_0 = \boldsymbol{x}_*$ and go to step 1.

In the most simple case, for the penalty function $p(\boldsymbol{x})$, the following function can be used

$$p_i(\boldsymbol{x}) = \begin{cases} \alpha_i(\boldsymbol{x}) - \alpha_i(\Phi_i(\boldsymbol{x}_0)), & \text{if positive;} \\ 0, & \text{otherwise.} \end{cases} \tag{8.4}$$

where $\Phi_i(\boldsymbol{x}_0)$ is the solution obtained by agent A_i starting from point $\boldsymbol{x}_0$ and using optimisation algorithm Φ_i (genetic, algorithm, simulated annealing, tabu search, scatter search, hill climbing, downhill simplex, differential evolution (Price 1999), random search, ...). Randomness can also be introduced, similar to simulated annealing (Laarhoven 1988, Tanner 1993) methods. The randomness could also be applied in step 3 of the above algorithm in deciding on the best and the worst solution.

8.6 Information agents

This class of agents should be more intelligent then the previous classes and they should be able to make autonomous decision concerning:

- "spawning" an agent to search in a given direction;
- "killing" an agent that is not very successful;
- negotiation between agents (unless they need to consult the designer);
- recognition of novelty of a solution (eventually consulting the database of existing solutions) and turning designer's attention towards it;
- when to consult the designer;
- etc.

The agents described below belong to the class of information agents.

8.7 Closing loop: Agents in a BAe conceptual design context

In a BAe context we can apply agents in a way as presented in Figure 8.10. In the conceptual design system

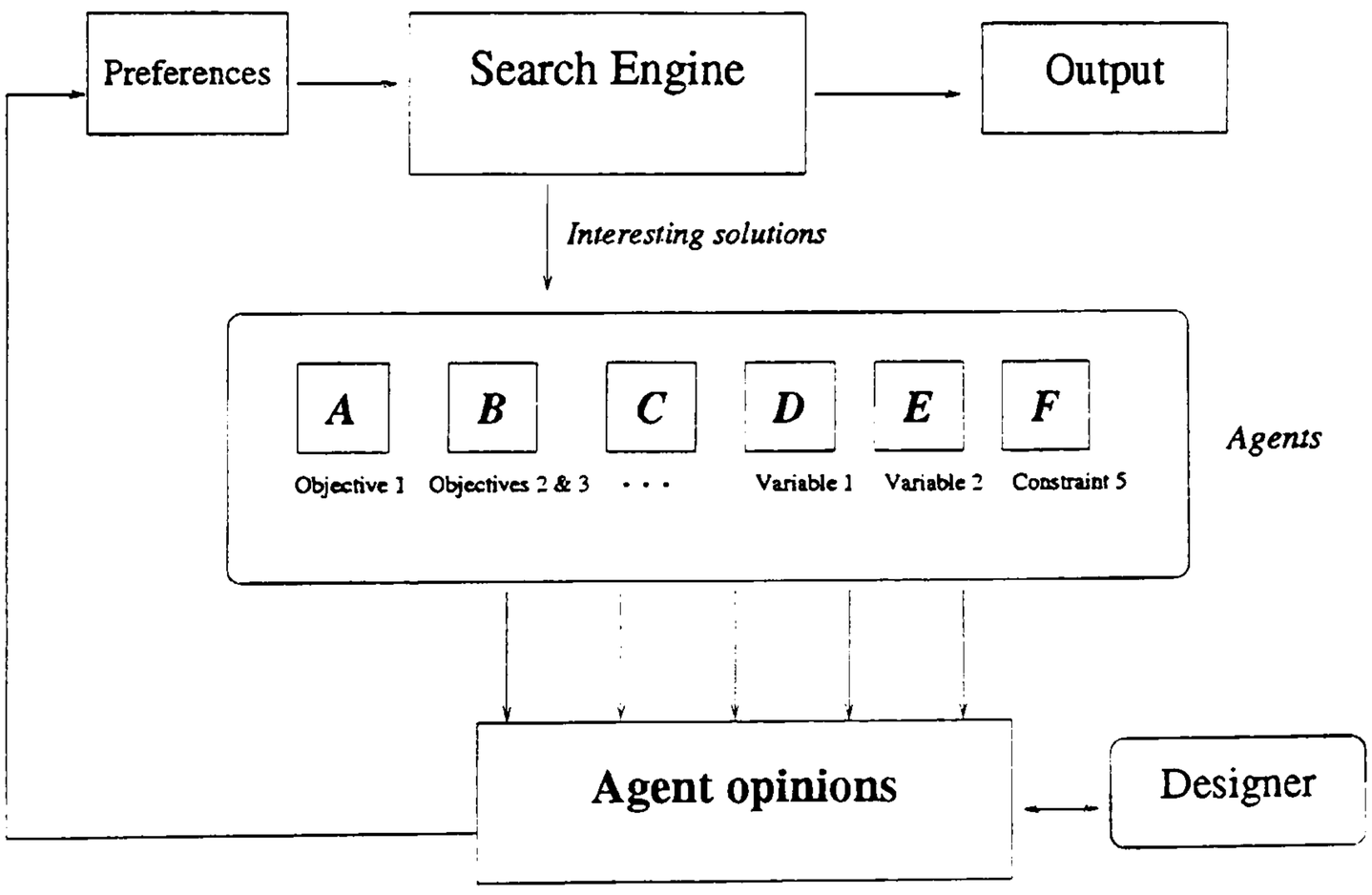


Figure 8.10. Agents in a BAe context.

developed so far, there is a standard search path: Preferences → Search Engine → Output. The new component here is a process picking up the solutions from the search engine and presenting them to the consortium of agents that look at it from the different interest or a point of view (say agent *A* monitors objective 1, agent *B* monitors objective 2, etc, agent *D* monitors variable 1 which, ideally should be between 0 and 5 or between 15 and 20 etc.). This information is presented to the designer together with some suggestions ("can we change this preference?" or "this solution path is no good for some constraints" etc.) and preferences and some

mathematical model details are being changed (with designer's approval). This connects agents nicely with the scenario concept described in the chapter 7, page 84. The agents can be employed to monitor scenarios and to analyse those that are never completely fulfilled. The unfulfilled scenarios can give an indication about the changes needed to improve the design. These changes should ideally be suggested by the agents. If the fulfilment could be achieved by simply changing preferences, agents should also suggest the necessary changes. How realistic this goal is depends mainly on the complexity of scenarios and/or interaction of the parameters. In some cases it wouldn't be possible to satisfy all constraints and scenarios, but in such a case the agent should admit that it is not able to resolve the situation in a satisfactory manner.

An agent has been implemented that monitors the fulfilment of different scenarios through a certain number of generations. In the case that some scenario was not fulfilled for that number of generations, it warns the system and the designer, about it. The autonomous decision relating to what exactly needs to be changed in order to fulfil some scenario (or constraint or objective) is very hard and therefore is best left to the designer.

The following text, section 8.7.1, describes an agent that tries to fulfil scenarios through changing preferences and variable ranges. Every time it finds an unfulfilled scenario, it suggests the changes to the designer and if approved, continues search in the modified setting.

8.7.1 Incremental agent

The incremental agent developed in this section will close the design loop as presented in Figure 8.11. In that figure it is assumed that agent and scenario values are incorporated into the fitness value as some form of penalty. In that way they give the search algorithm directions where to search.

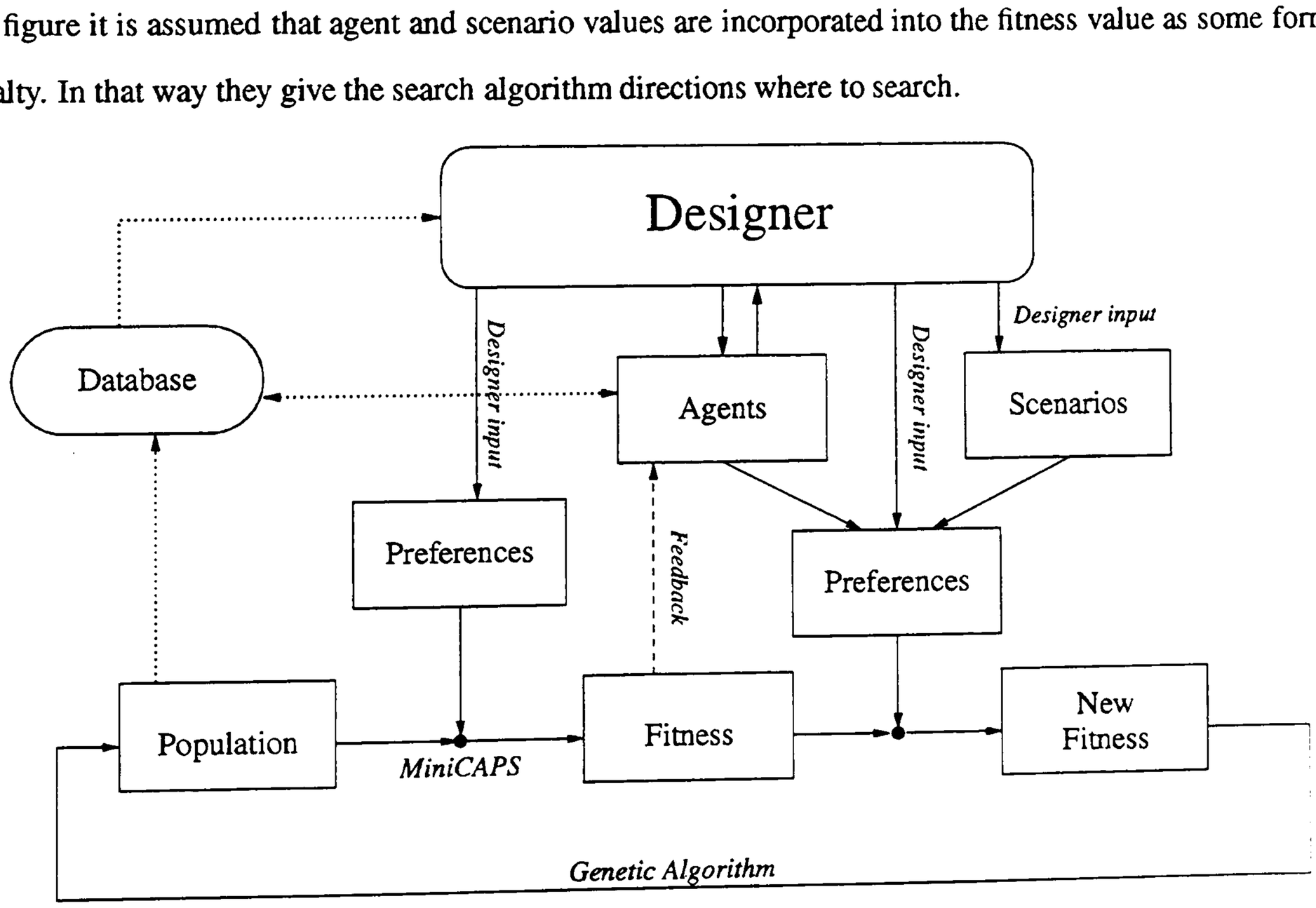


Figure 8.11. GA – Scenarios – Agents closed loop

The way the agent works is the following:

1. Use the original designer's preferences (both for objectives and for scenarios) and run optimisation process;

2. If some of the scenarios are not fulfilled, suggest increasing importance of those scenarios that are not fulfilled and repeat the search process;

3. If some scenarios are still not fulfilled although they are classified as the most important, suggest changing variable ranges (of those variables mentioned in scenarios) and repeat the search with this new setting;

4. If some scenarios are still not fulfilled, give up and report to the designer.

An example is given in section 8.7.2 below.

8.7.2 Agent–scenario example

Example 8.1 This example continues example 7.3(c) on page 91. Incremental agent method will be used here.

The scenarios are:

```
# constraints for F-111 Aardvark Bomber
y11 >= 9.74  & y11<=19.20 # Wing span
x4 <=61.07 & x4 >=48.77   # Wing plan area
y9 >=4707                 # Ferry range
y10 >= 45360              # take-off mass
x3 >= 0.75                # max cruise speed 919km/h
y1 <= 951                 # take-off run
```

The process goes as follows:

1. The original set of designer's objective preferences is:

$$y_1 \approx y_9 \approx y_{10} \approx y_{11}$$

and the original set of designer's scenario preferences is:

$$S_1 \approx S_2 \approx S_3 \approx S_4 \approx S_5 \approx S_6$$

The search process gives a solution $x_3 = 0.8710$, $x_4 = 61.07$, $y_1 = 880.457$, $y_9 = 7004.211$, $y_{10} = 30936.1328$ and $y_{11} = 19.1421$, so scenario S_4 is not fulfilled and this is noted by the agent.

2. At this stage the agent suggests increasing importance of scenario S_4 i.e.:

$$S_4 \succ S_1 \approx S_2 \approx S_3 \approx S_5 \approx S_6$$

This gives a solution: $x_3 = 0.8573, x_4 = 61.07, y_1 = 880.4575, y_9 = 7004.2085, y_{10} = 30936.1328$ and $y_{11} = 19.1421$, so scenario S_4 is still not fulfilled and is noted by the agent.

3. The agent suggests another increase of the importance of S_4 scenario i.e.:

$$S_4 \gg S_1 \approx S_2 \approx S_3 \approx S_5 \approx S_6$$

This gives a solution: $x_3 = 0.8908, x_4 = 61.07, y_1 = 880.4577, y_9 = 7004.2085, y_{10} = 30936.1367$ and $y_{11} = 19.1421$, and again scenario S_4 is not fulfilled and this is noted by the agent.

4. Since the importance of the scenario S_4 cannot be further increased, the agent suggests modifying the variable bounds. It suggests increasing variable ranges by 10% (using -o option of the RGA program described on page 37) and starting again.

5. With all equal scenario preferences there is no difference, so S_4 importance increase is suggested again.

6. For a preference setting

$$S_4 \succ S_1 \approx S_2 \approx S_3 \approx S_5 \approx S_6$$

the solution obtained finally satisfies S_4, but violates scenarios S_1, S_2 and S_6: $x_3 = 0.9062, x_4 = 86.0$, $y_1 = 1781.6176, y_9 = 6627.2627, y_{10} = 45947.1289$ and $y_{11} = 23.5521$.

7. Agent suggests

$$S_4 \succ S_1 \approx S_2 \approx S_6 \succ S_3 \approx S_5$$

which again gives a solution where S_4 is not fulfilled, so the next suggestion is

$$S_4 \gg S_1 \approx S_2 \approx S_6 \succ S_3 \approx S_5$$

This gives a solution where scenarios S_1, S_2 and S_3 are not fulfilled: $x_3 = 0.9167, x_4 = 86, y_1 = 758.8561$, $y_9 = 2897.9304$, $y_{10} = 46982.3555$ and $y_{11} = 23.5521$. At this point the agent calls the designer for further assistance.

8. From the analysis performed so far, the designer can immediately see that (using the given aeroplane model) either S_4 is feasible, or S_1 and S_2 are feasible (wing span and wing plan area are interdependent with take-off mass). At this point the designer decides to use the following preferences (the last set of preferences above)

$$S_4 \gg S_1 \approx S_2 \approx S_6 \succ S_3 \approx S_5$$

and to just increase the range of x_4 to $[20, 120]$, keeping all other to their original ranges. This gives the solution $x_3 = 0.871, x_4 = 120, y_1 = 950.9998, y_9 = 7846.061, y_{10} = 49923.9375$ and $y_{11} = 26.8328$, that violates scenarios S_1 and S_2 but is probably the best compromise in these circumstances.

9. If the designer, instead of maximising, decides to minimise take–off mass, the result obtained is $x_3 = 0.8635, x_4 = 120, y_1 = 878.0648, y_9 = 9829.6230, y_{10} = 45360.0234$, and $y_{11} = 26.8328$.

10. If the designer, out of curiosity, further increases the range of x_4 to $[20, 140]$, the solution is $x_3 = 0.8681$, $x_4 = 140, y_1 = 950.9993, y_9 = 8517.5781, y_{10} = 49452.2812$ and $y_{11} = 14.4914$, where only scenario S_2 is not fulfilled, resulting in an aeroplane design with a very large wing plan area but small wing span (probably some delta–shaped wing).

11. Etc.

As it can be seen from the above example, the use of scenario–agent–search system provides an excellent environment for interactive analysis and change of parameters in the run. It provides an almost immediate feedback about the objectives, constraints and preferences.

CHAPTER 9

Conclusion

In this thesis we have developed several techniques that we hope are going to be useful during the process of conceptual engineering design.

For the optimisation part of the design, genetic algorithms modified to handle multi–objective problems have been developed and a novel technique of weighted Pareto optimisation has been applied, as described in section 3.5.1 on page 33. A Pareto ranking method is also introduced in section 3.4, page 30. Different parameters and operators of a genetic algorithm have been tested and an optimal setting of GA parameters for the BAe problem estimated. The optimal GA characteristics are real coding, EXP mutation, SBX crossover and tournament selection of size 2. Details and justification are given in chapter 4, pages 37 on. The behaviour of weighted Pareto front size as a function of weightings vector w and threshold τ has been investigated and discussed.

Optimisation methods have been applied on the British Aerospace airframe conceptual design project using the miniCAPS module developed initially by BAe and further developed within PEDC by Dr Andrew Watson.

A module for handling scenarios has been developed. Scenarios are treated as dynamically specified constraints in fairly rich mathematical and constraint language. It enables dynamic evaluation of constraints, but also changing, deleting and adding new constraints on line. Two different versions of language have been developed:

- One, without a disjunction operator ($\vee$) and with a conjunction operator ($\wedge$) implemented as a percentage of components being satisfied;
- Second, with the full set of logical operators (and, or, not), with interpretation in the set $\{0, 1\}$.

Each method has its advantages and disadvantages, and it is not easy to say which one is the "right one". It usually depends on the context and on the expressive power of scenarios needed. Scenarios are described in chapter 7, pages 84 on.

A new method for transforming qualitative categories into quantitative measurements based on fuzzy preferences and the MCDM methods has been developed in order to simplify the transformation process. Several applications of the new preference method have been developed and described:

1. The use of preferences in optimisation:

   (a) Weighted sum based multi–objective optimisation;

   (b) Weighted Pareto method;

2. The use of preferences in co-evolutionary optimisation;

3. The use of preferences in scenario and constraint handling;

4. The use of preferences in agent based methods.

The question of complexity of the procedure has been investigated and although the worst case scenarios is quadratic i.e. $O(k^2)$ (in number of questions needed as a function of number of objectives k), the average case gives a more modest $O(k^{3/2} \cdot \ln k)$ (section 5.4.2, page 64).

The question of sensitivity of weights on the preference interpretation has also been investigated. The lower and upper limit on the weight value, depending on the preference parameters has been calculated.

The development of an agent based system for engineering design is described in chapter 8, page 94 on. Classes of agents have been developed that help the designer in the engineering design process. Certain agents monitor the population and perform appropriate actions, some agents enable different preference inputing ways and some agents suggest different actions and different preference settings according to the results of the search, scenario values and original designer's preferences. A closed loop between preferences and the results which includes the designer, has been established. Monitoring the results of scenarios and objectives and putting them back into the loop by the mean of suggested preference changes have been realised.

To the best of our knowledge, these methods and techniques are new or at least present a new application/combination of certain methods.

The research in this thesis could be further developed in the following areas:

- More levels of importance (an easy extension of the method) as already indicated in section 5.4 on page 59;
- More general interpretation function for preference levels;
- More details in parameter dependency between weighted Pareto front and the preferences;
- The integration of the two scenario methods developed in chapter 7, page 84;
- Additional classes of agents for handling preferences and for handling search process;
- The use of multiple agents and communication and negotiation between them;
- Automatic analysis of problems by agents and suggestions for changes and improvements;
- etc

This complete system with Graphical User Interface (GUI) is currently being developed in the PEDC (Parmee, Cvetković, Bonham & Mitchell 2000, Parmee, Cvetković, Bonham & Packham 2000).

The aim is the development of Interactive Engineering Design System (IEDS) (Parmee, Cvetković, Watson & Bonham 2000) graphically represented in Figure 9.1. It would include preference and agents methods developed here but also information gathering methods such as COGAs (Cluster Oriented GAs) (Bonham & Parmee 1998) and co-evolutionary optimisation methods (Parmee & Watson 1999).

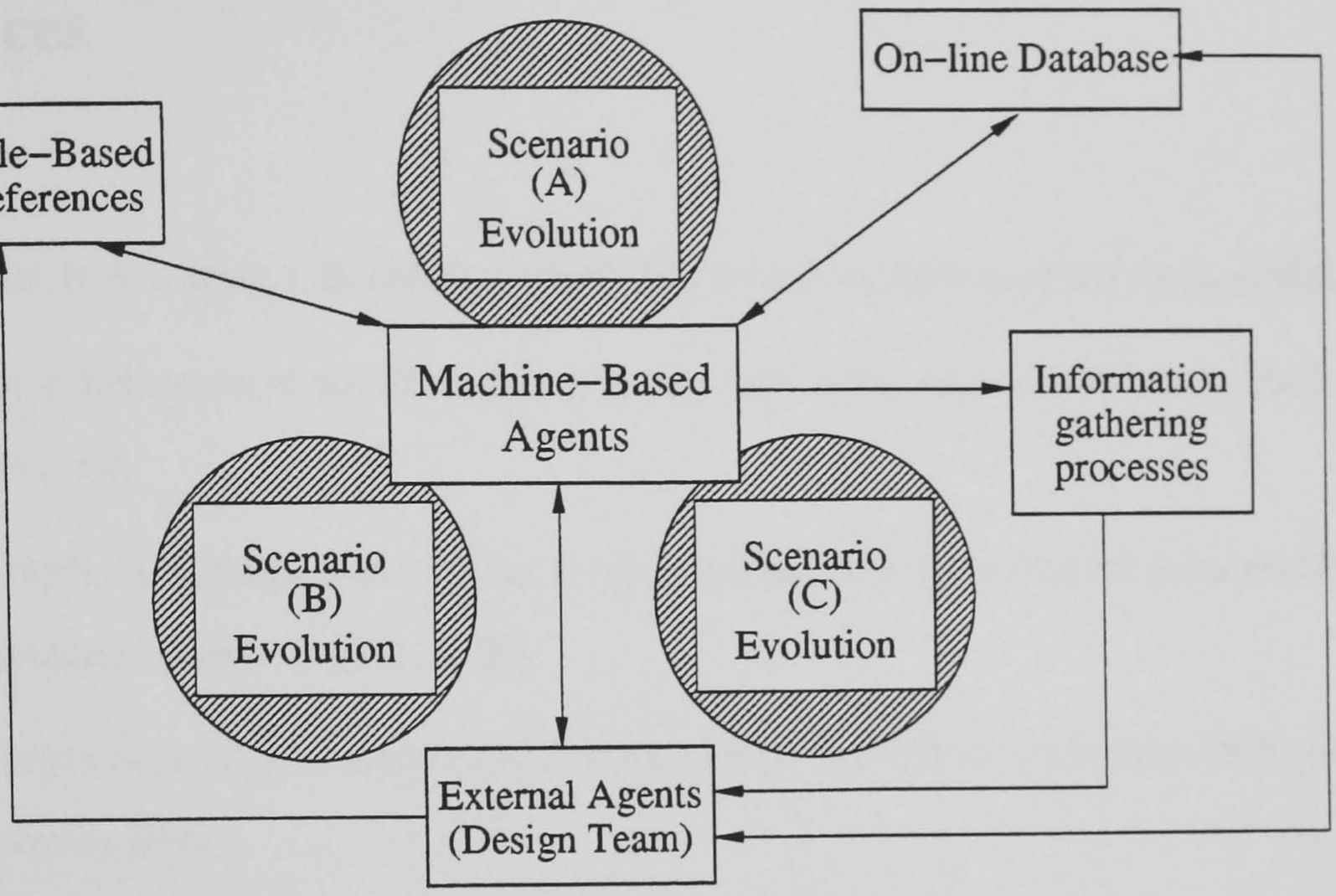


Figure 9.1. Schema of IEDS.

References

Aho, A. V., Sethi, R. & Ullman, J. D. (1987), *COMPILERS Principles, Techniques and Tools*, Addison-Wesley.

Anderson, A. R. & Belnap, Jr., N. D. (1975), *Etailment. The Logic of Relevance and Necessity*, Vol. I, Princeton University Press.

Arrow, K. J. (1950), 'A difficulty in the concept of social welfare', *Journal of Political Economy* **58**, pp. 328–346. Reprinted in (Arrow 1984, pp. 1–29).

Arrow, K. J. (1951), *Social Choice and Individual Values*, John Wiley & Sons. 2nd edition 1963, published by Yale University Press.

Arrow, K. J. (1952), 'The principle of rationality in collective decisions', *Economie Appliquée* **5**, pp. 469–484. Reprinted in (Arrow 1984, pp. 45–58).

Arrow, K. J. (1967), Values and collective decision making, *in* P. Laslett & W. G. Runciman, eds, 'Philosophy, Politics and Society', Third Series, Basil Blackwell, Oxford, pp. 215–232. Reprinted in (Arrow 1984, pp. 59–77).

Arrow, K. J. (1969), 'Tullock and an existance theorem', *Public Choice* **6**, pp. 105–12. Reprinted in (Arrow 1984, pp. 81–87).

Arrow, K. J. (1984), *Social Choice and Justice*, Vol. 1 of *Collected Papers of Keneth J. Arrow*, Basil Blackwell, Oxford.

Asenjo, F. G. & Tamburino, J. (1975), 'Logic of antinomies', *Notre Dame Journal of Formal Logic* **16**, pp. 17–44.

Asimov, I. (1982), I, robot, *in* 'St. Michael Great Science Fiction Stories', Octopus Books Limited, London, pp. 529–702.

Bäck, T. (1993), Optimal mutation rates in genetic search, *in* Forrest (1993), pp. 2–8.

Bäck, T. (1995*a*), *Evolutionary Algorithms in Theory and Practice. Evolution Strategies, Evolutionary Programming, Genetic Algorithms*, Oxford University Press, NY.

Bäck, T. (1995*b*), Generalized convergence models for tournament– and (λ, μ)–selection, *in* Eshelman (1995), pp. 2–9.

Bäck, T. & Schwefel, H.-P. (1993), 'An overview of evolutionary algorithms for parameter optimization', *Evolutionary Computation* **1**(1), pp. 1–23.

Bahrami, A. & Dagli, C. H. (1994), Design science, *in* Dagli & Kusiak (1994), pp. 7–25.

Baker, J. E. (1985), Adaptive selection methods for genetic algorithms, *in* Grefenstette (1985), pp. 101–111.

Bana e Costa, C. A. (1990*a*), An aditive value function technique with a fuzzy outranking relation for dealing with poor intercriteria preference information, *in* Bana e Costa (1990*b*), pp. 351–382.

Bana e Costa, C. A., ed. (1990*b*), *Readings in Multiple Criteria Decision Aid*, Springer–Verlag, Berlin.

Banzhaf, W., Daida, J. et al., eds (1999), *GECCO–99: Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, Orlando, Florida, USA.

Belew, R. K. & Vose, M. D., eds (1996), *Foundations of Genetic Algorithms 4*, Morgan Kaufmann, San Francisco, California.

Ben-Tal, A. (1979), Characterisation of Pareto and lexicographic optimal solutions, *in* G. Fandel & T. Gal, eds, 'Proceedings of the Third Conference on Multiple Criteria Decision Making Theory and Application', number 177 *in* 'Lecture Notes in Economics and Mathematical Systems', Springer Verlag Berlin Heidelberg, pp. 1–11.

Berker, I. (1995), Conflicts and negotiations in single function agent based design systems, Master's thesis, Worchester Polytechnic Institute, USA. http://www.cs.wpi.edu/Research/aidg/SiFA/ilan.html.

Blickle, T. & Thiele, L. (1995), A mathematical analysis of tournament selection, *in* Eshelman (1995), pp. 9–16.

Bonham, C. R. & Parmee, I. C. (1998), Cluster oriented genetic algorithms (COGAs) for the decomposition of multi-dimensional engineering design spaces, *in* B. H. V. Topping, ed., 'Advances in Computational Structures Technology', Civil-Comp Press, pp. 87–95.

Bonham, C. R. & Parmee, I. C. (1999), An investigation of exploration and exploitation within cluster oriented genetic algorithms (COGAs), *in* Banzhaf, Daida et al. (1999), pp. 1491–1497.

Branke, J., Kaußler, T. & Schmeck, H. (2000), Guidance in evolutionary multi–objective optimization, Technical report, AIFB, University of Karlruhe, Germany.

Brans, J. & Mareschal, B. (1994), 'The PROMCALC & GAIA decision support system for multicriteria decision aid', *Decision Support Systems* **12**(4/5), pp. 297–310.

Brenner, W., Zarnekow, R. & Wittig, H. (1998), *Intelligent Software Agents: Foundation and Applications*, Springer-Verlag Berlin Heidelberg.

Brown, D. C., Dunskus, B. V. et al. (1995), SINE: Support for single function agents, *in* 'Applications of AI in Engineering AIENG'95', Udine, Italy.

Carlsson, C. (1996), Active decision support systems, *in* '4th European Congress on Intelligent Techniques and Soft Computing EUFIT '96', ELITE–Foundation, Aachen, pp. 1279–1288.

Carlsson, C. (1998), Soft computing and decision support system, *in* EUF (1998).

Cha (1997), 'Chambers & Associates software engineering web – glossary pages'. Web page http://www.chambers.com.au/glossary/glossary.htm.

Chandler, S. (1999), 'Combat aircraft database', Web page http://www.btinternet.com/~military.aircraft/.

Chang, C.-L. & Lee, R. C.-T. (1971), *Symbolic Logic and Mechanical Theorem Proving*, Academic Press.

Chen, S.-J., Hwang, C.-L. & Hwang, F. P. (1992), *Fuzzy Multiple Attribute Decision Making*, Springer-Verlag Berlin Heidelberg.

Chiclana, F., Herrera, F. & Herrera-Viedma, E. (1998), 'Integrating three representation models in fuzzy multipurpose decision making based on fuzzy preference relations', *Fuzzy Sets and Systems* **97**, pp. 33–48.

Coello Coello, C. A. (1996), An Empirical Study of Evolutionary Techniques for Multiobjective Optimization in Engineering Design, PhD thesis, Tulane University.

Coello Coello, C. A. (1999), 'A comprehensive survey of evolutionary–based multiobjective optimization techniques', *Knowledge and Information System. An International Journal* 1(3), pp. 269–308.

Coello Coello, C. A. (2000), Handling preferences in evolutionary multiobjective optimization: A survey, *in* 'Congress on Evolutionary Computation (CEC'2000)', Vol. 1, IEEE Service Center, Piscataway, New Jersey, pp. 30–37.

Corne, D., Dorigo, M. & Glover, F., eds (1999), *New Ideas in Optimization*, MCGraw–Hil, London, UK.

Cvetković, D. (1993), 'Schlagwort/Lexikon Beiträge: Genetische Algorithmen', *KI – Künstliche Intelligenz* **2/93**.

Cvetković, D. & Mühlenbein, H. (1994), The optimal population size for uniform crossover and truncation selection, Technical Report GMD-AS-TR-94-11, GMD, St. Augustin, Germany.

Cvetković, D. & Parmee, I. C. (1998), Evolutionary design and multi–objective optimisation, *in* EUF (1998), pp. 397–401.

Cvetković, D. & Parmee, I. C. (1999*a*), Genetic algorithm–based multi–objective optimisation and conceptual engineering design, *in* 'Proceedings of the 1999 Congress on Evolutionary Computation – CEC99', IEEE, Washington D.C., USA, pp. 29–36.

Cvetković, D. & Parmee, I. C. (1999*b*), Genetic algorithms based systems for conceptual engineering design, *in* Lindemann, Birkhofer, Meerkamm & Vajna (1999), pp. 1035–1038.

Cvetković, D. & Parmee, I. C. (1999*c*), Use of preferences for GA–based multi–objective optimisation, *in* Banzhaf et al. (1999), pp. 1504–1509.

Cvetković, D. & Parmee, I. C. (2000*a*), The application of genetic algorithms and preferences in engineering design, Technical Report PEDC-01-2000, PEDC, University of Plymouth, Plymouth, UK.

Cvetković, D. & Parmee, I. C. (2000*b*), Designer's preferences and multi–objective preliminary design processes, *in* Parmee (2000), pp. 249–260.

Cvetković, D., Parmee, I. C. & Webb, E. (1998), Multi–objective optimisation and preliminary airframe design, *in* Parmee (1998*b*), pp. 255–267.

da Costa, N. C. A. (1974), 'On the theory of inconsistent formal systems', *Notre Dame Journal of Formal Logic* **15**, pp. 497–510.

Dagli, C. H. & Kusiak, A., eds (1994), *Intelligent Systems in Design and Manufacturing*, ASME Press, New York.

D'Ambrosio, J. G. (1996), ISMAUT tools: A software tool kit for rational tradeoffs among conflicting objectives, Technical Report CSE-TR-289-96, University of Michigan.

de Jong, K. A. (1975), Analysis of the Behaviour of a Class of Genetic Adaptive Systems, PhD thesis, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, MI.

Deb, K. (1998), Non–linear goal programming using multi–objective genetic algorithms, Technical Report 60/98, Department of Computer Science/LS11, University of Dortmund, Germany.

Deb, K. (1999*a*), Multi–objective evolutionary algorithms: Introducing bias among Pareto–optimal solutions, KanGAL report 99002, Indian Institute of Technology, Kanpur, India.

Deb, K. (1999*b*), 'Multi–objective genetic algorithms: Problem difficulties and construction of test functions', *Evolutionary Computation* **7**(3), pp. 205–230.

Deb, K. & Agrawal, R. B. (1995), 'Simulated binary crossover for continuous search space', *Complex System* **9**, pp. 115–148.

Deb, K., Agrawal, S., Pratap, A. & Meyarivan, T. (2000), A fast elitist non-dominated sorting genetic algorithm for multi–objective optimization: NSGA II, KanGAL report 200001, Indian Institute of Technology, Kanpur, India.

Deb, K. & Beyer, H.-G. (1999), Self–adaptive genetic algorithms with simulated binary crossover, Technical Report CI-61/99, Department of Computer Science, University of Dortmund, Germany.

Deb, K. & Kumar, A. (1995), 'Real–coded genetic algorithms with simulated binary crossover: Studies on multimodal and multiobjective problems', *Complex Systems* **9**(6), pp. 431–454.

Dunn, J. M. (1986), Relevance logic and entailment, *in* Gabbay & Günthner (1986), pp. 117–224.

Dym, C. L. (1994), *Engineering Design: A Synthesis of Views*, Cambridge University Press.

Eiben, A. E., Schoenauer, M. & Schwefel, H.-P., eds (1998), *Parallel Problem Solving from Nature – PPSN'98*, Springer, Amsterdam.

Eshelman, L. J. (1990), The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination, *in* Rawlins (1990), pp. 265–283.

Eshelman, L. J., ed. (1995), *Proceedings of the Sixth International Conference on Genetic Algorithms*, Morgan Kaufmann.

EUF (1998), *6th European Congress on Intelligent Techniques and Soft Computing EUFIT '98*, ELITE–Foundation, Aachen.

Finis, T., Weber, J. et al. (1993), Draft specification of the KQML agent–communication language, Technical report, The DARPA Knowledge Sharing Initiative External Interfaces Working Group.

Fishburn, P. C. (1991), 'Nontransitive preferences in decision theory', *Journal of Risk and Uncertainty* **4**, pp. 113–134.

Fishburn, P. C. (1996), Preference structures and their numerical representations, *in* 'A Conference on Order and Decision–Making ORDAL'96', Ottawa, Canada.

Fisher, H. & Thompson, G. L. (1963), Probabilistic learning combinations of local job–shop scheduling rules, *in* J. F. Muth & G. L. Thompson, eds, 'Industrial Scheduling', Prentice-Hall, pp. 225–251.

Fodor, J. & Roubens, M. (1994), *Fuzzy Preference Modelling and Multicriteria Decision Support*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

Fogel, L. J. (1962), 'Autonomous automata', *Industrial Research* **4**, pp. 14–19.

Fogel, L. J. (1964), On the Opganization of Intelligent, PhD thesis, UCLA.

Fogel, L. J., Owens, A. J. & Walsh, M. J. (1966), *Artificial Intelligence Through Simulated Evolution*, John Wiley, NY.

Fonseca, C. M. & Fleming, P. J. (1995), 'An overview of evolutionary algorithms in multiobjective optimization', *Evolutionary Computation* **3**(1), pp. 1–16.

Forrest, S., ed. (1993), *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann.

Fourman, M. P. (1985), Compaction of symbolic layout using genetic algorithms, *in* Grefenstette (1985), pp. 141–153.

Franzen, T., Ilaridi, S. & Janson, S. (1992), Overview of the Andorra Kernel Language, *in* 'Proceedings of the 2nd Workshop on Extensions to Logic Programming', Springer–Verlag.

French, S. (1986), *Decision Theory: An Introduction to the Mathematics and Rationality*, Mathematics and its Applications, Ellis Horwood Limited.

Gabbay, D. & Günthner, F., eds (1986), *Handbook of Philosophical Logic : Alternatives to Classical Logic*, Vol. III, D. Reidel.

Gehrlein, W. V., ed. (1990), *Intransitive Preferences*, Vol. 23 of *Annals of Operations Research*, Baltzer Science Publisher.

Gen, M. & Cheng, R. (1997), *Genetic Algorithms & Engineering Design*, Wiley Series in Engineering Design and Automation, J. Wiley & Sons.

Genesereth, M. R. & Ketchpel, S. P. (1994), 'Software agents', *Communications of the ACM* **37**(7), pp. 48–53, 147. Special issue on Intelligent Agents.

Gero, J. S. (1990), 'Design prototypes: A knowledge representation schema for design', *AI Magazine* **11**(4), pp. 26–36.

Glover, F. (1998), A template for scatter search and path relinking, *in* J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer & D. Snyers, eds, 'Artificial Evolution', number 1363 *in* 'LNCS', Springer, pp. 13–54.

Glover, F. (1999), Scatter search and path relinking, *in* Corne, Dorigo & Glover (1999), pp. 297–316.

Glover, F. & Laguna, M. (1997), *Tabu Search*, Kluwer Academic Publishers, Boston.

Glover, F. & Laguna, M. (in press), Tabu search, *in* P. M. Pardalos & M. G. C. Resende, eds, 'Handbook of Applied Optimization', Oxford Academic Press.

Goel, A. K. (1997), 'Design, analogy and creativity', *IEEE Expert, Intelligent Systems & Their Applications* **12**(3), pp. 62–70.

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

Grätzer, G. (1971), *Lattice Theory: First Concepts and Distributive Lattices*, W. H. Freeman, San Francisco, CA.

Grätzer, G. (1978), *General Lattice Theory*, Birkenhäuser, Basel.

Greenwood, G. W., Hu, X. S. & D'Abrosio, J. G. (1996), Fitness functions for multiple objective optimization problems: Combining preferences with Pareto ranking, *in* Belew & Vose (1996), pp. 437–455.

Grefenstette, J. J. (1986), 'Optimization of control parameters for genetic algorithms', *IEEE Transaction on Systems, Man and Cybernetics* **SMC-16**(1), pp. 122–128.

Grefenstette, J. J., ed. (1985), *Proceedings of the First International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates.

Hannson, B. (1970), *Preference Logic: Philosophical Foundations and Applications in the Philosophy of Science*, Lund.

Harik, G., Cantú-Paz, E., Goldberg, D. E. & Miller, B. L. (1999), 'The gambler's ruin problem, genetic algorithms, and the sizing of populations', *Evolutionary Computation* **7**(3), pp. 231–253.

Hayes-Roth, B. (1985), 'A blackboard architecture for control', *Artificial Intelligence* **26**, pp. 251–321.

Holland, J. H. (1975), *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Michigan. 2nd edition 1992 by MIT Press.

Horn, J. & Nafpliotis, N. (1993), Multiobjective optimization using the niched Pareto genetic algorithm, IlliGAL Report 93005, Illinois Genetic Algorithm Laboratory.

Huberman, B. A. & Clearwater, S. (1995), A multi–agent system for controlling building environments, *in* V. Lesser, ed., 'Proceedings of the First International Conference on Multi–Agent Systems, ICMAS'95', AAAI Press/ MIT Press, pp. 171–176.

Huylenbroeck, G. v. (1995), 'The conflict analysis method: Bridging the gap between ELECTRE, PROMETHEE and ORESTE', *European Journal of Operational Research* **82**, pp. 490–502.

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

Ingber, L. (1989), 'Very fast simulated annealing', *Jornal of Mathematical Computer Modelling* **12**(8), pp. 967–973.

Ingber, L. (1993), 'Simulated annealing: Practice versus theory', *Jornal of Mathematical Computer Modelling* **18**(11), pp. 29–57.

Jane's Information Group (1999), *Jane's All the World Aircraft*, Jane's Information Group.

Janson, S. & Ilaridi, S. (1993), An introduction to AKL a multi-paradigm programming language, *in* 'NATO-ASI Constraint Programming', Springer–Verlag.

Jantsch, E. (1980), *The Self–Organizing Universe: Scientific and Human Implications of the Emerging Paradigm of Evolution*, Pergamon Press, New York.

Jennings, N. R. & Wooldridge, M. J. (1995), 'Applying agent technology', *Journal of Applied Artificial Intelligence* **9**(4), pp. 357–369. Special issue on Intelligent Agents and Multi-Agent Systems.

Johnson, S. C. (1975), YACC — yet another compiler compiler, Computing Science Technical Report 32, AT&T Bell Laboratories, Murray Hill, NJ.

Keeney, R. L., Raifa, H. & Meyer, R. F. (1976), *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, John Wiley & Sons.

Kocis, L. & Whiten, W. J. (1997), 'Computational investigations of low discrepancy sequences', *ACM Transactions on Mathematical Software* **23**(2), pp. 266–294.

Kung, H. T., Luccio, F. & Preparata, F. P. (1975), 'On finding the maxima of a set of vectors', *Journal of the ACM* **22**(4), pp. 469–476.

Laarhoven, P. J. M. v. (1988), Theoretical and Computational Aspects of Simulated Annealing, PhD thesis, University of Roterdam.

Laarhoven, P. J. M. v. & Aarts, E. H. L. (1987), *Simulated Annealing: Theory and Applications*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

Labrou, Y. & Finin, T. (1997), A proposal for a new KQML specification, Technical Report TR CS-97–03, CSEE, University of Maryland Baltimore County.

Laguna, M. (in press), Scatter search, *in* P. M. Pardalos & M. G. C. Resende, eds, 'Handbook of Applied Optimization', Oxford Academic Press.

Lai, Y.-J. & Hwang, C.-L. (1996), *Fuzzy Multiple Objective Decision Making*, Springer-Verlag Berlin Heidelberg.

Lin, J. G. (1976), 'Maximal vectors and multi–objective optimization', *Journal of Optimization Theory and Application* **18**(1), pp. 41–64.

Lindemann, U., Birkhofer, H., Meerkamm, H. & Vajna, S., eds (1999), *Proceedings of the 12th International Conference on Engineering Design ICED'99*, TU München, München, Germany.

Lootsma, F. A. (1996), 'A model for the relative importance of the criteria in the multiplicative AHP and SMART', *European Journal of Operational Research* **94**(3), pp. 467–476.

Lootsma, F. A. (1997*a*), *Fuzzy Logic for Planning and Decision Making*, Delft University of Technology, The Netherlands. Lecture Notes a197.

Lootsma, F. A. (1997*b*), 'Multicriteria decision analysis in a decision tree', *European Journal of Operational Research* **101**, pp. 442–451.

Loveland, D. W. (1978), *Automated Theorem Proving: A Logical Basis*, North-Holland, Amsterdam.

Luce, R. D. & Raiffa, H. (1957), *Games and Decisions: Introduction and Critical Survey*, John Wiley & Sons.

Maes, P. (1994), 'Modelling adaptive autonomous agents', *Artificial Life Journal* **1**(1&2).

Mahfoud, S. W. (1995), Niching Methods for Genetic Algorithms, PhD thesis, University of Illinois at Urbana–Champaign.

Martin, R. M. (1963), *Intension and Decision. A Philosophical Study*, Prentice–Hall Inc.

Matsumoto, M. & Nishimura, T. (1998), 'Mersenne twister: A 623–dimensionally equidistributed uniform pseudorandom number generator', *ACM Transactions on Modeling and Computer Simulations* **8**(1), pp. 3–30. Special Issue on Uniform Random Number Generators.

May, K. O. (1954), 'Intransitivity, utility and the aggregation of preference patterns', *Econometrica* **22**, pp. 1–13.

Michalewicz, Z. (1995), A survey of constraint handling techniques in evolutionary computation methods, *in* 'Proceedings of the 4th Annual Conference on Evolutionary Programming', MIT Press, pp. 135–155.

Miller, G. A. (1956), 'The magical number seven, plus or minus two: Some limits on our capacity for processing information', *The Psychological Review* **63**(2), pp. 81–97.

Moriarty, D. E., Schultz, A. C. & Grefenstette, J. J. (1999), 'Evolutionary algorithms for reinforcement learning', *Journal of Artificial Intelligence Research* **11**, pp. 241–276.

Mühlenbein, H. & Schlierkamp-Voosen, D. (1993*a*), 'Analysis of selection, mutation and recombination in genetic algorithms', *Neural Network World* **3**, pp. 907–933.

Mühlenbein, H. & Schlierkamp-Voosen, D. (1993*b*), 'Predictive models for the breeder genetic algorithm I: Continuous parameter optimization', *Evolutionary Computations* **1**(1), pp. 25–49.

Mühlenbein, H. & Schlierkamp-Voosen, D. (1993*c*), 'The science of breeding and its application to the breeder genetic algorithm (BGA)', *Evolutionary Computation* **1**(4), pp. 335–360.

Murata, T., Ishibuchi, H. & Gen, M. (1998), Random weights in multi–objective genetic algorithms, *in* '2nd International Conference on Engineering Design and Automation (EDA'98)', Maui, Hawaii.

Nisbett, R. E. & Wilson, T. D. (1977), 'Telling more then we can know: Verbal reports on mental processes', *Psychological Review* **84**(3), pp. 231–259.

Nwana, H. S., Lee, L. & Jennings, N. R. (1996), 'Co-ordination in software agent systems', *BT Technical Journal* **14**(4), pp. 79–89.

Nwana, H. S. & Wooldridge, M. J. (1996), 'Software agent technologies', *BT Technical Journal* **14**(4), pp. 68–79.

Osyczka, A. (1984), *Multicriterion Optimization in Engineering with FORTRAN Programs*, Ellis Horwood Series in Engineering Science, Ellis Horwood Chichester, UK.

Pahl, G. & Beitz, W. (1996), *Engineering Design: A Systematic Approach*, 2 edn, Springer-Verlag, London.

Parmee, I. C. (1997), Strategies for the integration of evolutionary/adaptive search with the engineering design process, *in* D. Dasgupta & Z. Michalewicz, eds, 'Evolutionary Algorithms in Engineering Applications', Springer Verlag, pp. 453–477.

Parmee, I. C. (1998*a*), Exploring the design potential of evolutionary/adaptive search and other computational intelligence technologies, in *ACDM'98* (Parmee 1998*b*), pp. 27–42.

Parmee, I. C. & Bonham, C. R. (1998), Supporting innovative and creative design using interactive designer / evolutionary computing strategies, *in* J. S. Gero & M. L. Laher, eds, 'Computation Models of Creative Design Conference IV', University of Sydney, Key Centre of Design Computing and Cognition, University of Sydney, Sydney, Australia, Heron Island, Australia.

Parmee, I. C., Cvetković, D., Bonham, C. R. & Mitchell, D. (2000), Towards interactive evolutionary design systems for the satisfaction of multiple and changing objectives, *in* 'European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2000)', Barcelona, Spain.

Parmee, I. C., Cvetković, D., Bonham, C. R. & Packham, I. S. (2000), Introducing prototype interactive evolutionary systems for ill–defined multi–objective design environments, Technical report, PEDC, University of Plymouth, Plymouth, UK. Submitted to 'Advances in Engineering Software'.

Parmee, I. C., Cvetković, D., Watson, A. H. & Bonham, C. R. (2000), 'Multi–objective satisfaction within an interactive evolutionary design environment', *Evolutionary Computation* **8**(2), pp. 197–222.

Parmee, I. C. & Denham, M. J. (1994), The integration of adaptive search techniques with current engineering design practice, *in* I. C. Parmee, ed., 'Adaptive Computing in Engineering Design and Control '94', Plymouth Engineering Design Centre, pp. 1–14.

Parmee, I. C., ed. (1998*b*), *Adaptive Computing in Design and Manufacture: The Integration of Evolutionary and Adaptive Computing Technologies with Product/System Design and Realisation. Proceedings of the 3rd Conference on Adaptive Computing in Design and Manufacture (ACDM'98)*, Springer–Verlag.

Parmee, I. C., ed. (2000), *Evolutionary Design and Manufacture: Selected Papers from ACDM'00*, PEDC, University of Plymouth, Springer, London, Plymouth.

Parmee, I. C., Johnson, M. & Burt, S. (1994), Techniques to aid global search in engineering design, *in* F. D. Anger, R. V. Rodrigez & M. Ali, eds, 'Industrial and Engineering Applications of Artificial Intelligence and Expert Systems IAE/AIE '94', Gordon and Breach Science Publishers, Austin, Texas.

Parmee, I. C. & Purchase, G. (1997), Integrating computational intelligence technologies with design and manufacturing team practice, *in* 'Proceedings of Intelligent Design in Engineering Applications Symposium (IDEA'97)', Aachen, Germany, pp. 95–99.

Parmee, I. C. & Watson, A. H. (1999), Preliminary airframe design using co–evolutionary multiobjective genetic algorithms, *in* Banzhaf et al. (1999), pp. 1657–1665.

Peace, G. S. (1993), *Taguchi Methods: A Hands–On Approach*, Addison–Wesley, Reading, MA.

Phadke, M. S. (1989), *Quality Engineering Using Robust Design*, Prentice-Hall International.

Price, K. (1999), An introduction to differential evolution, *in* Corne et al. (1999), pp. 79–108.

Priest, G., Routley, R. & Norman, J., eds (1989), *Paraconsistent Logic. Essays on the Inconsistent*, Philosophia Verlag, München, Germany.

Pronzato, L., Wynn, H. P. & Zhigljavsky, A. A. (2000), *Dynamic Search: Applications of Dynamical Systems in Search and Optimization*, Chapman &Hall/CRC, Boca Raton, Florida, USA.

Quagliarella, D., Périaux, J., Poloni, C. & Winter, G., eds (1997), *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science. Recent Advances and Industrial Applications*, John Wiley & Sons.

Rawlins, G. J. E., ed. (1990), *Foundations of Genetic Algorithms 1*, Morgan Kaufmann.

Rechenberg, I. (1973), *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Fromman–Holzboog, Stuttgart.

Rechenberg, I. (1994), *Evolutionsstrategie '94*, Fromman–Holzboog, Stuttgart.

Richardson, J. T., Palmer, M. R., Liepins, G. & Hilliard, M. (1989), Some guidelines for genetic algorithms with penalty functions, *in* J. D. Schaffer, ed., 'Proceedings of the Third International Conference on Genetic Algorithm', pp. 191–197.

Roy, B. (1990*a*), Decision–aid and decision–making, *in* Bana e Costa (1990*b*), pp. 17–35.

Roy, B. (1990*b*), The outranking approach and the foundation of ELECTRE method, *in* Bana e Costa (1990*b*), pp. 156–183.

Schaffer, J. D. (1984), Some Experiments in Machine Learning using Vector Evaluated Genetic Algorithm, PhD thesis, Deparment of Electrical Engineering, Vanderbilt University, Nashville. TCGA file No. 00314.

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

Schaffer, J. D., Caruana, R. A., Eshelman, L. J. & Das, R. (1989), A study of control parameters affecting online performance of genetic algorithms for function optimization, *in* Schaffer (1989), pp. 51–60.

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

Schwefel, H.-P. (1977), *Numerische Optimierung von Computer–Modellen mittels der Evolutionsstrategie*, Birkhäuser Verlag, Basel.

Schwefel, H.-P. & Bäck, T. (1997), Artificial evolution: How and why?, *in* Quagliarella, Périaux, Poloni & Winter (1997), pp. 1–19.

Scott, M. J. & Antonsson, E. K. (1999), 'Arrow's theorem and engineering design decision making', *Research in Engineering Design* **11**(4), pp. 218–228.

Sen, P. & Yang, J.-B. (1998), *Multiple Criteria Decision Support in Engineering Design*, Springer–Verlag, London.

Sharpe, M. (1999), *Attack and Interceptor Jets*, Dempsey-Parr Book.

Shen, W. & Barthès, J.-P. A. (1996), Computer supported cooperative environments for engineering design: A review, Technical Report 96-122, CNRS UMR Heudiasyc Universitè de Technologie de Compiègne, France.

Shen, W. & Norrie, D. H. (1999), 'Agent–based systems for intelligent manufacturing: A state-of-the-art survey', *Knowledge and Information System. An International Journal* **1**(2), pp. 129–156. http://sern.cpsc.ucalgary.ca/CAG/publications/abm.htm.

Slovic, P. & Lichtenstein, S. (1971), 'Comparison of Bayesian and regression approaches to the study of information processes in judgement', *Organizational Behavior and Human Performance* **6**, pp. 649–744.

Srinivas, N. & Deb, K. (1995), 'Multiobjective optimization using nondominated sorting in genetic algorithms', *Evolutionary Computation* **2**(3), pp. 221–248.

Stenmark, D. (1999), 'Evaluation of intelligent software agents', Web page http://w3.informatik.gu.se/~dixi/agent/agent.htm.

Storn, R. & Price, K. (1995), Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces, Technical Report TR-95-012, ICSI, University of Berkeley.

Suh, N. P. (1990), *The Principles of Design*, number 7 *in* 'Oxford Series on Advanced Manufacturing', Oxford University Press, New York.

Sutton, R. S. & Barto, A. G. (1998), *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, USA.

Sycara, K. (1991), Cooperative negotiation in concurrent engineering design, *in* D. Sriram, R. Logcher & S. Fukuda, eds, 'Computer–Aided Cooperative Product Development', Springer Verlag.

Sycara, K., Decker, K. et al. (1996), 'Distributed intelligent agent', *IEEE Expert, Intelligent Systems & Their Applications* **11**(6), pp. 36–46.

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

Syswerda, G. (1990), A study of reproduction in generational and steady–state genetic algorithms, *in* Rawlins (1990), pp. 94–101.

Tanner, M. A. (1993), *Tools for Statistical Inference: Methods for the Exploration of Posterior Distributions and Likelihood Functions*, Springer Series in Statistics, second edn, Springer Verlag, New York.

Tversky, A. (1969), 'Intransitivity of preferences', *Psychological Review* **76**, pp. 31–48.

Urquhart, A. (1986), Many–valued logic, *in* Gabbay & Günthner (1986), pp. 71–116.

Valenzuela-Rendón, M. & Uresti-Chare, E. (1997), A non–generational genetic algorithm for multiobjective optimization, *in* T. Bäck, ed., 'Proceedings of the Seventh International Conference on Genetic Algorithms', Morgan Kaufmann, pp. 658–665.

Veldhuizen, D. A. V. (1999), Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations, PhD thesis, Air Force Institute of Technology, Wright–Paterson AFB.

Veldhuizen, D. A. V. & Lamont, G. B. (1998), Multiobjective evolutionary algorithm research: A history and analysis, Technical Report TR-98-03, Air Force Institute of Technology, Wright–Paterson AFB.

Vincke, P. (1990), Basic concepts of preference modelling, *in* Bana e Costa (1990*b*), pp. 101–118.

Voigt, H.-M., Mühlenbein, H. & Cvetković, D. (1995), Fuzzy recombination for the breeder genetic algorithm, *in* Eshelman (1995), pp. 104–111.

von Wright, G. H. (1963), 'The logic of preference. An essay', Edinburgh.

von Wright, G. H. (1972), 'The logic of preference reconsidered', *Theory and Decision* **3**, pp. 140–167. Reprinted in (von Wright 1984).

von Wright, G. H. (1984), *Philosophical Logic: Philosophical Papers*, Cornell University Press, Ithaca, NY.

Warshall, S. (1962), 'A theorem on Boolean matrices', *Journal of the ACM* **9**(1), pp. 11–12.

Watt, S. N. K. (1996), 'Artificial societies and psychological agents', *BT Technical Journal* **14**(4), pp. 89–97.

Webb, E. (1997), MINICAPS – a simplified version of CAPS for use as a research tool, Unclassified Report BAe-WOA-RP-GEN-11313, British Aerospace.

Whitley, D. (1989), The genitor algorithm and selection presssure: Why rank–based allocation of reproductive tasks is best, *in* Schaffer (1989), pp. 116–121.

Whitley, D. (1993), A genetic algorithm tutorial, Technical Report CS-93-103, Colorado State University.

Whitley, D. & Kauth, J. (1988), Genitor: A different genetic algorithm, Technical Report CS-88-101, Colorado State University.

Wolpert, D. H. & Macready, W. G. (1997), 'No free lunch theorems for optimization', *IEEE Transactions on Evolutionary Computation* **1**(1), pp. 67–82.

Wooldridge, M. J. & Jennings, N. R. (1995), 'Intelligent agents: Theory and practice', *Knowledge Engineering Review* **10**(2), pp. 115–152.

Ygge, F. & Akkermans, H. (1999), 'Decentralized markets versus central control: A comparative study', *Journal of Artificial Intelligence Research* **11**, pp. 301–333.

Zadeh, L. A. (1973), 'Outline of a new approach to the analysis of complex systems and decision processes', *IEEE Transaction on Systems, Man and Cybernetics* **SMC-2**, pp. 28–44.

Zadeh, L. A. (1975), 'The concept of linguistic variable and its application to approximate reasoning', *Information Sciences* . Part I in volume 8, pp. 199-249; Part II in volume 8, pp. 301–357; Part III in volume 9, pp. 43-80.

Zitzler, E. (1999), Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications, PhD thesis, ETH Zürich, Switzerland. TIK–Schriftenreihe Nr. 30.

Zitzler, E., Deb, K. & Thiele, L. (2000), 'Comparison of multiobjective evolutionary algorithms: Empirical results', *Evolutionary Computation* **8**(2), pp. 173–195.

Zitzler, E. & Thiele, L. (1999), 'Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach', *IEEE Transactions on Evolutionary Computation* **3**(4), pp. 257–271.

APPENDIX A

Definability of orders

It can be noted that our definition of Pareto front (definition 3.9 section 3.5.1 on page 33) uses non–strict partial order $\geq$ (defined by (A.1)) instead of (more usual) strict partial order $>$ defined by (A.2):

Definition A.1 We say that (in object space) the vector $\boldsymbol{x} = (x_1, \ldots, x_k)$ is *non–dominated* by vector $y = (y_1, \ldots, y_k)$, denoted $\boldsymbol{x} \succeq \boldsymbol{y}$, if $x_i \geq y_i$ for all $1 \leq i \leq k$.

The following quotation from (Lin 1976) gives some form of justification:

> *A relation* $\sqsupset$ *is a* partial order *in a set S of elements (and S is partially ordered by succ) if* $\sqsupset$ *is transitive, antisymmetric and reflexive in S; a partial order* $\sqsupset$ *is a* total order *in S (and S is totally ordered by succ) if, for every pair of elements x and y in S, either* $x \sqsupset y$ *or* $y \sqsupset x$. *A relation* $\sqsupset$ *is a* strict partial order *in a set S if it is transitive and antireflexive in S. A relation* $\sqsupset$ *is* transitive *in S if* $x \sqsupset y$ *and* $y \sqsupset z$ *for elements x, y and z of S, then* $x \sqsupset z$; *it is* antisymmetric *if* $x \sqsupset y$ *and* $y \sqsupset x$ *for elements x and y of S, then* $x = y$; *it is* reflexive *if* $x \sqsupset x$ *for every element x of S; it is* antireflexive *if, for no element x of S, does* $x \sqsupset x$ *hold.*
>
> *A relation is* asymmetric *in S if, for no elements x and y of S, do* $x \sqsupset y$ *and* $y \sqsupset x$ *hold simultaneously. If* $\sqsupset$ *is an asymmetric transitive relation in S and if* $x \sqsupseteq y$ *is defined to mean that either* $x \sqsupset y$ *or* $x = y$, *then* $\sqsupseteq$ *is a* partial order *in S. Conversely, if* $x \sqsupset y$ *is a partial order in S and if* $x \sqsupset' y$ *is defined to mean that* $x \sqsupset y$ *and* $x \neq y$, *then* $\sqsupset'$ *is an asymmetric transitive relation in S. It is shown that a transitive relation is asymmetric iff it is antireflexive. Hence, an asymmetric transitive relation is a* strict partial order *and* vice versa. *Consequently,* a given strict partial order in a set defines a partial order in the set, and conversely. *In other words, the ordering of a set by a* strict partial order *is equivalent to the ordering by a partial order if both relations are definable by each other (in the above manner) ...*

Lin (1976) also distinguish between 3 orders on k–dimensional vectors:

$$\boldsymbol{x} \geq \boldsymbol{y} \quad \text{if and only if} \quad (\forall i \leq k)(x_i \geq y_i) \tag{A.1}$$

$$\boldsymbol{x} > \boldsymbol{y} \quad \text{if and only if} \quad (\forall i \leq k)(x_i > y_i) \tag{A.2}$$

$$\boldsymbol{x} \geqslant \boldsymbol{y} \quad \text{if and only if} \quad (\boldsymbol{x} \geq \boldsymbol{y}) \wedge (\exists j \leq k)(x_j > y_j) \tag{A.3}$$

This quotation gives the explanation why Lin only considers strict orders (A.2) and (A.3). However, since $\geq$ and $\geqslant$ *are* definable in terms of each other in above sense (i.e. $\boldsymbol{x} \geqslant \boldsymbol{y}$ iff $\boldsymbol{x} \geq \boldsymbol{y}$ and $\boldsymbol{x} \neq \boldsymbol{y}$), we can use the order $\geq$ with the same effect.

APPENDIX B

C code for preferences

The code give bellow compute weight factors associated with preferences. It is enough to call the function `ComputeWeights(nobj, weights)`, where `nobj` is the number of objectives and `weights` is the vector which returns the weights of the objectives according to the user preferences.

```
/*
  Ask the user about the preferences, compute preference order and weights.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define SEP ","

/* for much more important down to much less important */
#define MMI 1.9      /* much more important */
#define MI 1.3       /* more important */
#define EI 1.0       /* equally important */
#define LI 0.7       /* less important */
#define MLI 0.1      /* much less important */
#define NCMP EI      /* not comparable  - the same as equally important */
#define NN_MAX 4     /* number of comparisons */

#define RMMI 2       /* much more important for Ra */
#define RMI 1        /* more important for Ra */

struct OBJECTIVE {
  int *equ;      /* list of objectives equally important:
                    binary mask - position j is 1 if objective j is equally
                    important as the current one, 0 otherwise */
  double score;  /* entering score computed as Sc(a) = sum_{b !=a} R(a, b) */
  double weight;  /* weight that is assigned to it at the end */
};

/* allocate size x size matrix S of type type and named T */
#define AllocMatrix(S, type, size, T) \
{ \
  int ii; \
  if ((S = (type**)malloc((size)*sizeof(type*))) == NULL) \
    nomemory(T); \
```

```
    for(ii=0;ii<(size);ii++) { \
        if ((S[ii] = (type*)calloc((size_t)size,sizeof(type))) == NULL) \
            nomemory(T); \
    } \
}

static int idcmp(int *p1, int *p2)        /* sorting integers in decreasing order */
{
    int tmp = *p1 - *p2;
    if (tmp <0)
        return -1;
    else if (tmp == 0)
        return 0;
    else
        return 1;
}

/* gets a line from a file but ignores it if it is a comment i.e. a line starting with # */
static char *Fgets(char* l, int len, FILE *f)
{
    do {
        if (fgets(l, len, f) == NULL)
            return NULL;
        else if (l[0] ≠ '#')
            return l;
    } while (l[0] == '#');
    return l;
}

static void MWarshall(int **A, int N)        /* computes transitive closure of a relations A */
{
    int k, i, j;

    for(k=0;k<N;k++)
        for(j=0;j<N;j++)
            for(i=0;i<N;i++)
                A[i][j] = Min(2, Max(A[i][j], A[i][k]*A[k][j]));
}

/* checks if x is in vector v of length n and returns 1 or 0 */
static int InVector(int x, int *v, int n)
{
    int found = 0, i=0;

    while (i<n && !found)
        found = (x == v[i++]);
    return found;
}

static void GetNonImportant(int *nonimp, int *n1, int N)
{
    char *t, Line[80];
    int l, nn, ff, j;

    printf("Do you have any ADDITIONAL nonimportant objectives?\n");
    printf("Enter them all in one line separated by %s.\n", SEP);
    printf("Empty line means all are important\n");
    printf("Enter non-important objectives:  ");

    if ((Fgets(Line, 79, stdin) == NULL) || (strlen(Line) == 1)) {
        return;
```

```
}
else {
   l = strlen(Line);
   if (Line[l-1] == '\n')
      Line[l-1]='\0';      /* discard new line character */
   t = strtok(Line, SEP);
   while ((*n1 <N) && (t ≠ NULL)) {
      nn = atoi(t)-1;
      if ((nn ≥ 0) && (nn <N)) {
         ff = 0;
         for(j=0;j<*n1 && ff==0;j++) {
            if (nonimp[j]==nn)      /* already there? */
               ff = 1;
         }
         if (ff == 0) {      /* not found yet */
            nonimp[*n1] = nn;
            (*n1)++;
         }
         t = strtok(NULL, SEP);      /* get the next one */
      }
   }
   /* sort in increasing order */
   qsort((void*)nonimp, (unsigned)(*n1), sizeof(int),
         (int (*)(const void*, const void*))(idcmp));
}

static void GetEquallyImportant(struct OBJECTIVE *O, int N)
{
   char Line[80], *t;
   int j, i, m, l, nn;
   int *tmpv;

   if ((tmpv = (int*)calloc((size_t)N, sizeof(int))) == NULL)
      nomemory("tmpv");

   printf("Please enter list of equally important objectives\n");
   printf("Enter a list per line separated with %s.  ", SEP);
   printf("Empty line ends.\n");
   printf("Please enter per line ALL equally important objectives\n");

   while ((Fgets(Line, 79, stdin) ≠ NULL) && (strlen(Line) ≠ 1)) {      /* no empty line yet */
      j = 0;
      l = strlen(Line);
      if (Line[l-1] == '\n')
         Line[l-1]='\0';      /* discard new line character */
      t = strtok(Line, SEP);
      while ((j <N) && (t ≠ NULL)) {
         nn = atoi(t)-1;
         if ((nn ≥ 0) && (nn <N))      /* insert it into the list of equivalent */
            tmpv[j++] = nn;
         t = strtok(NULL, SEP);
      }
      for(i=0;i<j;i++)
         for(m=0;m<j;m++)
            O[tmpv[i]].equ[tmpv[m]] = 1;
      for(i=0;i<N;i++)
         tmpv[i] = 0;
   }
}
```

```
static int AskPreference(int i1, int i2)
{
   int nn = 0;

   printf("Considering objectives %d and %d, is:\n", i1+1, i2+1);
   printf(" Enter 1 if %d is much more important then %d\n", i1+1, i2+1);
   printf(" Enter 2 if %d is more important then %d\n", i1+1, i2+1);
   printf(" Enter 3 if %d is less important then %d\n", i1+1, i2+1);
   printf(" Enter 4 if %d is much less important then %d\n", i1+1, i2+1);

   while (nn <1 || nn > NN_MAX) {
      printf("\n Please enter number between 1 and %d --> ", NN_MAX);
      scanf("%d", &nn);
   }
   return nn;
}

static void SetPreference(int i1, int i2, int p, double **P, int **Pa)
{
   switch (p) {
   case 1:      /* much more important */
      P[i1][i2] = MMI;
      P[i2][i1] = MLI;
      Pa[i1][i2] = RMMI;
      break;
   case 2:      /* more important */
      P[i1][i2] = MI;
      P[i2][i1] = LI;
      Pa[i1][i2] = RMI;
      break;
   case 3:      /* less important */
      P[i1][i2] = LI;
      P[i2][i1] = MI;
      Pa[i2][i1] = RMI;
      break;
   case 4:      /* much less important */
      P[i1][i2] = MLI;
      P[i2][i1] = MMI;
      Pa[i2][i1] = RMMI;
      break;
   case 5:      /* not comparable */
      P[i1][i2] = NCMP;
      P[i2][i1] = NCMP;
      Pa[i1][i2] = Pa[i2][i1] = 1;
      break;
   }
}

/* ComputeWeights() asks questions about pairwise preferences and compute
    weights etc accordingly. */
int ComputeWeights(int nobj, double *PWeights)
{
   int i, j, nn, k, N, no, n, found, Nc;
   double **R, ts;
   int **Ra, *C;      /* adj. matrix for R, 1 if in relationship, 0 if not */
   int *obj, *imp, *nonimp;
   struct OBJECTIVE *o;

   N = nobj;

   if ((o = (struct OBJECTIVE*)malloc(N*sizeof(struct OBJECTIVE))) == NULL)
```

```
    nomemory("OBJECTIVE");

for(i=0;i<N;i++) {
    o[i].weight = -1;
    if ((o[i].equ = (int*)calloc((size_t)N, sizeof(int))) == NULL)
        nomemory("o");
    o[i].equ[i] = 1;        /* it is equally important as itself */
}

if ((obj = (int*)malloc(N*sizeof(int))) == NULL)
    nomemory("obj");
else {
    for(i=0;i<N;i++)
        obj[i] = i;
}

if ((nonimp = (int*)calloc((size_t)N, sizeof(int))) == NULL)
    nomemory("nonimp");
if ((imp = (int*)calloc((size_t)N, sizeof(int))) == NULL)
    nomemory("imp");

j=0;
printf("Objectives are numbered from 1 to %d\n", N);
printf("Nonimportant objectives are (according to the mask):  ");
for(i=0;i<N;i++)
    if (Use[i]==0) {
        printf("%d ", i+1);
        nonimp[j++]=i;
    }
printf("\n");

no = j;
GetNonImportant(nonimp, &no, N);
n = N-no;       /* n is the number of important objectives, no of nonimportant */


/* assign weights of nonimportant objectives to 0 */
for(i=0;i<no;i++)
    o[nonimp[i]].weight = 0;
j = 0;

/* OK, find the important elements */
for(i=0;i<N;i++) {
    if (!InVector(i, nonimp, no))
        imp[j++] = i;
}

printf(" Important objectives are:  ");
for(i=0;i<n;i++)
    printf("%d%s", imp[i]+1, SEP);

printf("\n Nonimportant objectives are:  ");
for(i=0;i<no;i++)
    printf("%d%s", nonimp[i]+1, SEP);
printf("\n");

GetEquallyImportant(o, N);

/* OK, now we have important objectives in imp[] vector */
/* Put elements in classes, Nc will be number of classes */
if ((C = (int*)calloc((size_t)n, sizeof(int))) == NULL)
```

```
   nomemory("C");

k=0;
for(i=0;i<n;i++) {       /* n is the number of important elements */
   found = 0;
   for(j=0;j<i && !found;j++)
      found = (o[imp[i]].equ[imp[j]] == 1);
   if (!found)
      C[k++] = imp[i];
}
Nc = k;

if (Nc == 1) {       /* there is only one class, not very exciting */
   o[C[0]].weight = 1;
}
else {       /* ask for preferences, compute weights etc */
   /* allocate matrices R and Ra of size Nc*Nc */
   AllocMatrix(R, double, Nc, "R");
   AllocMatrix(Ra, int, Nc, "Ra");

   /* asign them to unity matrix */
   for(i=0;i<Nc;i++) {
      for(j=0;j<Nc;j++) {
         R[i][j] = 0;
         Ra[i][j] = 0;
      }
      R[i][i]=1;
      Ra[i][i] = 1;
   }

   /* ok, for everything else, ask a question */
   printf("Now you have to answer some questions:\n");
   for(i=0;i<Nc;i++) {
      for(j=0;j<Nc;j++) {
         if (Ra[i][j] + Ra[j][i] == 0) {
            nn = AskPreference(C[i], C[j]);
            if (nn==0) {
               fprintf(stderr, "Wrong preference %d between %d=C[%d] and %d=C[%d]!\n",
                        nn, C[i], i, C[j], j);
               exit(1);
            }
            SetPreference(i, j, nn, R, Ra);
            MWarshall(Ra, Nc);
         }
      }
   }

   /* Compute matrix R */
   for(i=0;i<Nc;i++)
   for(j=0;j<Nc;j++) {
      if (i==j)
         R[i][j] = EI;
      else if ((Ra[i][j] == 0) && (Ra[j][i] == RMI)) {       /* j > i */
         R[i][j] = LI; R[j][i] = MI;
      }
      else if ((Ra[i][j] == 0) && (Ra[j][i] == RMMI)) {       /* j >> i */
         R[i][j] = MLI; R[j][i] = MMI;
      }
      else if ((Ra[i][j] == RMI) && (Ra[j][i] == 0)) {       /* j < i */
         R[i][j] = MI; R[j][i] = LI;
      }
```

```
      else if ((Ra[i][j] == RMMI) && (Ra[j][i] == 0)) {      /* j << i */
         R[i][j] = MMI; R[j][i] = MLI;
      }
      else {
         fprintf(stderr, "Wrong pair Ra[%d][%d]=%d, Ra[%d][%d]=%d\n",
                  i, j, Ra[i][j], j, i, Ra[j][i]);
      }
   }

/* Compute weights of the classes */
ts = 0;
for(i=0;i<Nc;i++) {
   o[C[i]].score = 0;
   for(j=0;j<Nc;j++)
      if (j≠i)
         o[C[i]].score += R[i][j];
   ts += o[C[i]].score;
   }
   for(i=0;i<Nc;i++)
      o[C[i]].weight = o[C[i]].score/ts;
}  /* end of the long else part */

/* now assign the same values to all other in the same class */
for(i=0;i<Nc;i++)
   for(j=0;j<N;j++)
      if (o[C[i]].equ[j] == 1)
         o[j].weight = o[C[i]].weight;

/* normalise and output weights */
ts = 0;
for(i=0;i<N;i++)
   ts += o[i].weight;
   for(i=0;i<N;i++) {
      o[i].weight /= ts;
      PWeights[i] = o[i].weight;
   }
   return 0;
}
```

APPENDIX C

Scenario examples

Here are some further examples of scenarios targeted towards certain aircraft types. All data is from (Sharpe 1999, Chandler 1999):

```
# constraints for B-2 Spirit Stealth Bomber
# file expr.b2
y11 <=52.43    # Wing span
x4 <=464.5     # Wing plan area
y9 >=8167      # Ferry range
y10 >= 170550 # take-off mass
x3 >= 0.8      # cruising speed 980km/h
```

```
# constraints for SR-71 Blackbird Recon
#file expr.blackbird
y11<=16.94    # Wing span
x4 <=149.10   # Wing plan area
y9 >=5230     # Ferry range
y10 >= 78017 # take-off mass
x3 >= 3.35    # max cruise speed 4100km/h
y1 <= 1646    # take-off run
```

```
# constraints for F-111 Aardvark Bomber
# file expr.f111
y11 >= 9.74  & y11<=19.20 # Wing span
x4 <=61.07 & x4 >=48.77   # Wing plan area
y9 >=4707                 # Ferry range
y10 >= 45360              # take-off mass
x3 >= 0.75                # max cruise speed 919km/h
y1 <= 951                 # take-off run
```

```
# constraints for F-117A Stealth Fighter
# file expr.f117
y11 <=13.20   # Wing span
x4 <=105.9    # Wing plan area
y9 >=2327     # Ferry range estimated as combat radius * 2.7
y10 >= 24894 # take-off mass
y1 <= 1890    # take-off run
```

```
# constraints for McDonnell Douglas F-15J Eagle
# file expr.f15j
y11 <= 13.05 # Wing span
x4 <=56.48   # Wing plan area
y9 >= 5745   # Ferry range
y10 >=30844  # take-off mass
```

```
# constraints for General Dynamics F-16A
# file expr.f16a
y11 <= 9.45  # Wing span
x4 <=27.87   # Wing plan area
y9 >= 925    # Ferry range
y10 >= 16057 # take-off mass
```

```
# constraints for Mikoyan-Gurevich MiG-29M
# file expr.mig29m
y11 <= 11.36 # Wing span
x4 <=35.2    # Wing plan area
y9 >= 1500   # Ferry range
y10 >=18500  # take-off mass
```

```
# constraints for Dassault Mirage 2000H
# file expr.mirage2000h
y11 <= 9.2   # Wing span
x4 <=42      # Wing plan area
y9 >= 1480   # Ferry range
y10 >=17000 # take-off mass
```

```
# constraints for Grumman F-14D Tomcat
# file expr.tomcat
y11 >= 11.65  & y11 <= 19.55 # Wing span
x4 <=52.49                   # Wing plan area
y9 >=3200                    # Ferry range
y10 >= 33724                 # take-off mass
x3 >= 0.83                   # max cruise speed 1019km/h
y1 <= 427                    # take-off run
```

```
# constraints for Tornado GR.Mk1  Bomber
# file expr.tornado
y11 >= 8.60  & y11 <= 13.91 # Wing span
x4 <=26.60                  # Wing plan area
y9 >=3890                   # Ferry range
y10 >= 27951                # take-off mass
y1 <= 900                   # take-off run
```

```
# constraints for Eurofighter ED-2000 Typhoon
# file expr.typhoon
y11 <= 10.5  # Wing span
x4 <=53      # Wing plan area
y9 >= 600    # Ferry range
y10 >= 21000 # take-off mass
```