MULTI-OBJECTIVE OPTIMISATION APPLIED TO INDUSTRIAL ENERGY PROBLEMS

THÈSE N° XXXX (2002)

PRÉSENTÉE AU DÉPARTEMENT DE GÉNIE MÉCANIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DE GRADE DE DOCTEUR ÈS SCIENCES TECHNIQUES

PAR

Geoff LEYLAND

BE (Engineering Science), ME (Mechanical Engineering)

de nationalité néo-zélandais

acceptée sur proposition du jury:

Prof. D. Favrat, directeur de thèse
Prof. A. Germond, rapporteur
Prof. S. Kraines, rapporteur
Prof. L. Thiele, rapporteur
Prof. D. Wallace, rapporteur

Lausanne, EPFL
2002

Abstract

This thesis presents the development of a new multi-objective optimisation tool and applies it to a number of industrial problems related to optimising energy systems.

Multi-objective optimisation techniques provide the information needed for detailed analyses of design trade-offs between conflicting objectives. For example, if a product must be both inexpensive and high quality, the multi-objective optimiser will provide a range of optimal options from the cheapest (but lowest quality) alternative to the highest quality (but most expensive), and a range of designs in between—those that are the most interesting to the decision-maker.

The optimisation tool developed is the queueing multi-objective optimiser (QMOO), an evolutionary algorithm (EA). EAs are particularly suited to multi-objective optimisation because they work with a population of potential solutions, each representing a different trade-off between objectives. EAs are ideal to energy system optimisation because problems from that domain are often non-linear, discontinuous, disjoint, and multi-modal. These features make energy system optimisation problems difficult to resolve with other optimisation techniques.

QMOO has several features that improve its performance on energy systems problems—features that are applicable to a wide range of optimisation problems. QMOO uses cluster analysis techniques to identify separate local optima simultaneously. This technique preserves diversity and helps convergence to difficult-to-find optima. Once normal dominance relations no longer discriminate sufficiently between population members certain individuals are chosen and removed from the population. Careful choice of the individuals to be removed ensures that convergence continues throughout the optimisation. Preserving of the 'tail regions' of the population helps the algorithm to explore the full extent of the problem's optimal regions.

QMOO is applied to a number of problems: coke factory placement in Shanxi Province, China; choice of heat recovery system operating temperatures; design of heat-exchanger networks; hybrid vehicle configuration; district heating network design, and others. Several of the problems were optimised previously using single-objective EAs. QMOO proved capable of finding entire ranges of solutions faster than the earlier methods found a single solution. In most cases, QMOO successfully optimises the problems without requiring any specific tuning to each problem.

QMOO is also tested on a number of test problems found in the literature. QMOO's techniques for improving convergence prove effective on these problems, and its non-tuned performance is excellent compared to other algorithms found in the literature.

Résumé

Cette thèse traite du développement d'un outil d'optimisation multi-objectifs et de ses applications à divers problèmes industriels dans le domaine des systèmes énergétiques.
L'optimisation multi-objectifs permet d'apporter l'information nécessaire à l'analyse détaillée des compromis de conception entre des objectifs en conflit. Par exemple, si on désire qu'un produit soit bon marché et de bonne qualité, les résultats d'une optimisation multi-objectifs vont fournir une plage de solutions optimales entre le meilleur marché (mais de moindre qualité) et celle de meilleure qualité (mais plus cher), et plusieurs autre solutions entre ces deux extrêmes—qui seront les plus intéressantes pour le décideur.
L'outil d'optimisation développé, le 'queueing multi-objective optimiser' (QMOO), est un algorithme évolutif (EA). Les EAs sont particulièrement indiqués pour l'optimisation multi-objectifs, car ils travaillent avec une population de solutions potentielles, chacun de ses membres représentant un compromis entre les objectifs. Les EAs sont aussi indiqués pour les problèmes d'optimisation des systèmes énergétiques, car ces problèmes sont souvent non-linéaires, discontinus, disjoints et multimodaux. Ces caractéristiques rendent de tels problèmes laborieux à résoudre avec des méthodes d'optimisation conventionnelles.
Dans ce travail, plusieurs fonctionnalités ont été implémentées dans QMOO afin de d'améliorer ses performances lors de l'optimisation des systèmes énergétiques. QMOO utilise les techniques d'analyse de grappes permettant d'identifier les differents optima locaux. Ceci permet de maintenir la diversité de la population, et d'améliorer la convergence vers les optima difficiles à trouver. La convergence est assurée tout au long de l'optimisation par un choix soigneux des individus à supprimer de la population lorsque les relations de dominance entre les individus n'arrivent plus à les discriminer. La préservation des régions limitrophes de la population permet à l'algorithme d'explorer l'ensemble des zones optimales du problème.
QMOO est appliqué à divers problèmes d'optimisation réels: le localisation optimale des fabriques de coke dans le Province de Shanxi en Chine; le choix des niveaux des températures pour l'analyse énergétique d'un procédé; la conception des réseaux d'échangeurs de chaleur et la conception de systèmes de chauffage urbain. Certains de ces problèmes avaient été résolus auparavant par des méthodes d'optimisation mono-objectif. QMOO a été capable de trouver des plages entières de solutions en moins de temps que les méthodes utilisées précédemment. En trouvant en plus plusieurs solutions alors que précédemment une seule solution était trouvée. Dans la plupart des cas, QMOO a résolu les problèmes sans préparation spéciale.
QMOO a aussi été testé sur divers problèmes de la littérature. Ces essais montrent que les techniques développées pour améliorer la convergence de l'algorithme sur des problèmes de systèmes énergétiques sont aussi utiles sur les problèmes tests, et que la performance de QMOO sans préparation particulière est excellente comparée aux algorithmes présentés dans la littérature.

Acknowledgements

First thank in this thesis must go to Geoff Bates, who, many years ago now, showed me round the EPFL, and introduced me to LENI. Some years later, again, thanks to Geoff, I found myself in Switzerland all alone, unable to speak much more French than 'Je m'appelle Geoff Leyland'.

For the thesis, of course, many thanks are due to Daniel Favrat, who has managed to follow the sometimes twisted course of this thesis, and kept me on track—as well as all the other things for which I am grateful—giving me the opportunity to work here, funding the work, letting me study all the interesting parts of the project, and correcting the drafts.

Thanks for the funding are also due to the Alliance for Global Sustainability, a collaboration between MIT the University of Tokyo and the Swiss Technical institutes, which funded this work as part of the 'Holistic Design' project, and through which I met many interesting people, including Steven Kraines and David Wallace.

A number of other people have made work on the thesis much easier, and more enjoyable. Thanks to Adam Molyneaux for being, besides a great friend, snowboarding companion and juggling partner, for working with me on some of the projects, and always having something useful to say about optimisation. To Steven Kraines, again for being a friend, but also introducing QMOO to a whole new problem domain (and for showing me how good Japanese food can be!), and to David Wallace for always being interested in the work, being a source of inspiration, and for the tour of Tokyo. Finally, (because he got here last) I have to thank Francois Marechal for his help with the last parts of the thesis, and for seeming to believe in my work even more than I do.

The people working at LENI deserve a lot of thanks, first for making me learn French—Roger Röthlisberger, Claude-Alain Paschoud, Sébastien Germano, Robert Zanelli (we won't forget you), Stefan Pelster and Benoit Olsommer, then for putting up with me looking after the computers and finishing my thesis, and most importantly, I guess, for snowboarding and skiing with me—Michele Zehnder, Xavier Pelet and Diego Larrain, and running with (well, far in front of) me, Marc Salle.

Being so far from 'home' doesn't make one's family any more distant, even after five years I miss them more every day, or any less important, I would never have finished this without them—some of them have even read (and corrected) this. Thanks Mum, Dad, Maury, Roger, Max and Oli. Having the family so far away does mean that you grow a second family where you live. I'm not going to name you all, you know who you are. Thanks.

Contents

List of Figures

List of Tables

Abbreviations and Symbols

Abbreviations

EA	Evolutionary Algorithm
EOC	Evolutionary Operator Choice
EP	Evolutionary Programming
ES	Evolution Strategies
FCM	Fuzzy C-Means Clustering
GA	Genetic Algorithm
MOEA	Multi-Objective EA
MOO	Multi-Objective Optimiser
MOP	Multi-Objective Problem
NDS	Non-Dominated Set
PBA	Population-Based Algorithm
POF	Pareto-Optimal Front
SOEA	Single Objective EA

Multi-Objective Evolutionary Algorithms

NPGA	Niched Pareto Genetic Algorithm
NSGA(-II)	Non-dominated Sorting Genetic Algorithm
MOGA	Multi-Objective Genetic Algorithm
MOSGA	Multi-Objective Struggle Genetic Algorithm
PAES	Pareto-Archived Evolutionary Strategy
QMOO	Queueing Multi-Objective Optimiser
SGA	Struggle Genetic Algorithm
SPEA(-II)	Strength Pareto Evolutionary Algorithm
VEGA	Vector Evaluated Genetic Algorithm
SPEA(-II)	Strength Pareto Evolutionary Algorithm

Symbols

Problem Definition

$\mathcal{S}$	Search space
n	Number of decision variables
$\mathcal{F}$	Objective space
m	Number of objectives
$\boldsymbol{f}(\boldsymbol{x})$	Objective function $\boldsymbol{f} : \mathcal{S} \rightarrow \mathcal{F} = \boldsymbol{f}(\boldsymbol{x} \in \mathcal{S})$
$g(\boldsymbol{x})$	Constraint functions
q	Number of constraints
$F(\boldsymbol{x})$	Single-objective objective function
$\boldsymbol{w}$	Weighting factors for forming a weighted sum

Populations

p	Population size
$\boldsymbol{x}, \boldsymbol{y}$	Decision variable values of population members
$\boldsymbol{u}, \boldsymbol{v}$	Objective function values of population members
i, j, k	Indices for decision variables, objectives, and generations
a, b	Population member indices
I	An individual in the population
$\boldsymbol{\mu}$	Vector or random values

Grouping and Ranking

N_g	Number of groups
G_{min}	Minimum group size
G_{max}	Maximum group size
r_{min}	Minimum rank
r_{max}	Maximum rank
N_q	Number of points in a quadratic interpolation for thinning

Evolution Strategies

σ	variance of mutation
$\boldsymbol{\alpha}$	rotation angles of mutation
μ	population size
λ	number of children
κ	the maximum lifespan of an individual, in generations
ρ	number of parents involved in combination

Chapter 1

Introduction

1.1 Introduction

This thesis concerns the development of a multi-objective evolutionary algorithm and its application to a range of optimisation problems arising from the Laboratoire d'Energétique Industrielle's (LENI) projects and from collaborations with other institutes. These problems involve optimisation of energy systems, with respect to energetic, environmental, and economic objectives. Such optimisations allow the decision-maker to consider a wide range of criteria, and assess design choices in a holistic sense—normally a difficult task when systems are complex.

The algorithm is discussed in terms of energy system optimisation, and features are developed to enhance performance when solving this kind of problem. However, the algorithm can be applied to a wider range of problems—in this thesis one purely economic problem is treated—and the features of the algorithm that make it appropriate for energy system models apply to many problem domains.

The competitive advantage of the algorithm developed here is its ability to find a wide range of interesting solutions to an optimisation problem, in contrast to earlier methods that return only one, or very few, hopefully optimal solutions. The range of solutions found gives the decision-maker far more information about the model than conventional methods, which aids considerably in making final design choices.

Comparative tests on real problems that have been optimised by other methods show the algorithm developed here can find sets of points describing the optimal region in *one tenth* the time that the methods used earlier took to find a single, less optimal point. On a set of multi-objective test problems, the untuned algorithm proves to be, at least as good as other methods found in recent literature, and at best up to three times faster.

1.2 Motivation

The development of the algorithm is primarily motivated by the need to optimise energy system problems. Energy system problems present particular difficulties for optimisation algorithms, and earlier work has shown that they are best solved by evolutionary algorithms (EAs) [24, 96, 102].

1.2.1 Energy Systems and Optimisation

‘Optimising’ an energy system is a complex problem, and difficulties begin well before the optimisation stage is reached, at the stage of model definition. The question of what is to be considered as optimal must be addressed very early on. For example, based on economic criteria, one may wish to construct the least expensive system which will perform the task required, or it may be more important to construct a system with low operation and maintenance costs. On the other hand, if the system in question is a *real* energy system, the optimal system may be defined as that with the best first or second law efficiency. If energy costs are a significant part of operating costs a thermally efficient system might also have low operating costs, but it probably will not be the cheapest to construct, and quite likely, the advanced machinery will require considerable maintenance. In other cases, product quality is most important—so the objective is to construct a plant that produces the highest quality product. Finally, one may wish to optimise environmental criteria, and minimise emissions or negative environmental impact of the plant.

Even if all the options for defining an optimum are considered early in the design of a system, it is unlikely that any one criteria will emerge that defines the best possible system. Instead there will be a variety of acceptable trade-offs between objectives—it may be acceptable to pay a little more for a cleaner plant or better quality. However, within the bounds of these trade-offs and constraints there will be a range of possible plant designs, each with its strong and weak points. The final decisions will probably be influenced by qualitative aspects of the plant’s design, guided by the results of optimisation.

A ‘naive’ view of an optimisation algorithm is a process which, given a system model, some constraints, and a single objective, will return a single point in the search space representing the best possible system design. However, in most cases the process will include not only the initial solving of the model, but also parameter studies. In these parameter studies, constraints and components of the objective are varied, in order to see how the optimal solution changes with respect to these parameters. However, even with parameter studies, a single-objective optimisation does not provide a sufficiently global view of the system.

For a decision maker the ideal ‘result’ of an optimisation would be a characterisation of all of the *interesting* regions of the model. Each region would represent a separate design choice, with different performance in each of several objectives, and would contain a number of solutions, each representing a particular trade-off between those objectives.

Harik [61] summarises this view very succinctly:

> There is, in the realm of optimization, quite often the need to identify several good solutions to a problem, as opposed to one optimal solution. This need arises from several sources. Often an optimization problem is a simplification of a real world problem that in part requires human or inquantifiable judgement. In these types of problems, an optimizer needs to suggest various possible alternatives that can later be judged by a human expert. In other situations, a better understanding of a search space is desired in terms of the location of its various optima.

1.2.2 Energy System Models

Energy system models have a number of characteristics which make them difficult to optimise and require the broader definition of optimisation presented above.

From the point of view of an optimisation algorithm, energy system models must often be treated as 'black boxes'—a set of parameters are presented as inputs, and the black box gives a number of outputs. Derivatives of the outputs with respect to the inputs, and any extra information about the form of the model, are usually not available. Many of the components of the system models used here are programmed in procedural languages, and cannot always be easily modified, and which sometimes crash unexpectedly. This problem may well be a passing one—advanced modelling software now makes it possible to construct models that are more transparent for analysis and optimisation. However, many more problems with optimising energy systems relate to the form of those problems, and not to the way in which they are described.

Because energy system models are highly non-linear relatively robust optimisation tools must be used to optimise them. Furthermore, they often have some integer parameters (such as the number of a certain component, or the size of a component that is only available in a fixed set of sizes), and even combinatorial parameters (such as choosing which members of a set of possible components should be used). The model then becomes not only non-continuous (the optimal surface does not have smooth derivatives), but discontinuous (the optimal surface has jumps in it) and disjoint (the feasible region has holes in it, or there are several completely different feasible regions, each of which has its own local optimum or optima, with no easy route for passing from one region to another.).

Such problems are difficult to optimise by conventional methods, which are based on assumptions of some continuity in the optimal surface. When, in addition to these features, the decision maker would like information about all the interesting feasible regions, and about the trade-offs between objectives, conventional optimisation methods start to get overstretched.

These difficulties were the driver for the development of the algorithm presented in this work.

1.2.3 The Role of optimisation

Energy system models are often inexact, and occasionally wrong. This puts into question the value of the results of an optimisation, and poses questions about what the role of optimisation should be. Clearly, unless the fidelity of a model with the real system it attempts to simulate is nearly perfect, the results of an optimisation of that model are of limited value. Ideally, the optimiser should give a characterisation of the interesting regions of the model, so that these regions can be investigated on the real system.

System models are not inexact just because the real system is not fully understood. Often the system modeller will make reasonable assumptions about the system in order to reduce computational time, or because not all the data about a subsystem is available. Often this is entirely acceptable, but in some cases this will result in one system configuration appearing better or worse than a more accurate model would reveal it to be. In these cases the optimiser should identify the erroneous region, as well as any others of interest, and the modeller or decision-maker, seeing that their assumptions are critical to the form of the model, will choose to spend time increasing the accuracy of the model, or collecting more data from the real system.

Errors in the model also pose problems. Sometimes (it has certainly been the experience of this author), errors will make some very unusual design choices seem to perform extremely well. In these cases, the optimisation algorithm identifies errors by finding the overperforming solutions, and can aid debugging the model by providing as much information about the region as possible.

Thus the traditional view of optimisation as a process of 'model definition → model implementation → optimisation → implementation of optimal solution' chain, is replaced by a more realistic scheme where optimisation plays the continual role of characterising the model while it is being developed, and after it has been proved reasonably correct.

In addition to aiding in the development and debugging of a model, optimisation can also play a large part in analysing and understanding a model's behaviour. The optimisation process rarely finishes when the optimisation algorithm gives its results. The decision-maker or engineer will then examine the results in order to understand the solution the optimisation algorithm proposes, and why it has made such a choice.

In most cases optimal designs will have an underlying logic that can be understood in engineering terms. If a satisfactory explanation for the choice of optimum is not found, it is usually an indication there is an error in the model or optimisation, or that the model is not fully understood. Certainly, engineers and decision-makers are often wary of system configurations that do not have a rational explanation.

To summarise: optimisation is not only a method for finding the optimal points for a model. Careful examination of the results can aid in debugging the model, or in understanding the behaviour of the model. If there does not appear to be a common sense explanation for the choices made in an optimal system, there may well be an error in the optimisation. Often the common sense in question is not evident before the model has been optimised, but in studying the results, particularly for a difficult system, the engineer or decision-maker will either find errors in the model, or gain new insight into the system's behaviour.

Thus, in the opinion of this author, optimisation algorithms should concentrate on providing as much information as possible about all interesting regions of a model, rather than solely on finding an exact single optimum of an inexact system model. Optimisation thus becomes an integral part of the set of engineering tools used to develop energy system models and analyse their behaviour.

1.3 Earlier Work at LENI

Before this work was commenced, LENI had already amassed considerable experience in the domain of energy system modelling. Notably three theses and subsequent publications, cover the modelling and optimisation by EA of three thermal systems—a district heating system [23, 24], a waste incineration plant [95, 97, 98, 96], and a gas turbine co-generation system with CO_2 capture [101, 102].

In this earlier work, optimisation of the system models by non-EA methods was investigated, without success. After some time, all three authors began using the Struggle Genetic Algorithm (SGA) [60, 120], an EA developed at MIT, to optimise their respective problems. The SGA proved an extremely robust optimiser, more than capable of tackling each of the problems. In all cases, insights gained from the optimisation lead to considerable improvement of the systems under consideration.

The SGA is a single objective optimiser (though it was later extended to multi-objective problems by Andersson and Wallace [3]), but the projects all intended to take into account both the costs of their respective systems, and their environmental impacts. For this purpose, the 'environomic' approach was developed. Here, environmental impacts are taken into account in the optimisation by estimating their costs, and adding these costs to the economic costs and benefits of the system. Thus the 'total cost' of the system is minimised. The weakness of the environomic approach lies in uncertainties in the *estimates* of the costs of the environmental impacts. Estimates of the cost of, for example, a kilogram of CO_2 emitted vary wildly, as they do for many other pollutants. For the environomic approach, these costs must be specified before the optimisation begins, and the results are only appropriate for that cost. Investigating a range of costs for pollutants involves re-running the optimisation with a number of different values of the cost of pollutants. Given that the optimisations in question could take up to a week to solve, investigating a wide range of values for pollution costs is impractical.

The multi-objective optimisation (MOO) approach developed here means that the two objectives can be kept separate and instead of a single optimal point, a set of optimal trade-offs, ranging from, for example, the cheapest but most polluting solution, to the cleanest but most expensive are returned. Improvements to the optimisation algorithm itself mean that the new algorithm can find such a set up to ten times faster than the earlier methods could find a single optimal point.

1.4 About this Thesis

1.4.1 Chapter Two: Optimisation by Evolutionary Algorithms

This chapter introduces of optimisation by EAs, and the terminology of the field. The history of the development of EAs is reviewed and proofs of convergence for EAs, and their requirements are presented. Details of specific methods used in well-known EAs are discussed.

After a presentation of the development of the three main 'families' of EAs, the development of EAs is presented from a more modern viewpoint. This starts with the concept of a general 'population-based algorithm', and presents a proof of convergence and its requirements for such an algorithm. Next, it is shown that such an optimisation is essentially a problem of estimating the distribution of a set of points, and then sampling from that distribution. A cheap and easy way of estimating the distribution of a set of points and sampling from that distribution in one step, is by using the kinds of 'operators' developed by EAs. Furthermore, it is shown that these operators lend themselves easily to estimations of distributions that prefer 'better points', even in cases where it is not clear what better means.

1.4.2 Chapter Three: Multi-Objective Evolutionary Algorithms

Chapter three introduces concepts specific to multi-objective optimisation (MOO), and the application of EAs to these problems. First, motivations for MOO are discussed, and early methods of reducing multi-objective problems (MOPs) to single objective problems, along with their drawbacks, are presented. Next the definition of Pareto-optimality, which formalises the concept of 'better' in a multi-objective sense, is presented, along with the concept of a Pareto-optimal frontier, and the related concepts of domination and non-dominated sets. This is sufficient background to understand the basic concepts of multi-objective optimisation. The history of multi-objective EAs (MOEAs) is then presented, first describing early algorithms that did not use Pareto-optimality, and then on to modern, elitist, Pareto-optimal algorithms. Finally, aspects of EAs specific to multi-objective problems (ranking and management of non-dominated sets) are presented, and approaches found in the literature are reviewed.

1.4.3 Chapter Four: The Queueing Multi-Objective Optimiser

This chapter presents QMOO, the 'Queueing Multi-Objective Optimiser' developed in this work. QMOO has a number of unique features and innovations, both in its algorithmic design and its implementations of the 'building blocks' of MOEAs. These make it uniquely suited to optimisation by MOEAs, easy to parallelise, ensure that it finds a number of local optima, and give it rapid and robust convergence.

QMOO is a steady-state, 'queue-based' EA, in which one can either view potential solutions to an optimisation problem as entirely independent entities, or where all the processes that must be applied to these solutions run in parallel, and continuously. This architecture makes QMOO extremely flexible, allowing processes to be added to the algorithm transparently, whether they are processes that work best on a single solution at a time, or whether they work most efficiently in batches, or must be applied to the entire population. It also allows for a simple and efficient (and apparently novel) method of parallelism to be applied to the evaluation of objective function values.

Another notable feature of QMOO is its method of diversity preservation, and of searching for multiple local optima. The population is divided into independent groups in decision variable space, and these groups are allowed to evolve relatively independently of each other, with some interbreeding between the groups, and no competition between them. This means QMOO can find several local optima. The groups also provide valuable sources of 'diversity', which help to prevent 'premature convergence' to non-optimal solutions.

Two measures are taken to improve the robustness of convergence over a wide range of problems. The first, evolutionary operator choice (EOC), also proves to improve convergence speed over a wide range of (though not all) test problems. The second, the preservation of the 'tails' of a non-dominated set, greatly improves convergence robustness on a number of particularly difficult problems, without a large detrimental effect on other test cases. Both methods prove their worth on 'real' problems. However, the long solution times of such problems make statistical data impractical to generate.

1.4.4 Chapter Five: Test Problems and Results

The choices made in the design of QMOO have been tested on problems in order to see what effects they have on performance, and under what conditions. The design choices tested were made in response to observations of the algorithm's behaviour on a number of real problems, and have improved convergence on these problems. However most of the real problems tackled by QMOO take too long to solve for it to be practical to perform statistical studies of these effects, so the effects of the choices are studied on test problems.

This chapter introduces a number of such test problems, discusses the issues that they present, and illustrates how an EA converges to their optima. It was originally intended that no new test problems would be introduced in this work, as there are already many available in the literature. However one of the test problems used had idiosyncrasies that were easily exploited, and modifications were made to its form, in order to make it more realistic.

A number of measures exist for quantifying the convergence performance of multi-objective EAs where the question of performance measurement is not as simple as for single-objective EAs. In a single-objective single-modal EA the entire population (hopefully) converges to a single point—the global optima—and an easy measure of performance is the lowest objective function value found so far. This can be extended over a number of test runs to provide average convergence curves, and measures of convergence robustness. Further, if the true optimal solution is known, the results can be normalised to allow comparison over a number of test problems. In multi-objective optimisation, however, the population must converge to a region, with each member of the population 'covering' a different part of that region, and convergence cannot be measured so simply. The convergence measure used here, normalised non-dominated volume, is introduced.

Results from the test problems are presented, demonstrating the effects of choices made in the design of QMOO. In general the techniques improve performance, though not always over the full range of problems, Studying the performance of QMOO leads to insights into how the algorithm is working, and into the behaviour of the test problems. The performance of QMOO is also briefly compared to other MOEAs and found to be more than adequate.

1.4.5 Chapter Six: Applications

QMOO has been used on a large number of 'real-world' problems. These include problems studied as part of this work, problems studied by other researchers at LENI, and problems from other Institutes.

The resolution of these problems has been the driving force behind the development of QMOO—most of the new techniques presented here were first developed to resolve an issue with a real-world problem, and later proved on test problems.

The adoption of QMOO for nearly all the optimisation performed at LENI, as well as the interest of researchers from other departments and universities, illustrates QMOO's utility in the domain of energy systems analysis, as well as in many other domains.

Two of the problems presented here had already been optimised using single objective methods. QMOO was used on these problems to see what advantages multi-objective optimisation would bring, and to see how fast QMOO converged compared to earlier methods. The results are excellent.

The problems presented are the following:

- **Coke Production in Shanxi Province, China.** The 'Shanxi Problem' is about optimising the coke (a coal derivative, not the drink) industry in China. Currently in Shanxi province, there are a large number of small plants that produce coke from coal. The small plants have the advantage that they can be near either or both the coal mines and coke consumers, but the disadvantage that they have high per-unit production costs, and are polluting. The Provincial government would like to replace the network of small plants with fewer large plants. The large plants will be cleaner and have lower per-unit production costs, but transport costs will be higher. The optimisation consists of finding plant sitings that minimise plant construction costs and transport costs, and then investigating the trade-offs between the two objectives. Once more data on pollution from coke production and transport becomes available, the optimisation will be performed with minimising emissions and costs as objectives.

  The Shanxi problem has a combinatorial nature, and it was for this problem that one of the unique features of QMOO, 'evolutionary operator choice' (EOC) was developed. This technique was originally developed so that a 'problem specific' operator could be used occasionally throughout the optimisation process. When this proved successful, EOC was generalised to be applicable to a wider range of problems.

- **Characterising Load Regimes for Plant Simulation** Many plant processes can be modelled with a 'discrete time' simulation: dynamic effects are ignored and the operation of a plant is simulated under a number of fixed, quasi-steady-state load or demand conditions. Often, a new plant or process will be simulated using historical data for operating conditions, a simulation being run for each data point. If there is a large amount of historical data, such as data collected hourly for a year, or number of years, the time to execute a full simulation can be extremely long.

  Often, the operating conditions fall into just a few 'modes', such as summer and winter or morning and evening demands on a heat and power system. However, finding the best modes in a mass of data can be difficult. Statistical methods exist for identifying patterns in data, but tend to be inflexible in how they characterise the groups.

  QMOO is used to identify such typical operating points in historical data. Because of the flexibility of the objective functions, its results are better than those produced by other methods.

- **Optimal Cuts on Grand Composite Curves** Most industrial processes will have a variety of heat and power demands and sources. These demands and supplies can be characterised by a grand composite curve which shows the energy requirements as a function of temperature ('pinch technology'). In order to minimise a process's waste energy, heat (and cold) can be re-used between sub-processes. For example, the waste heat from a burner can be used in a process requiring lower grade heat, such as a pre-heating or drying stage. Equally, sources of cold can be used for cooling in other parts of the plant.

The transfer of heat energy around the plant necessarily requires some equipment to bridge between temperature levels. The heat-exchangers and heat-pumps used, are costly, and ideally they will be placed where they can recover energy most efficiently.

Thus, given a grand composite curve for a process, choosing the temperature levels at which heat should be exchanged is an optimisation problem that involves both maximising the heat recovered, and minimising the equipment required to recover that heat.

The problem is non-linear and discontinuous and optimising it with conventional methods (such as mixed integer linear programming) requires a complex problem formulation. QMOO solves the problem, requiring only a very simple problem formulation, and giving excellent results.

- **Batch Processing** Krummenacher [73] studied the use of heat-exchanger networks for the recovery of heat in batch processes. The work primarily concerns the modelling of such systems, and covers a variety of possibilities for optimisation. Optimisation is with respect to the cost of each batch, but ideally also takes account of the heat recovered per batch. The optimisation can be of process parameters on a fixed heat exchanger network, or of both the design of the network and the process parameters. The networks were optimised using a single-objective EA, with some difficulty—a two-stage (to cope with the two objectives), two-level (for the design and process variables) optimisation process was developed, and considerable attention had to be paid to a distance function that measured the differences between pairs of solutions.

  QMOO was applied to the problem without any special preparation, and found entire two-objective solution sets ten times faster than the earlier, carefully tuned method could find a single solution.

  Given that the objective for the design of QMOO was an optimisation algorithm that was rapid, robust, and required very little tuning to the problem, this is a perfect result.

- **Configuration of Hybrid Vehicle Drivetrains.** The 'hybrid' problem involves choosing component configurations and sizes for a hybrid vehicle drivetrain. Given a vehicle configuration, a simulation of the vehicle is 'driven' through a number of standard drive cycles, and a number of performance measures are taken. Thus, using multi-objective optimisation, trade-offs between performance parameters such as emissions and fuel economy through an urban drive cycle can be studied.

  The modelling of the system and further analysis are presented in Molyneaux [89].

- **Configuration of a District Heating System.** The 'district heating' problem involves configuring a central co-generation power plant and the network of heat exchangers and heat pumps at client sites. The problem was originally studied in Curti et al. [24] as a single objective problem, using 'environomic' techniques to reduce two conflicting objectives (plant construction costs and emissions) to a single, monetary objective. Molyneaux [89] re-posed the problem as a multi-objective problem, in a study of the advantages multi-objective optimisation offered over single-objective optimisation in this case. Results from single runs of the multi-objective algorithm included all solutions found over a large number of runs in the earlier work, as well as many more interesting solutions.

The nature of the problem (an extremely disjoint optimal region) lead to the development of QMOO's grouping methods, and to the development of techniques for preserving the 'tails' of optimal regions, both of which were applied to a number of other problems.

This work showed again that QMOO was capable of finding a wide range of interesting solutions to the problems with less computational effort than the algorithm used in earlier work had taken to find a single point.

- **Optimisation of Heating and Co-generation Systems.** The work of Curti et al. [24] has also been extended to include that of Pelster et al. [102], which optimised co-generation systems. The new projects study more complex systems using co-generation, fuel cells and heat-pumps for both heating and cooling. QMOO is applied to these problems with very little tuning and provides excellent results.

- **Technology Options for Rural Areas.** Another study using QMOO at LENI concerns the integration of energy systems in remote rural areas (in this case, Tunisia) [100, 113]. Here, as well as considering the efficiency and costs of the technologies used (such as hybrid solar and co-generation systems), it must be ensured that a range of environmental factors, such as life-cycle impact, noise pollution water quality, and social acceptance must be respected.

  Another EA, GEATbx was tested on some of these problems, and QMOO was found to outperform it in every case, both in terms of convergence and of ease of use.

Chapter 2

Optimisation by Evolutionary Algorithms

2.1 Introduction

Energy system models have a number of features that make them difficult to optimise by conventional methods, and which make a number of demands on what information the solutions obtained from an optimisation should offer. It is not sufficient for an optimisation algorithm to provide a single optimal point for a model, instead, the optimiser must identify and characterise 'interesting regions' of the solution space, providing the decision maker with a number of choices, and as much information as possible about those choices. With this information the decision maker can make a qualitative choice about a good solution from among the regions returned by the algorithm, or can decide that some unexpected interesting region is worthy of further study, or even use the results of the optimiser to debug the model.

Evolutionary algorithms (EAs) can meet all of the requirements for an energy system optimiser. They are robust optimisers that will solve most classes of problems, needing no supplementary knowledge of the problem space (for example derivatives), and very little preparation of the system model. Furthermore, because, unlike almost all 'conventional' methods, they work with a population of potential solutions, a single optimisation run can return a range of solutions, illustrating either a range of separate choices in a multi-modal problem, illustrating a trade-off between objectives in a multi-objective problem, or both.

Often an optimisation problem can be solved more rapidly by conventional methods than by an EA, and where possible and practical, those methods should be used. However, the specificity of analytic methods can cause problems. Often, a simple 'first pass' model of a design problem will prove to be linear, and thus solvable by linear programming techniques—rapidly and efficiently. A more realistic model may require the presence of (0,1) variables—on/off switches, and again be solvable by an appropriate technique. Later the model may prove to be non-linear, requiring either moving to a non-linear optimiser, or linearising the non-linear parts of the problem into piecewise-linear sections. Two problems appear: firstly, the need to continually change the optimiser used, and secondly, the requirement to 'fit' the system model to the form required by the optimiser. In simple cases like the example above, such problems are easily overcome by a number of powerful optimisation packages, but at higher complexities, models will need to be re-designed and re-written, and interfaces to optimisation methods will need to be changed. The work required to make these changes can often lead to the decision-maker going to considerable lengths to assure that a system model fits into a particular optimisation method, compromising the model's fidelity with the real system it attempts to simulate.

EAs are attractive in these cases, because, given enough time, they will find a good solution to almost any problem, with only minimum preparation of the problem. In cases where the times for optimisation runs are large, and the systems modelled complex, the time lost due to the slower convergence of an EA will be more than repaid in time saved by not having to prepare the model, or not having to learn new optimisation techniques. But this can lead to the opposite problem to that above: the decision-maker, having found that EAs work on a wide range of problems, applies an EA all problems encountered, including linear problems, resulting in solution times of a few hours for problems that can be solved in a matter of seconds. It is well beyond the scope of this work to teach decision-makers best optimisation practise, but two important observations can be made.

Firstly, if a problem is being solved by an EA, and it is assumed that this is an appropriate use of an EA, then it can also be assumed that the problem has a some of the problematic features mentioned in the introduction. The problem is probably non-linear and disjoint, and so few assumptions can be made about derivatives—linear hill-climbing and quadratic interpolation should be only used cautiously, if at all. The problem may well be multi-modal, and so early convergence to a single optimum should be avoided, and preservation of a range of solutions should be attempted. The problem may well be multi-objective, so the presence of a good solution at one point in decision space indicates not only that the search should continue near that point for better solutions, but also in a larger neighbourhood of that solution for 'equally good' solutions that represent a different trade-off between objectives.

The second observation is that, though the system being optimised is probably 'awkward', it is, nonetheless, a model of a real system, and so not impossible to optimise. Even though assumptions about derivatives can not be made, the presence of a good solution at a certain point in decision space can be taken to imply the presence of other good solutions in the neighbourhood of that point. Exactly what *is* reasonable to assume about the form of the function being optimised will be discussed in §2.5, where a two of proofs concerning the convergence of EAs will be presented.

As a summary of the above, there are two rules of thumb to keep in mind for optimisation by EAs: Firstly, the problem *is* being solved by an EA, so one need not be too precise about its properties. An EA is heuristic, so there is not much point having any better than heuristic estimations of the model's properties. Secondly, the problem *is* being optimised, and is a model of a real system. The model will not appear entirely random, and in some neighbourhoods will probably behave reasonably nicely.

Having established these 'rules of engagement', the chapter continues by introducing the early history of EAs. Three original 'families' of EA development will be introduced, of these, particular attention will be paid to Genetic Algorithms, because the have had a large influence on the EA community, some of which needs to be carefully reconsidered.

We will then turn to an even simpler algorithm that an EA—a population-based algorithm—in order to investigate the elements needed to design an algorithm that will optimise. The population-based algorithm will pose a number of questions—significantly, which points to remove from the population, and how to add points to the population.

Discussing methods for removing points from the population will lead to an algorithm definition that is sufficiently detailed to introduce two proofs about the convergence of such algorithms. The first sets out sufficient conditions for convergence to an optimal solution in finite time, and consequently sets some ground rules for algorithm design. The second discusses limits on the potential performance of algorithms. However it will be seen that the class of problems to which the proof applies is too general, and in practise the limits imposed have no effect. Discussing the proof will nonetheless need a more detailed definition of the types of problems that are likely to be encountered by an optimiser.

While the first proof presented guarantees convergence under certain conditions, it says nothing (apart from 'finite time') about how long convergence will take. By carefully choosing where to place points that are added to the population, the convergence of the algorithm for certain problems—hopefully those likely to be encountered by an optimiser—will be improved. One of the best regions to add new points will be near the existing population. From this, the idea of finding a model of the distribution of the population, and drawing new points from that model will be developed. Then, noting that the 'distribution model' used by evolution—recombination and mutation between population members—is cheap and easy to implement, the conventional wisdom behind EAs will be turned on its head. Algorithms for optimisation are *not* attempting to simulate natural evolution, but are using tricks learnt from evolution to implement an optimisation algorithm based on distribution estimation.

Finally, a number of topics raised in the earlier sections will be revisited. There is a brief review of some of the operators used in EAs, discussion of parent selection and diversity, and the introduction of a very implementation-oriented topic—parallelism in EAs.

The introduction to EAs thus complete, the next chapter will examine issues specific to multi-objective optimisation, and how to apply EAs to such problems.

2.2 General History of Evolutionary Algorithms

A paper on simulating genetic systems by computer appeared in as early as 1957 [52], but concentrated entirely on simulation, not on using evolution-like procedures for optimisation. At the same time Box [10] proposed 'Evolutionary Operation'—the use of evolution-like methods for improving industrial productivity. However, despite some reflection on possibilities for automating the process, Box's work relied on having a 'human in the loop'—the method was not performed on a computer. It is generally considered that EAs for optimisation have their origins in the 1960's, when several independent groups of researchers described methods for optimisation that simulated natural evolution.

The researchers fell into three main groups, naming their EAs 'Evolution Strategies', 'Evolutionary Programming', and 'Genetic Algorithms', each of which approximated evolutionary processes in different ways. After a brief introduction to some terminology, these three groups are reviewed, along with indications as to where more detail about each of the methods can be found.

2.2.1 Terminology

In an EA, a *population* of *individuals* is *evolved* toward the solution of an optimisation problem by means of *operations* on the population which produce new individuals, and by removing individuals from the population.

An *individual* is a representation of a possible solution to the optimisation problem. It contains information on where the solution lies in the decision space of the problem, and also about the solution's degree of optimality.

The population evolves toward better solutions by two processes: how the *operators* act on the population; and how individuals are removed from the population.

Crossover and *mutation* are two operators that simulate natural reproduction processes. Crossover involves selecting two (or more) individuals from the population and creating a *child* (or children) that in some way resembles its *parents*, much like sexual reproduction. Mutation takes a single individual and changes some of its parameters a little, much like the natural mutation of the genetic code. In order to make sure that the children reflect the best of the current population (or are better than it), the parents can be *selected* from the the better parts of the population, much like a breeder of sheep or horses will select the animals best fitting her goals to be parents of the next generation. Which parts of the population are 'better' is judged on the *fitness* of each individual. Fitness may be a direct reflection of the individual's objective function value, or it may be modified in some way so that selection not only prefers individuals with low objective function values, but also preserves other characteristics of the population. One such characteristic is *diversity*—the spread of the population in the search space, which assures that the algorithm does not converge too quickly to too small a region, and thus get trapped in a local, non-global, optima.

Individuals are removed from the population in a number of ways. In *generational* EAs, the population is processed a *generation* at a time: a number of children equal to the size of the population is generated, and these children entirely replace the existing population. This method relies entirely on the selection of good parents to move the population toward an optimum. In *elitist* EAs, the generation structure is generally retained, but a few of the very best individuals in the population are retained from generation to generation, acting as a marker of the best performance of the algorithm, and contributing to convergence by having more children. In *steady-state* EAs, the generational structure of the algorithm is removed, and individuals are added to, and removed from, the population as necessary. Very often the criteria for removal of points in steady-state EAs result in some measure of elitism.

Given this brief overview, and introduction to the terminology, the three main 'branches' in the development of EAs noted above can be discussed.

2.2.2 Evolution Strategies

Evolution Strategies (ES) was initially developed in 1964-65 by Rechenberg and Schwefel at the the Technical University of Berlin, on the first computer available at that institute. The reader is referred to Schwefel [117; 118], Bäck [5], Fogel [46] for comprehensive discussion.

In the original ES of 1965 [116], a single individual, containing a vector of design parameters, is mutated by adding normally distributed random vector with a mean of zero and a fixed variance of σ (the same in all dimensions), to create a child. If the child it better than the parent, it is used as the next individual. Otherwise, the parent remains and the child is discarded. This algorithm is denoted a $(1+1)$ ES, meaning that there is a population of one, which generates one new individual, and from the union of the population of one and the new individual, the next population (of one), is generated. The $(1+1)$ ES is a trivial EA—with a population of only one, there is no crossover between individuals.

Rechenberg [109] developed a convergence rate theory for the $(1+1)ES$[1] based on minimising the 'sphere model':

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2 \tag{2.1}$$

The results were used to find a rule for changing the variance of mutations: $1/5$ of the children should be successful at replacing the parent. If more children are successful, then the search region is too small, and σ should be increased. If too few are successful, σ should be decreased.

Rechenberg also proposed a $(\mu+1)$ evolution strategy, where a population of μ individuals is used to generate a new child, using both crossover and mutation, and the child replaces the worst in the population if it is better.

Schwefel [117] went on to extend this to the $(\mu+\lambda)$ ES and the (μ,λ) ES. In the $(\mu+\lambda)$ ES the new population is made up of the best μ individuals from the total of μ old individuals and λ candidates. In the (μ,λ) ES, the new population is chosen from the best μ of *only* the λ children. Thus in the (μ,λ) ES $\lambda \geq \mu$. Schwefel also described a method for giving each individual I_i its own mutation variance, σ_i, and letting the σ_is evolve throughout the course of the optimisation. The logic behind the (μ,λ) search strategy was that the 'search parameters' (σ_i) of older individuals, now evolving, would quickly become out of date, and so it was better to remove older members of the population. The preference for (μ,λ) over $(\mu+\lambda)$ ES is a little surprising, especially as it has now been shown that by keeping the best members of a population from generation to generation convergence of an EA can be guaranteed. Tests, again on the sphere model, (2.1), indicated that a ratio of λ/μ of about 7 was a good choice—equivalent to giving an individual a life of seven generations.

The combination operators used in ESs can be varied, and fall into two large families: sexual and panmitic. In sexual combination, two parents are chosen, and parameter values are generated from the two parents. In panmitic combination, the whole population contributes to the new individual—one fixed parent is chosen, and for every variable another member of the population is chosen for combination. Bäck [5] provides a

[1] Reproduced in English in the more easily obtainable Bäck [5].

taxonomy of possible operators:

$$x'_i = \begin{cases} x_{a,i} & \text{no combination} \\ x_{a,i} \text{ or } x_{b,i} & \text{discrete} \\ x_{a,i} \text{ or } x_{b_i,i} & \text{panmitic discrete} \\ x_{a,i} + (x_{b,i} - x_{a,i})/2 & \text{intermediate} \\ x_{a,i} + (x_{b_i,i} - x_{a,i})/2 & \text{panmitic intermediate} \\ x_{a,i} + \mu(x_{b,i} - x_{a,i}) & \text{generalised intermediate} \\ x_{a,i} + \mu_i(x_{b_i,i} - x_{a,i}) & \text{generalised panmitic intermediate} \end{cases} \tag{2.2}$$

Discrete and generalised intermediate combination are discussed further in §2.7, along with other operators. Notably, the 'blend crossover' operator ($x'_i = x_{a,i} + \mu_i(x_{b,i} - x_{a,i})$) is missing from this mostly complete list. Despite the choices enumerated by Bäck, most studies only use discrete combination.

Systems of varying complexity have been developed for describing the shape of the mutation distribution. In the simplest, the variance is described by a scalar σ_i and the mutation space is a hypersphere around the individual, next, $\boldsymbol{\sigma}_i$ becomes a vector, one value for each decision variable, and the mutation space a hyper-ellipsoid. In the most complex case, a matrix of rotation angles $\boldsymbol{\alpha}_i$ is added to each individual and also allowed to evolve. The hyper-ellipsoid can now rotate in any direction, hopefully allowing it to follow the contours of the objective function—narrowing the search length in directions that are already close to the optimum, and lengthening it in directions where there is progress to be made. This is illustrated in Figure 2.1.

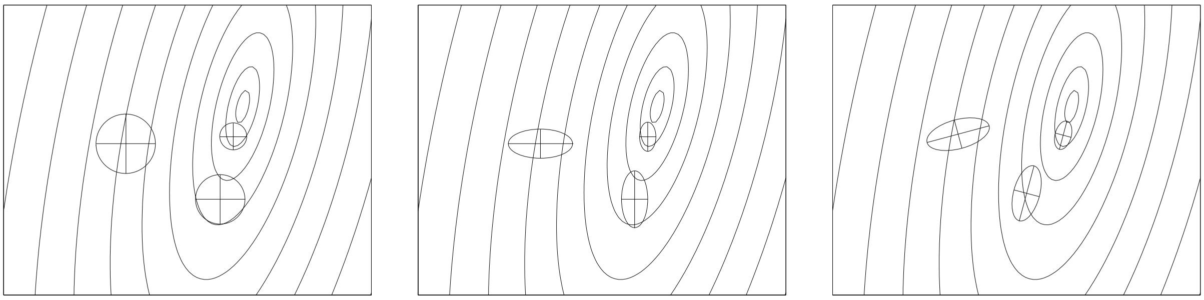


Figure 2.1: ES mutation radius variants. Left: scalar σ_i, mutation is in a circle around the individual. Middle: vector $\boldsymbol{\sigma}_i$, mutation is in an ellipse. Left: vector $\boldsymbol{\sigma}_i$ and rotation angles $\boldsymbol{\alpha}_i$, the ellipses can rotate.

Recently [119], ES has been extended to $(\mu, \kappa, \lambda, \rho)$ ES, with two extra parameters; κ is the maximum lifespan of an individual, in generations; ρ is the number of parents that will be involved in combination. The authors remark that $\kappa = 1$ is effectively the 'comma' strategy, while $\kappa = \infty$ is the 'plus' strategy. Oddly, they do not make the recommendation that $\kappa\lambda/\mu$ be 7.

2.2.3 Evolutionary Programming

Evolutionary Programming (EP) also has its origins in the early 1960's, when it was developed as a method of designing state machines for predicting sequences of symbols [48]. The method had some success, but was criticised because the sequences it predicted were relatively simple, and because for the method to work efficiently, carefully crafted penalty functions for mis-prediction were needed. In the mid 1980's Fogel and his son transformed EP (or effectively invented new techniques under the same name—see Fogel [45] for the history) into a method, much like ES, for solving real-valued optimisation problems, which is discussed here. For a more complete review of the early 'state machine' EP, the reader is referred to Fogel [45; 49].

Implementations of 'modern' EP are largely similar to ES (Bäck et al. [7] summarises the differences), the largest difference being that EP lacks any form of crossover. Fogel [46] argues that this is because in the EP view, individuals represent entire 'species', and not members of a single species. Furthermore, Fogel had empirical evidence, based on a few test problems, that the mutation used was more efficient than single-point crossover (defined in the §2.2.4) on a number of problems. Bäck [5] warns against these kinds of empirical results from a limited range of problems[2].

A second difference between EP and ES is the selection scheme used for choosing the new generation. An ES selection scheme, once defined by the (μ, λ) notation, or the $(\mu + \lambda)$ notation, is entirely deterministic—the best μ individuals of the available population are used for the population at the next iteration. EP, on the other hand, uses a stochastic selection strategy—'tournament selection'. In tournament selection, a set of new individuals the same size as the population is generated and, like a $(\mu + \mu)$ ES, is added to the existing population. Then, for each individual $I_s, s \in (1, ..., 2\mu)$, T individuals are chosen at random from the population, and compared against I_s. I_s is given a score equal to the number of individuals it is better than. Once the whole population has been given a score, the μ individuals with the highest scores are retained for the next generation.

The advantage of such a selection scheme is that the population retains a more diverse set of potential solutions than the ES scheme (diversity will be defined and discussed in detail in subsequent sections). This means that the population will not crowd around a local optimum as quickly as the ES scheme, there so is less likelihood that the algorithm will converge prematurely, and only find a local optimum (or reach a point where it only converges very slowly toward an optimum because the span of the population is small). The disadvantage is that sometimes some of the better individuals will be lost, and so will incur a cost in the convergence speed of the algorithm.

It is important to note that though the ES selection procedure is stochastic, it guarantees the inclusion of the best point in the current population in the next population (as it will always obtain a score of T). This point is important with respect to the proof of convergence that will be presented in §2.5.

For more information on EP, the reader is referred to Fogel [46] or Bäck [5].

[2] This is an interesting criticism from a member of a research group whose two 'rules of thumb' (the $1/5$ and $1/7$ rules) come from theoretical and empirical studies of *just one* problem.

2.2.4 Genetic Algorithms

Like other methods for evolutionary optimisation, Genetic Algorithms (GA) trace their roots to the early 1960s The modern concept of a GA is based on work by Holland in the 1960s [62], and later by Goldberg [57]. GAs are probably the best known of the three families of EAs, though not necessarily the most appropriate for energy system optimisation.

GAs have a number of characteristics that distinguish them from other algorithms:

- GAs use a 'binary encoding' scheme to encode decision variables;
- the 'canonical' GA is strictly generational[3]—subsequent generations completely replace the previous, and thus are non-elitist;
- GAs use a 'roulette wheel' parent selection method to make the population evolve toward better solutions.

GAs have had a large influence on the development of evolutionary algorithms, but in the opinion of this author, this influence is somewhat unjustified. Design features of the 'canonical' GA are not used in this work. However, given their reputation, some care will now be taken to justify this choice.

Binary Encoding

Binary encoding involves 'coding' all of a problem's decision variables into a binary representation, and having crossover and mutation operators that work directly on that binary representation. Thus a mutation operator chooses whether or not to change a *single bit* of a binary representation, and a crossover operator (for example 'uniform crossover'—Bäck's *discrete* operator) chooses individual bits from each parent. A popular crossover operator is the 'single crossover' operator. Here a bit position is chosen randomly and bits before that position come from the first parent, bits after, from the second, as shown in Table 2.1.

Parent 1	**1**	**0**	**1**	1	0	1	1	1
Parent 2	0	0	1	**0**	**1**	**0**	**0**	**1**
Child	1	0	1	0	1	0	0	1

Table 2.1: Single crossover at the third bit.

In the case of binary decision variables (0,1 integer problems), binary coding is a logical representation choice. However, in real-valued problems, the continuous search domain must be divided into 2^{B_n} discrete points, where B_n is the number of binary bits used for the representation of the real variable n. B_n must be chosen carefully beforehand in order to have acceptable resolution of an optima, which can prove difficult.

Representation of integer values is even more problematic than for real variables—if the range of the integer value is not a power of 2, something must be done to deal with the remaining binary values. One way of

[3] this is not the case for the steady-state GA

Integer	0	1	2	3	4	5	6	7
Binary	000	001	010	011	100	101	110	111
Gray	000	001	011	010	110	111	101	100

Table 2.2: Gray Coding

resolving this problem is to over-specify the integer variable. Thus, to represent an integer value from 1 to 5, a B of 9 (and thus a range of 0-1023) could be chosen, with the first 205 values representing 1, the next 205 2, and so on. However, solutions such as these seem to lack elegance.

These are not the only problems binary encoding poses. Because binary encoding works at the bit level, individuals generated by mutation tend to be a few bits apart. Because all bits are equally likely to be changed, this means that a good parent with a value of 1 (and a binary encoding of 000001) is as likely to produce a child with a value of 33 (100001) as 0 (000000), even though our assumptions about the nature of the problem to be optimised mean that it is much more likely that 0 is a good choice that 33 is. A further problem is that its nearly impossible to get from a value of 31 (011111) to a value of 32 (100000) by mutation. At these points, termed 'Hamming cliffs', too many bits need to be changed to get to a neighbouring value. To get a around this problem, a Gray coding can be used [14]. In a Gray coding (shown in Table 2.2) only one bit changes between consecutive integer values. However, the relation is not two-way—one has to change the correct bit in order to get a change of one in the represented value While the problem with Hamming cliffs is avoided, Gray coding does not make it much more likely that a mutation will result in a child near a parent.

Another problem with binary encoding is that the precision of the solution required must be effectively specified before the optimisation starts—the number of bits to represent each decision variable must be chosen. If the search space is very large, a large number of bits must be chosen. Unfortunately, the binary operators have no preference for exploring the 'larger' bits of the binary representation before exploring the less significant ones. This means that if the binary representation of a decision variable is very long, the GA may well spend a lot of time trying to optimise insignificant low order parts of the representation, (whether, say, an integer parameter is even or odd) before first establishing general search ranges (whether the same parameter is more or less than 1024, for example).

Schraudolph and Belew [115] propose 'Dynamic Parameter Encoding' (DPE) to solve this problem. In DPE, the problem is first solved at a relatively low resolution over a large domain. Once the population is found to have sufficiently converged in the large domain, the domain is redefined to fit the new limits of population, and the algorithm continues. The process is repeated as many times as necessary to obtain the desired precision on the problem. DPE dramatically improves convergence on some problems, but is detrimental on others. DPE is particularly bad at problems where algorithms risk converging prematurely to non-optimal regions—there is a possibility that the *real* optimal region will be completely eliminated from the search space before it its explored.

The original motivation behind binary encoding of variables is not clear. Holland was both a psychologist and computer scientist, and GAs *were* intended to simulate biological systems. By 1968 though, Holland had developed the theory of 'schema processing'[4].

A *schema* is a pattern in 'encoded variable space'. For example, if a problem is encoded into a five-bit binary representation, then (1****) is a schema representing all patterns beginning with a 1, while a schema (**1*0) is all patterns with a 1 in the third position, and a 0 in the last. Schema processing counts the number of schema that are 'processed' by a population in a single generation, making the implicit assumption that this count is an reasonable indicator of algorithmic performance. Holland shows that a binary GA processes p^3 (where p is the population size) schemata per generation, and that encoding systems with low cardinality will process the most schema per population member per generation (meaning binary encodings are the best) and used this as an argument for superior performance of GAs compared to other EA approaches.

Despite practical evidence that binary encoding was not outperforming other methods, in 1989 Goldberg dismissed ESs saying 'the use of real parameters limits the schema processing inherent in these methods'. A large number of studies on both real and test problems showed no advantage for binary coded GAs, and in fact, that representation appropriate for each variable was more effective. In 1997 Fogel and Ghoziel [47] showed that 'functionally equivalent algorithms can be constructed regardless of the chosen representation'—ending any confusion as to the status of binary encoding.

Nonetheless, binary encodings persist in EA research, and continue to pose problems for researchers. Dasgupta and Michalewicz [26] note the problems associated with matching an encoding in a GA to a problem, and Ronald [111] discusses the problem of *encoding brittleness*—where similar problems require completely different encodings on order to be solved—and develops a 'robust encoding system', where variables are encoded in—surprise—their 'natural form'. A simpler criticism comes from Alander and Lampinen [2], where a 'real-coded GA' is used for internal combustion engine valve cam shape optimisation. Here, comparison of the real-coded GA with a binary-coded GA showed no difference in terms of function evaluations required for convergence, but because the objective function evaluation was quick, the time needed to encode and decode the binary representation doubled the elapsed time for an optimisation, from three to six hours.

Binary representations will be considered no further in this thesis.

Parent Selection in Generational GAs

The strictly generational nature of GAs, and the use of 'parental selection' can be treated at the same time. Effectively, parental selection is the remedy used by GAs to compensate for the fact that it lacks elitism—not preserving the best of each generation—caused by the strict generation approach. Unfortunately, parental selection provides no convergence guarantees, and should be seen as a compliment to elitism, rather than an alternative to it. The original method of parent selection used in GAs, 'roulette wheel' selection is briefly presented here. Parent selection will be reviewed in more detail in §2.8.

In roulette wheel selection, individuals are given a number of chances to contribute to individuals in the new

[4]For a comprehensive discussion of the theory behind schema processing, the reader is referred to Goldberg [57]. For a quick, and rather cutting appraisal, see Fogel [46].

population that is proportional to their fitness. For this to work, fitness must be maximised, and all fitness values must be positive, and a linear transform is usually applied to the objective function to ensure this. Thus, in a case of a population of four with transformed fitnesses of 10, 8, 7 and 5 (totalling 30), the first individual would contribute to 66% of new individuals (not 33%, because each new individual requires two parents), the second 53%, the third 47% and the fourth 33%.

Roulette wheel selection has been widely criticised, mostly because of the need for a linear transform. Generally, it is claimed that as the algorithm converges, all transformed fitnesses will become both large and similar, and selection will generally break down. However, there is nothing to imply that the linear transform must be fixed, nor how it should be chosen. One can easily imagine schemes where, for example, the linear transform is chosen such that the 'best' individual contributes to (say) 50% of all new individuals, and the worst, only 10%. Such issues will be covered in §2.8.

Steady-State GAs

The Steady-State GA (SSGA) was first introduced by Holland [62]. In this work, Holland proposed two 'plans' for implementing a GA. The first was the standard generational GA. In the second, at each 'generation' only one new individual is produced (from parents selected in the normal manner—preferring better parents) and replaces a *random* member of the population. Holland showed that *if* the fitness of an individual (relative to the population) did not change much over its lifetime, then it was expected to produce the same number of individuals as in a generational GA, and that the two algorithms seemed functionally equivalent. Initially the generational GA was preferred, because it was easier to implement on the computers available at the time.

It is unclear *why*, in practise, one would want the two algorithms to be functionally equivalent—in the context of Holland's work the equivalence was used in the derivation of results for schema processing. In any case, DeJong and Sarma [36] showed that the fitness of an individual did change over its lifetime, and more, that as the lifetime of an individual was highly variable, individuals rarely had the number of offspring that would be expected in the generational GA—the number that would be expected given their fitness at the moment they entered the population. The observed effect was that the convergence of the SSGA was more variable than that of the generational GA. Two methods for rectifying this 'problem' were suggested—a first-in-first-out population management scheme, where the oldest individual was removed , in order to ensure that individual's lifetimes were the same, and a scheme where any children than an individual had beyond its expected number of children where removed. Again, the reason for wanting the SSGA to be like the generational GA was unclear. There are arguments for achieving a more stable convergence, but other methods exist for obtaining faster and more stable convergence.

Rogers and Prügel-Bennett [110] analysed an SSGA where the removal of points also involved some selection. Both the parent of a new individual (there was no crossover) and the individual to replace were chosen using a Boltzmann distribution, that preferred better parents, and removed worse individuals. The Boltzmann distribution was used because it made subsequent analysis possible. The analysis showed that such an SSGA outperformed an SSGA using random removal, and a generational GA by a factor of two—the better performance coming from the removal strategy.

The term ‘steady-state’ is now applied to any EA where there is no generational structure. It may imply that the algorithm also performs some selection of the individuals that are removed from the population, though as has been seen, this is not the case for the original SSGA. Equally, some generational EAs now keep the best part of the population from generation to generation—effectively a method of selecting individuals for removal.

In this work, a steady-state algorithm is used. In contrast to the findings of earlier work, a steady-state algorithm was easier to implement.

2.2.5 Summary

The history of the development of evolutionary algorithms has left its mark on the field.

There are questions of terminology—many researchers will refer to an EA of any form as a GA, which is not at all unreasonable—while others will immediately insist that such an algorithm can not possibly be a GA because it does not use a binary variable representation. This can lead to some confusion, and consequently many researchers now prefer the term EA—as used here—which has not been used by any particular research group.

Researchers are often left with a false impression of what will make an efficient algorithm. For example, Miller et al. [86] presents the application of a ‘GA’ to designing electronic circuits, and states ‘Contrary to initial expectations it was found to be beneficial to employ elitism’, a view quite likely based on the history of the GA. However, in the same year, Rudolph [112] presented a proof showing just how important elitism is to the performance of EAs.

Thus in the next section, an algorithm simpler than an EA is described. This algorithm will be developed over the following sections along a quite different path to that taken historically by the EAS. Nonetheless, the development will arrive at very similar conclusions.

2.3 Population-Based Algorithms

To give a general overview of EAs, discussion will commence with the larger class of algorithms—generally termed ‘population-based algorithms’ (PBA).

Simply put, a PBA works in the following way:

1. Given a starting population of ‘individuals’ in the search space
2. Generate a number new individuals and add them to the population
3. Remove a number individuals from the population
4. Repeat until finished.

Compared to the precise definitions of a GA or ES, nearly everything in a PBA is left unspecified:

- there is no indication of how to create new individuals;
- there is no indication of which individuals to remove from the population;
- there is no requirement that the algorithm be ‘generational’, or even that the population size remains constant;
- there is no specification of how an individual must be coded;
- there is no specification of what the algorithm is ‘looking for’.

Because of its simplicity, the PBA is a powerful tool for both examining what must be done in order to implement an effective search, and for analysing the choices made in earlier EAs.

Nonetheless, some key aspects of the PBA have already been defined. In contrast to almost any conventional optimisation method, the PBA works with a number of points in parameter space simultaneously, as opposed to following a single point around that space. This immediately makes a PBA an effective tool for both multi-modal and multi-objective optimisation—each individual in a ‘final population’ can represent a different but nonetheless interesting solution. Furthermore, no assumptions at all have been made about the form of the objective function—it need not be continuous, and no derivative information is necessary—the objective function can effectively be a ‘black box’. There is a price to pay for this however—while under certain conditions there are convergence guarantees for PBAs, these guarantees say little about how fast convergence will be.

A PBA can be a search algorithm for a point or region in the search space that fills almost any criteria. This thesis, however, is concerned with optimisation, and so the most significant question above: ‘what are we searching for?’ has been answered. From here on, the development of the PBAs and EAs will be explained in terms of optimisation. Some of what follows may be applicable to other cases, such as root searches or constraint-satisfaction problems (the search for a feasible region), but these subjects do not fall into the domain of this work.

In order for the skeletal algorithm above to become an effective tool for optimisation and characterisation, we must answer several more questions. The two most significant ones are:

- choosing which individuals to remove from the population;
- how to generate new individuals.

The following sections will present an overview of the vast range of approaches that have been used to answer these two questions, and hopefully dispel any confusion that may result from the early history of ES, EP, and GAs, or from the use of the words Genetic and Evolutionary.

2.3.1 Random Search

Here we briefly note that from the initial PBA, a random search can be easily generated. An initial population is not even needed, no individuals are removed, and new individuals are generated randomly. Stopping criteria may vary: ideally the algorithm will stop when one of the individuals in the population has certain properties (such as a function value of zero in a root search, or a low objective in an optimisation), but often the stopping criteria will be based either on elapsed time for the search, or the number of individuals visited. Notably, a random search fulfils the criteria set out in §2.5, for guaranteed convergence. Such a search is illustrated in Figure 2.2.

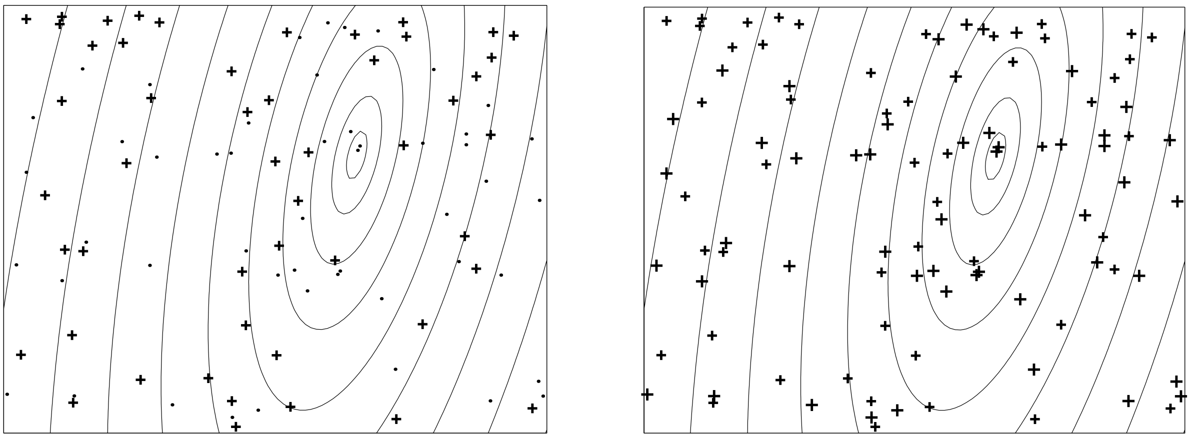


Figure 2.2: A Random Search 'Converging' to an Optimum. Contours of a two-variable optimisation problem. Left: the first fifty random points (+) and the 'next' fifty (.). Right: all 100 points—happily, the best point is now closer to the optimum.

Alternatively, a random search can be seen as PBA with a population of one, where at each iteration a new random point is generated, and if the new point is 'better' than the existing point, the existing point is replaced—an example of a $(1 + 1)$ algorithm using ES notation.

These cases of the random search illustrate the questions we will discuss next—how to decide if a point is better than another, and how to generate a new point in a better manner than randomly.

2.4 Choosing Which Points to Remove

In a PBA, given an individual, we must have some way of calculating the 'desirability' of that individual. In single-objective optimisation, the desirability could be taken to be the objective function value for that individual, in multi-objective optimisation however, as will be discussed in Chapter 3, it cannot.

Nonetheless, without yet having a good definition of desirability, one can imagine that an algorithm that tends to keep more desirable points in the population, and remove the less desirable ones, will tend toward a more desirable population.

Surprisingly, many EAs do not work like this at all—take the example of the canonical GA. Unfortunately for such algorithms, while a proof of convergence exists for algorithms that keep 'good' points in the population (§2.5), there is none for those that hope to generate 'better' individuals while ignoring the good points already in the population.

For the moment though, the problem of how new points are generated will be ignored—one can assume randomly, though it will be shown later that we can do much better than that—and focus on the problem of deciding which points to remove, which will lead to a clearer definition of desirability.

An obvious tactic is to assume that desirability is some function of the objective function only, and remove the individuals with the worst objective function values (or, put the other way, keep the best individuals in the population). However, following the 'rules of engagement' presented in the introduction, the problems to which a PBA or EA will be applied will be generally 'difficult' problems, and such problems can cause difficulties for an algorithm that preserves only the points with the best objectives.

Specifically, if new points are being generated in a manner that makes them resemble the existing population (§2.6 will discuss why this is a good idea), then removing points based solely on their objective function values will result in an algorithm that performs a less-than-perfect exploration of the problem domain. Such an algorithm will converge quickly to the 'easily found' neighbourhoods of a few local optima and not to a large proportion of the local optima (Figure 2.3).

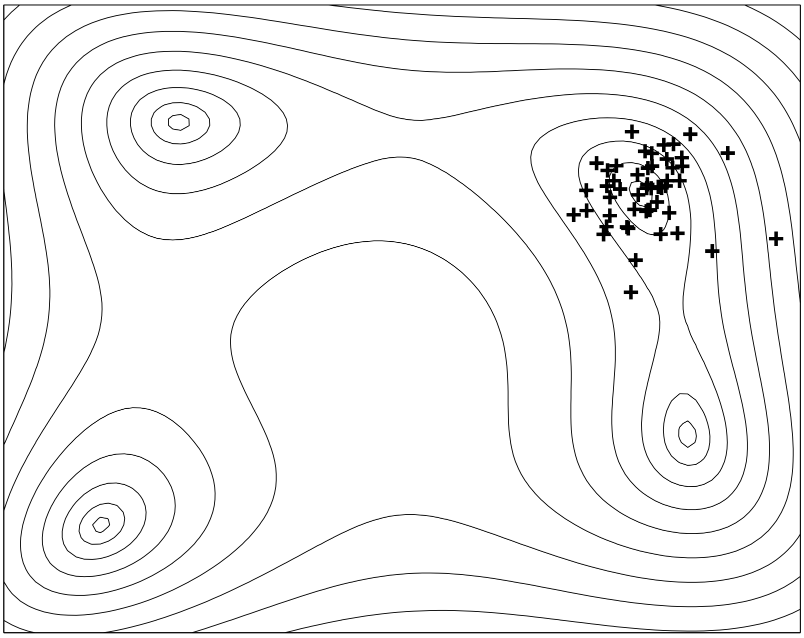


Figure 2.3: Convergence to just one of several local optima is best avoided. The objective function has four local optima, each of with the same optimal value. Ideally an algorithm will find them all.

These problems suggest that desirability should encompass more than just the objective function value, and that it should be used to preserve the 'diversity' of the population. Preservation of diversity is a preoccupation of work on EAs, and will be discussed in §2.9.

An elitist algorithm (which keeps only the best part of the population) is illustrated in Figure 2.4. Despite the apparent improvement on random search, new points are still generated randomly, and convergence is no better.

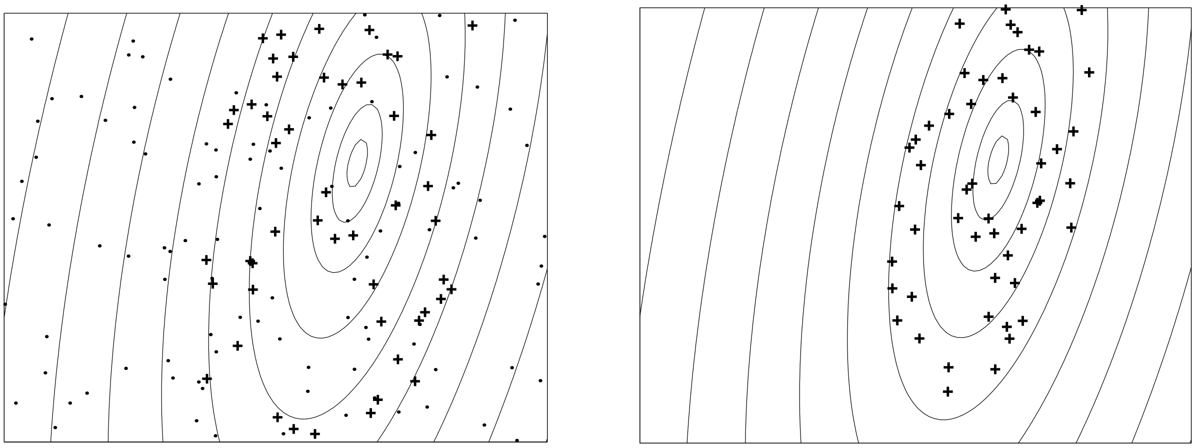


Figure 2.4: An elitist algorithm with randomly generated new individuals. Left: the best individuals 'so far' (+), and new individuals (.). Right: the best of the combined population—the population is now more concentrated around the optimum.

Although the PBA developed so far is far from finished and therefore would not make an efficient optimiser, sufficient detail has been introduced to that a proof of convergence for the algorithm can be presented. Further, the proof will formalise the rather vague assertions that 'it seems best to keep the best points in the population', and 'it seems like a good idea to preserve some kind of diversity'.

2.5 Proofs of Convergence for Population-Based Algorithms

Rudolph [112] presents a general proof of convergence for both single- and multi-objective EAs obeying certain conditions. Using the results from the proof an algorithm that is *guaranteed* to converge in finite time can be constructed. However the guarantee should not be overstated: beyond the assertion 'finite time', no indication of the time to converge is given—performance beyond that of random search is not guaranteed.

Wolpert and Macready [135] show that the average convergence performance of *any* search algorithm over *all possible* objective functions can be no better than random search. However, we are unlikely to wish optimise most of the members of all possible objective functions (a set which is largely made up of random objective functions), and in the opinion of this and other authors, the proof is largely irrelevant for optimisation of real systems.

Other authors have imagined problems that will be difficult for an optimiser to solve, and have shown that these problems can be strikingly similar to real problems. The problems constructed, however, are not real problems, and the question remains as to whether it is likely that a decision maker will encounter such a problem.

2.5.1 Conditions for Guaranteed Convergence for Population-Based Algorithms

Rudolph [112] presents a proof of convergence for 'Evolutionary Search for Minimal Elements in Partially Ordered Finite Sets' for evolutionary searches that meet a number of conditions. For full details of the proof, the reader is referred to the original work, here the necessary details of the proof are presented so that the conditions on the algorithm can be presented and discussed.

Rudolph's proof is unique in that it treats *partially ordered* sets. This is critical because it means that the results can be extended to multi-objective optimisation, whereas earlier proofs could not. We will now examine the elements of the title, leading to the conditions for convergence.

An *ordered set* is formed by a set $\mathcal{X}$ and a relation between members of the set obeying a number of conditions, which are not discussed here. These conditions imply a relation like $\leq$, with the result that $\mathcal{X}$ can be ordered by the relation, i.e. $\forall x, y \in \mathcal{X}, x \leq y$ or $y \leq x$ and if $x \leq y \wedge y \leq x$ then $x = y$. Thus a sequence $x_1 \leq x_2 \leq x_3 \leq \ldots \leq x_n$ can be created. Furthermore, given a relation like $\leq$, we can find a single global minimum $x^* \in \mathcal{X}$ such that $\forall y \in \mathcal{X}, x^* \leq y$.

A *partially ordered* set differs from an ordered set in that some members of $\mathcal{X}$ are *indistinguishable*, that is, if the relation for a partially ordered set is denoted by $\preceq$, then $x \preceq y$ and $y \preceq x$ *does not* imply that $x = y$, only that they are indistinguishable under $\preceq$, and so a simple sequence such as above cannot be created. Nor is there necessarily a single global minimum, instead there is a set of local minima $\mathcal{M}(\mathcal{X}, \preceq)$ such that $\forall x \in \mathcal{X}, \exists m \in \mathcal{M}(\mathcal{X}, \preceq)$ such that $m \preceq x$. As we shall see in §3.4, the Pareto dominance relationship introduced for multi-objective optimisation results in a *partial ordering* $\forall \boldsymbol{x} \in \mathcal{R}^n$

Rudolph's proof applies to *finite* sets—it does not apply to continuous optimisation. In practise this does not pose a problem. Optimisation is generally performed on computers with limited precision, and so problems optimised on computers are optimisations over finite sets.

The EA used as a first example in the work encompasses the random search presented above—as an EA it is certainly a PBA (but not a GA or an ES). Rudolph's results in fact, apply to all algorithms that we have termed PBAs. The first EA, searching for a minimum of the objective function f over a space $\mathcal{S}$ is a generalised $(1+1)$ EA[5]:

1. Generate an individual $x_0 \in \mathcal{S}$ at random and set $k = 0$.
2. Apply some variation operator to obtain an offspring $y_k \in \mathcal{S}$ from x_k.
3. If $f(y_k) \preceq f(x_k)$ then set $x_{k+1} = y_k$ otherwise $x_{k+1} = x_k$.
4. Increase k and goto (2) unless some termination condition is fulfilled.

The algorithm thus generates a sequence $(x_k : k < 0)$. Rudolph proves that this sequence converges in finite time to $x_k \in \mathcal{M}(\mathcal{X}, \preceq)$ and stays there *if* the variation operator is such that $\forall x, y \in \mathcal{X}, P(y, x) \geq \delta > 0$— given any parent x, there is a finite probability that the child y can be anywhere in the search space. Thus, the $(1+1)$ random search presented above (where every point in the search space has an equal probability of being selected as a child, regardless of the parent), is guaranteed to converge. Other algorithms, which may prefer to place the child in the neighbourhood of the parent can also be guaranteed to converge if their

[5] the algorithm is taken literally from Rudolph's paper

preference for the region of the parent still leaves a small probability of generating a child anywhere in the search space.

Rudolph then generalises this result to a $(\mu + \lambda)$ EA, and the second condition for convergence becomes clear. This condition appears in the algorithm above in (3)—the criteria for replacing x_k with y_k. In an algorithm with a population larger than one, this becomes the condition that the algorithm must obey the *elite preservation strategy*:

> Let $(x_k^1, x_k^2, \ldots, x_k^n)$ be the population of parents at time k, and $\mathcal{Y}_k = (y_k^1, y_k^2, \ldots, y_k^N)$ be the children generated at k. Let one parent, say x_k^1, be the *elitist parent*, then if $\nexists y \in \mathcal{Y}_k$ such that $y \preceq x_k^1$, then let $x_{k+1}^1 = x_k^1$. Otherwise, choose any $y^* \in \mathcal{Y}_k$ such that $y^* \preceq x_k^1$ to assign to x_{k+1}^1.

An algorithm due to Peschel and Reidel [103] for multi-objective optimisation (here defined as finding *all* of $\mathcal{M}(\mathcal{X}, \preceq)$ of a partially ordered set, rather than just one element as in the first algorithm) which obeys the elite preservation strategy is as follows:

1. Generate a collection $\mathcal{P}_0$ of $n_o = N$ parents at random and set $\mathcal{O}_0 = \mathcal{P}_0$.
2. Select those offspring whose images (in f) are minimal in the partially ordered set $(f(\mathcal{O}_0), \preceq))$. This yields n_1 distinct parents after deletion of potential duplicates. Set $k = 1$.
3. Generate a collection $\mathcal{O}_k$ of N offspring from the collection $\mathcal{P}_k$ of n_k parents by variation.
4. Select those individuals from $\mathcal{P}_k \cup \mathcal{O}_k$ whose images are minimal in the partially ordered set $(f(\mathcal{P}_k \cup \mathcal{O}_k), \preceq))$. This yields n_{k+1} distinct parents after deletion of potential duplicates.
5. Increase k and goto (3) unless some termination criterion is fulfilled.

Rudolph shows that this algorithm, which meets the conditions for guaranteed convergence, will converge in finite time to the *entire* minimal set of the search space (though n_k may become very large, as the search space is finite, it will remain finite).

The two conditions for guaranteed convergence presented here are of fundamental importance, and are formalisations of ideas discussed in the previous section.

Firstly, the requirement that there be a finite probability of generating a child *anywhere* in the search space, is a formalism of the earlier statement that it would seem best to preserve diversity to ensure a thorough search of the decision space. Secondly, the suggestion that a good 'removal' strategy would be to remove the worst individuals from the population and to keep the best, has become a requirement for a precisely defined form of elitism.

2.5.2 The No Free Lunch Theorem

Wolpert and Macready [135] present the 'No Free Lunch Theorem' for optimisation[6]. This theorem states,

[6] note that the paper appeared as a technical report in 1995, thus some of the subsequent papers which refer to it were published earlier than 1997

given its preconditions, that the average performance of a search algorithm over *all possible* objective functions can be no better than any other search algorithm, and consequently, no better than random search. The preconditions are relatively simple:

- the search space must be discrete and finite. Since most optimisation problems are solved on computers with finite precision on floating-point values, this assumption is reasonable;
- a search algorithm is an algorithm that searches a sequence of points to see if they are optimal. The algorithm never searches the same point twice (this point is significant).

For details of the proof of the NFL, the reader is referred to the original work, or any number of papers which re-state the proof in a variety of (more and less comprehensible) forms; for example Radcliffe and Surry [108], Droste et al. [37; 38].

The original authors are fairly circumspect about the conclusions that from the NFL, but do state that the NFL indicates that just because an algorithm performs well on a particular problem, this does not imply that it will work as well on other problems. This has been widely interpreted as meaning that if an optimisation method is effective on one class of problems, then it must necessarily perform poorly on another class of problems [25, 132]. Taken literally, this interpretation is true, however, in practise, the assertion becomes mostly irrelevant.

To see why, recall that NFL talks about average performance over all possible problems. To understand the significance of its results it must be clear what 'all possible' problems means, and the question whether it is likely that an optimisation algorithm will be used to optimise any possible problem must be answered.

The NFL requires discrete search spaces. This means that a function can be defined as a mapping from a finite number of parameter values, to a single discrete value. Thus any optimisation problem covered by the NFL can be described as a mapping from an integer in a search space $\mathcal{S}$ to an integer in an objective space $\mathcal{F}$. Even if the set of machine-precision real parameters $\boldsymbol{x}$ is a vector, or the problem is a multi-objective problem and thus $\boldsymbol{u}$ is a vector, because the elements of $\boldsymbol{x}$ or $\boldsymbol{u}$ are discrete, there is a mapping from $\boldsymbol{x}$ to an integer $\mathcal{S}$ and $\boldsymbol{u}$ to an integer $\mathcal{F}$. This means that an objective function can be viewed as a long list of (s, f) pairs, or even just a list of values in $\mathcal{F}$ for increasing values in $\mathcal{S}$. For example, if we have a two variable optimisation problem, where both elements of x are integer that can vary from 1 to 5, with a single objective function which can take integer values between 1 and 10, then $\mathcal{S}$ could be the set of integers from 1 to 25, and an objective function would be any string of 25 integers between 1 and 10. There are thus 10^{25} possible objective functions in this space. In a more concrete case, there may be 10 32-bit floating-point parameters, with one 32 bit floating-point objective. In this case there are $(2^{32})^{2^{320}}$ possible objective functions—very roughly a googleplex[7] possible functions.

Many of these functions are 'purely random'—they contain no structure, and are probably of no interest in terms of real world optimisation problems at all, especially if the likely properties of a function to be optimised are taken into account. We now investigate whether there is some way to classify the problems

[7] $10^{10^{100}}$—ten to the power of one followed by a hundred zeros.

likely to be optimised in mathematical terms, and whether the size of this set of problems relative to the space of all possible problems can be estimated.

Sharpe [122], discussing the NFL, asserts that all optimisation algorithms are designed on some basic assumptions[8]:

1. Points in the search space with high fitness are indicators of the location of points in the space with higher fitness;
2. There are local correlations in the fitness landscape;
3. There are global correlations in the fitness landscape;
4. There are sufficiently many points that represent successful solutions that a search strategy can find them, but not so many that random search will find one faster.

Sharpe continues, citing a number of works that indicate that these assumptions are more than reasonable. Sharpe's assumptions are critical to the development to search algorithms other than random search, and reference will be made them in a number of places in this chapter, when discussing both convergence behaviour of EA, and techniques used to improve convergence. However, though it is clear that an optimiser is unlikely to encounter problems for which these assumptions do not hold, the assumptions do not, in themselves, suggest a method for classifying these problems.

Droste et al. [37] state that "nobody in applications is concerned with the unrestricted black box optimization scenario which is the base of the No Free Lunch Theorem", and shows that there are a number of classes of problems from which one can imagine that real problems will be drawn, and shows that NFL does not apply to these classes of problems—that is, one algorithm can perform better than all other on all the problems in one of these classes. The classes of problems considered are:

- problems that have limited time complexity—that is, problems that for which the objective function can be evaluated in at most polynomial time;
- problems that are limited in the complexity of the circuit that is needed to evaluate them—problems for which the objective function can evaluated by an existing computer;
- problems of limited Kolmogoroff complexity. Kolmogoroff complexity measures the amount of 'information' contained in a function, and is a measure of the 'shortest' way of writing a problem. Thus a function such as x^2, which can be written very easily, has a low Kolmogoroff complexity, a function that takes a large computer program has a higher complexity, and a completely random function, where the definition is effectively the list of the values the function takes, is of very high complexity.

Thus NFL does not apply *within* the classes of problems that, in simple terms are only problems that are possible to solve on a computer. However, NFL still applies to these classes in that an algorithm that solves

[8]verbatim from Sharpe

all polynomial time complexity problems very efficiently must ‘pay’ on the rest of the problem space. If polynomial time problems occupy a large portion of the problem space, algorithms that perform effectively on them will have to pay their good performance on the small portion of the space that is not occupied by polynomial time algorithms—implying that designing such an algorithm would be extremely difficult.

An idea of the size of the space of problems with limited Kolmogoroff complexity can be obtained easily from basic information theory [21]. Simply, the Kolmogoroff complexity is the length of the shortest string of bits that represent the data in question. Thus, if a string of bits (data) can be represented by a shorter string of bits (compressed data), then the longer string has a Kolmogoroff complexity of no more than the length of the shorter string. How many bit strings, though, can be represented by shorter strings? It is found that no more than half the strings of N bits can be represented in N-1 bits, simply because there are only half the number of 1-bit shorter strings. This estimate is generous, because both the length of the longer string, and the compression algorithm need to be stored into the shorter string.

Consider an example of a single objective optimisation problem with two 32-bit parameters, and one 32-bit objective. A *single* objective function, written as a list of f values, takes 32×2^{64} bits (roughly 67 million Terabytes) to store. Requiring that the function be compressible by just four bytes will result in a set of functions just $1/2^{32}$ (about one four-billionth) of the size of the entire set. This function still can’t be stored on any computer on the planet in 2002. Requiring that the functions be representable on a computer of reasonable size clearly leaves an enormous number of functions, but only a vanishingly small fraction of all possible functions.

Like Droste et al. [37], we conclude that, for anyone involved in applying optimisation to real, representable problems, the NFL is irrelevant.

Algorithms that Revisit Points

Radcliffe and Surry [108] points out that ‘Wolpert & Macready only consider algorithms that do not re-visit points already considered, and seem to regard the issue of revisiting as a trivial technicality’. In fact, thought, most search algorithms probably *do* revisit points, and though they *may* not in some cases, actually preventing them from doing so is impractical. Computer memories are probably large enough to store most points visited by a search algorithm, but processing the data (finding if a new point matches any previously visited point) will take a prohibitive amount of time.

Nonetheless, the performance of algorithms that revisit points, and those that manage to avoid doing so, is probably different, and the performance of those that do not revisit *could* be better[9]. Thus, it is quite likely that an optimisation algorithm that revisits less points than another can outperform the second algorithm over all problems considered by the NFL.

[9] No proof exists yet, and the algorithm would have to take advantage of this property.

2.5.3 Difficult Problems

Droste et al. [38], having shown that NFL does not apply to the classes of problems that are likely to be optimised, go on to consider problems that are hard for evolutionary algorithms, and show that some of these problems are very similar to problems that are easy to solve.

The underlying logic is fairly simple. If one assumes that a sensible optimiser will tend to search in the region of the better points that it has already found, one can show that that it is unlikely to find an optimal solution surrounded by poor points, and more, that the optimiser can be lead away from the region of the optimal solution by a more obvious non-optimal solution in another region. Thus, a simple-to-optimise function that has a small, hidden optimal region added away from the obvious optimal region will become hard to solve. Such a simple problem, and the corresponding hard problem are shown in Figure 2.5.

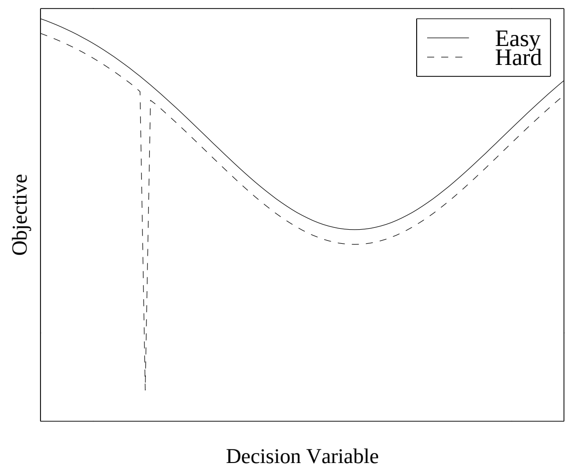


Figure 2.5: A simple problem and a very similar hard problem

Some development is presented as to how fast an EA will be lead away from the true optimum by the simple optimum, and how long one can expect it to take before the optimiser finds the true optimum by chance. Such a problem will be misleading for most optimisers, and is very similar to an easy to optimise problem, but one must again pose the question as to whether it is likely that such a problem will be optimised.

Sharpe's assumptions, particularly 1 and 4 (that points with high fitness are indicators of nearby points with higher fitness; that there are sufficiently many points representing successful solutions that they can be found) would seem to eliminate Droste's hard problems. Unfortunately, though, in this case, only experience with real problems will give a satisfactory, probably qualitative answer.

One of the features of the algorithm developed in this work—dividing the population into groups in decision variable space—makes in reasonably good at solving problems like Droste's hard problems.

2.6 Creating New Points

In a PBA new points are generated somewhere in the search space. If, following the development above, the new points are more desirable than members of the existing population, they will remain in the population (until newer points replace them). A random search will place the new points anywhere in the search space, and after a short time very few of the randomly generated new points will be as good as the best of the population. Ideally then, we would like to do better than choosing random points—to find methods of generating new points that give them a good chance of being desirable.

Sharpe's assumptions give a good starting point—particularly the first (points with high fitness in a region indicate the presence of points with higher fitness) and the second (there are local correlations in the fitness landscape), and algorithms that exploit these assumptions can be designed.

One promising approach is to fit a polynomial approximation of the objective function to the population. This method has been used in what could effectively be referred to as a PBA by Crittin and Bierlaire [22]. In this work, a polynomial surface is interpolated through a set of points, and a new point is generated at the optimal point of the polynomial. Points are generally not removed from the population.

Crittin's approach is an interesting and powerful approach for optimising without derivatives. However, it relies on the objective function being continuous, and demands that it has a reasonable resemblance to the polynomial being fitted, and so cannot cope with discontinuous and multi-modal problems. Furthermore, the method can not cope with integer variables, and critically it does not work for multi-objective problems[10].

Similar approaches have also been used in EAs. Kanacharos et al. [67] present a method that uses the EA to explore the objective function in general, and to use backtracking line search to improve points found by the EA. This technique was shown to be effective on a number of test problems. However, discussions with the authors revealed that they had not used the technique on more than about ten variables (with more decision variables, the population size needed to perform an interpolation becomes very large) and that they had not experimented with multi-objective problems.

A different approach is to create a probabilistic model of the distribution of population, and generate new points randomly from the model. Thus, given a population in the neighbourhood of a local optima, new individuals will also be generated in the neighbourhood of that optima. When the worst individuals are removed from the population, the remaining population will have moved toward, and concentrated around the local optima. This is illustrated in Figure 2.6.

As an aside, note that in an algorithm that generates new points in the region of the existing points, loss of diversity can be seen, as was suggested in §2.4. Whether the population runs out of diversity before it can move all the way to the optima is a question of critical importance.

Diversity issues aside, such an algorithm brings up an issue of critical importance—that of viewing an EA as a problem of *parameter estimation*.

This view, more and more prevalent recently, turns the conventional view of EAs on its head. Not only does it

[10] without considerable extension, which is neither in the scope of this work, nor Crittin's.

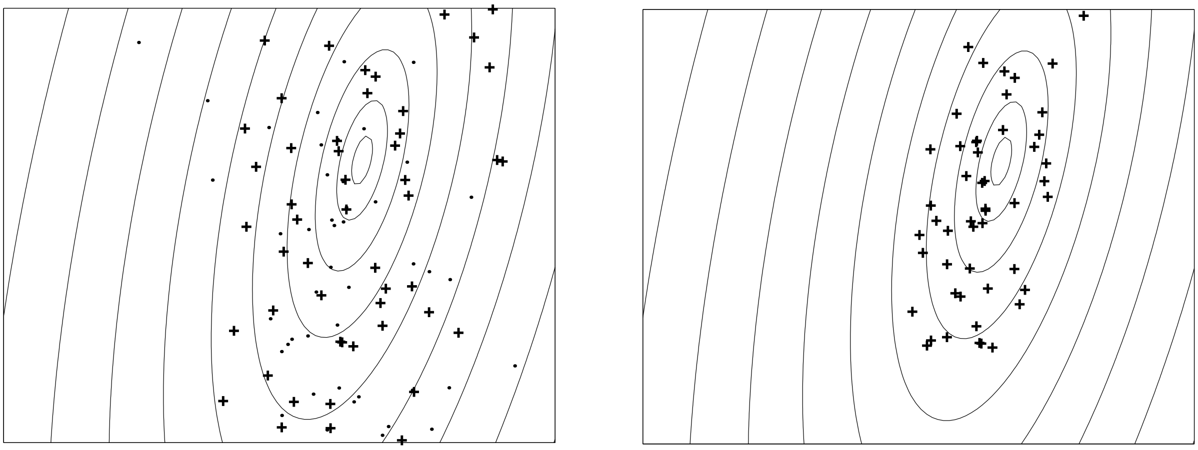


Figure 2.6: An Elitist Algorithm using a Density Model of the Population. Left: best population 'so far' (+) and new population in same region (.). Right: the best of the combined population.

remove much of the 'black magic' of simulating a natural process by putting it on a firm theoretical footing, it also, in the view of this author, gives new insight into the value of the natural process of evolution—a simple and effective way of estimating the density of a population in parameter space.

This approach is discussed in a number of recent publications, which try to analyse conventional EAs using probability theory [132], or present algorithms based directly on probabilistic models of the population distribution. Ocanasek and Schwarz [94] used a Bayes model of the population, while Muhlenbein and Mahnig [92; 93] develop models based on factorised distributions.

An interesting approach to parameter estimation is presented by Thierens and Bosman [127][11]. Thierens develops a conditional factorisation model of the population's structure, a reasonably difficult task, and uses this to generate new individuals, the resulting algorithm has similar performance to more traditional algorithms. Interestingly, Thierens also tried a much simpler model of the population, which, for most of the test functions, gets better results.

This result illustrates one of the rules of thumb presented in the introduction—an EA is essentially heuristic, so there is little point obtaining excessively precise estimates of the structure of the function being optimised. Estimating the density of the population so as to generate new individuals from that distribution is a fundamental aspect of EAs, however, since this *is* a heuristic, if there exists a way to generate individuals in a distribution approximating that of the population without having to estimate the distribution of the population then it would be more than acceptable to use it.

Thus, another possibility for generating new individuals is to take individuals in the existing population, and create points similar to them. Such methods are much simpler than trying to find a probabilistic model of the population, but can give essentially similar results. The most common methods used for generating new individuals are either to take a single individual and make small changes to its parameters—mutation—or to

[11] which presents a review of the field, where a longer list of references in the field is given

take a pair[12] of individuals, and try to combine them—combination.

While these concepts can clearly be seen to have an evolutionary basis, and indeed are inspired from evolution, it is the opinion of this author that they should not be viewed this way. Instead mutation and crossover should be viewed as simple methods of achieving that which an distribution estimation algorithm attempts to do: to generate new individuals that are similar to the existing population.

2.7 Operators

In general, the combination operator used by an EA is either specified in the algorithm definition, or chosen by the user before the optimisation starts. Often, combination is used in the generation of *every* new individual. Mutation, however, is often treated differently. In most cases, there is only a low probability that mutation will be used in the generation of each individual, this probability being chosen by the user. In some cases there will be more than one mutation operator available, and the user will assign probabilities to the use of each one. This is often the case where some problem-specific knowledge is incorporated into the solution process—a special mutation operator which favours the creation of what the user knows are likely to be good solutions is used.

The following discusses a number of significant combination operators in detail. The operators are those used in the algorithm developed in this work. One of the operators mentioned, simulated binary crossover, was tested with the algorithm, but does not appear in the results—its inclusion would have increased the already large computation time to generate test results.

Mutation operators are not discussed. The three mutation operators used in this work are introduced in Chapter 4.

2.7.1 Real Variable Uniform Crossover

This is the combination operator generally used in Evolution Strategies (ES) [117, 8], termed ‘discrete combination’ in the taxonomy given in Bäck [5]. A ‘child’ may take a value a particular parameter from either of its two parents. Children are thus placed at the vertexes of the hypercube described by the two parents.

Parent 1	**0.43**	0.98	0.28	**0.12**	0.03	**0.63**
Parent 2	0.48	**0.24**	**0.86**	0.35	**0.21**	0.57
Child	0.43	0.24	0.86	0.12	0.21	0.63

Uniform crossover works with integer variables without modification.

[12]Or more.

2.7.2 Blend Crossover—BLX–α

Blend crossover was first proposed by Eschelman and Schaffer [42], and is generally used in real-valued evolutionary algorithms. Deb [31] refers to it with respect to multi-objective problems. Briefly, blend crossover takes two parents and creates a new individual in a hypercube surrounding the two parents. How much larger that hypercube is than the hypercube *defined* by the two parents is controlled by the parameter α. Thus if $\boldsymbol{x}$ and $\boldsymbol{y}$ are two 'parent' vectors, their mean is $\bar{\boldsymbol{x}} = \frac{\boldsymbol{x}+\boldsymbol{y}}{2}$, and the distance between them is $\boldsymbol{d} = \boldsymbol{x}-\boldsymbol{y}$. The limits in which we can produce a child are thus $\boldsymbol{x}_l = \bar{\boldsymbol{x}} - (0.5+\alpha)\boldsymbol{d}$ and $\boldsymbol{x}_r = \bar{\boldsymbol{x}} + (0.5+\alpha)\boldsymbol{d}$. If there are no limits on $\boldsymbol{x}$, a child can be created without further consideration, however, if there are limits (as there usually are) $\boldsymbol{x}_l$ and $\boldsymbol{x}_r$ must be checked to assure they do not exceed those limits, giving $\boldsymbol{x'}_l$ and $\boldsymbol{x'}_r$. Then, given $\boldsymbol{\mu}$, a vector of random numbers between 0 and 1, we can generate a new individual:

$$x_n^i = (1-\mu^i)x_l'^i + \mu^i x_r'^i$$

As recommended by the original authors, in this work $\alpha = 0.5$ is used.

The blend crossover operator can be modified to deal with integer variables. Here, the operator chooses an integer between the limits defined by the two parent values and α. It must be ensured that the probabilities of selecting the 'limit values' are the same as all the other values, and in cases where there is little mutation, it must be ensured that even if the parents are very close or the same (which is far more likely with integer variables than real-valued variables) that there is a possibility to select a value 'outside' the range defined by the parents. (i.e. it is best to round the low limit down, and the high limit up).

2.7.3 Linear Crossover

Linear crossover is termed 'generalised intermediate combination' in the taxonomy given in Bäck [5]. This is a real variable operator, similar to blend crossover, except that the child is placed on the line between the two parents, rather in the hypercube (i.e. $\forall i\ \mu^i = \mu$).

The use of the linear crossover operator for multi-objective optimisation is motivated by the observation in Fonseca and Fleming [51], that solution sets to multi-objective problems will quite likely lie along lines in decision variable space, and further, and that the progression of many EAs along the ridges will be slow.

The linear crossover operator encourages exploration along these lines. However, as will be seen in later chapters, in most problems, growing the set of optimal solutions is often more complicated than searching along straight lines in decision variable space.

2.7.4 Simulated Binary Crossover—SBX

Deb and Agrawal [32] introduce the simulated binary crossover operator for real variable crossover. As its name suggests, the operator simulates the effect of the binary single crossover operator on real variables—that is, the operator gives similar results to those that would be given if the parents were binary encoded, and binary single crossover was performed. Given $\boldsymbol{\mu}$, a vector of random numbers between 0 and 1, a child of two parents $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined by:

$$\beta = \begin{cases} 2\mu^{\frac{1}{\eta_c+1}} & \text{if} \quad \mu < 0.5; \\ \left(\frac{1}{2(1-\mu)}\right)^{\frac{1}{\eta_c+1}} & \text{otherwise.} \end{cases} \tag{2.3}$$

$$x_i' = \frac{1}{2}\left((1+\beta)x_i + (1-\beta)y_i\right) \tag{2.4}$$

The probability distribution of of children from this operator is shown in Figure 2.7 for two values of η_c. The children are most likely to be near one of the two parents, and the spread of the distribution is determined by the distance between the parents and η_c.

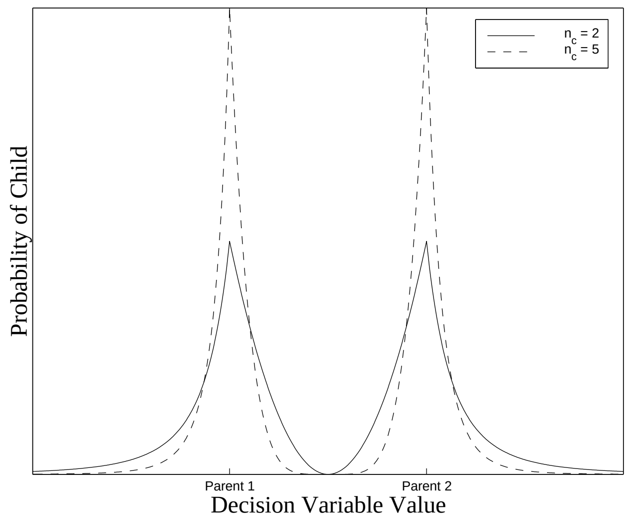


Figure 2.7: Probability density function of children from SBX.

The critical difference between SBX and binary crossover is that operator is not dependent on the precision of the binary coding for determining the spread of the distribution, this instead being controlled by η_c. Deb and Beyer [34] show that algorithms using SBX exhibit ‘self-adapting’ behaviour, much like the evolving mutation radii used in ES.

2.8 Parent Selection

Non-elitist algorithms must rely on methods other than elitism to push the population toward an optimum. As the best individuals are not selected to remain in the population, enabling it to drift toward the optimum in a slow, but guaranteed manner, attention must be paid to how new points are generated so that there is some hope that a subsequent generation will be better than the current one. Not only can these methods be used to aid algorithms lacking elitism, they can also be used to improve the performance of elitist algorithms[13].

The algorithms developed above improved their performance by generating a new population that resembled the current one, and then selecting the best of the two populations. When there is no elitism, stronger measures must be taken, or no convergence will occur, as illustrated in Figure 2.8. In order that new population taken on its own be better than the existing population, it must resemble the better members of the existing population.

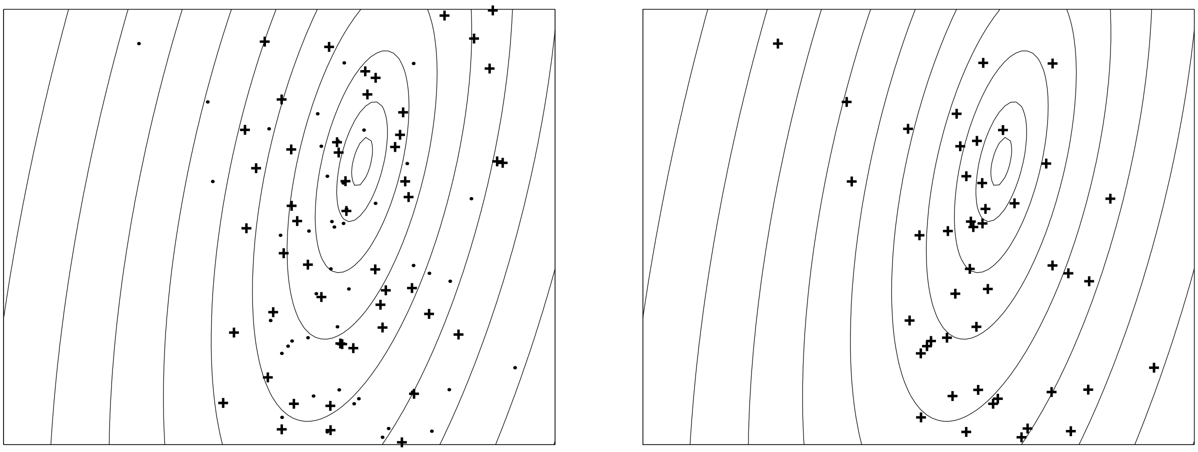


Figure 2.8: A non-elitist algorithm using a density model of the population. There is no clear convergence toward the optimum.

Here, some of the 'convenience' of evolutionary operators, as opposed to estimated distributions is illustrated.

Many of the simpler distribution estimation algorithms result in a model of the existing population distribution that takes no account of the desirability of each individual, so when generating a new individual from the model, there is no way to choose a 'better region'. Bayesian models can be constructed that also include a prediction of the desirability of each region, and from these, individuals in 'better' regions can be preferred. However, this requires some work. Critically, for the moment there is no clear method for applying such a model to a multi-objective problem, either because of the problems associated with using rank as a measure of desirability (§3.7.4) or because there of problems choosing between the two objectives [91].

With evolutionary operators, however, preferring better parents is simple: one does just that. The best known method is the GA's roulette wheel selection, but a number of other methods, such as 'stochastic remainder proportionate selection' [124], and 'restricted tournament selection' [64], have been used.

When applied to a non-elitist algorithm, these methods allow convergence toward an optimum, as illustrated in Figure 2.9.

When applied to an elitist algorithm, parent selection works even better (Figure 2.10), almost to the point of causing problems with diversity preservation.

[13]Perhaps proofs will also be found for such algorithms, also offering guaranteed convergence, results describing performance, or even that apply to continuous optimisation.

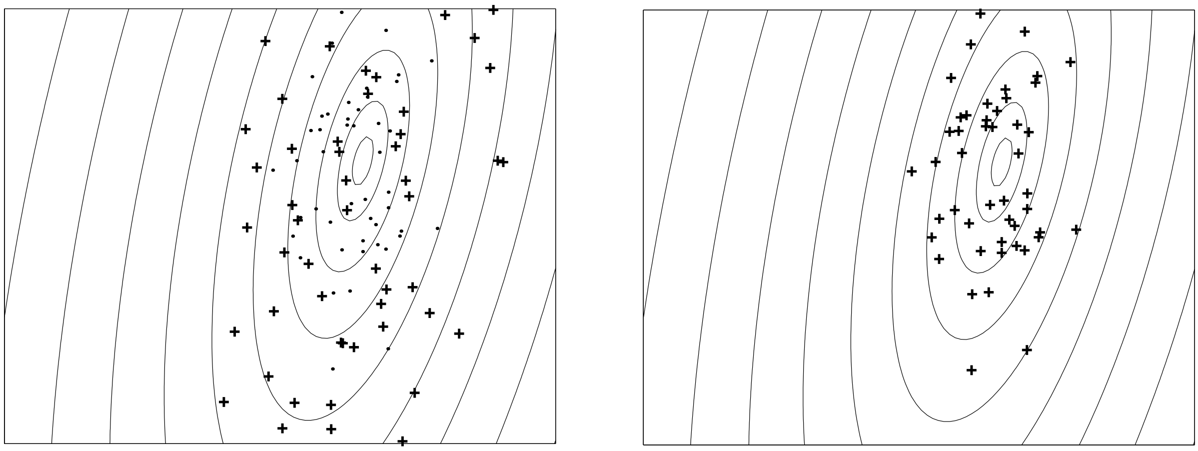


Figure 2.9: A non-elitist algorithm using parent selection. There is some convergence toward the optimum.

In single objective optimisation, there will be a large difference between the best and worst individuals, and so it will be worthwhile making an effort to prefer in the best individuals. In multi-objective optimisation however, it is less easy to distinguish between better and worse individuals, and often the entire population will be 'best', making parent selection less effective.

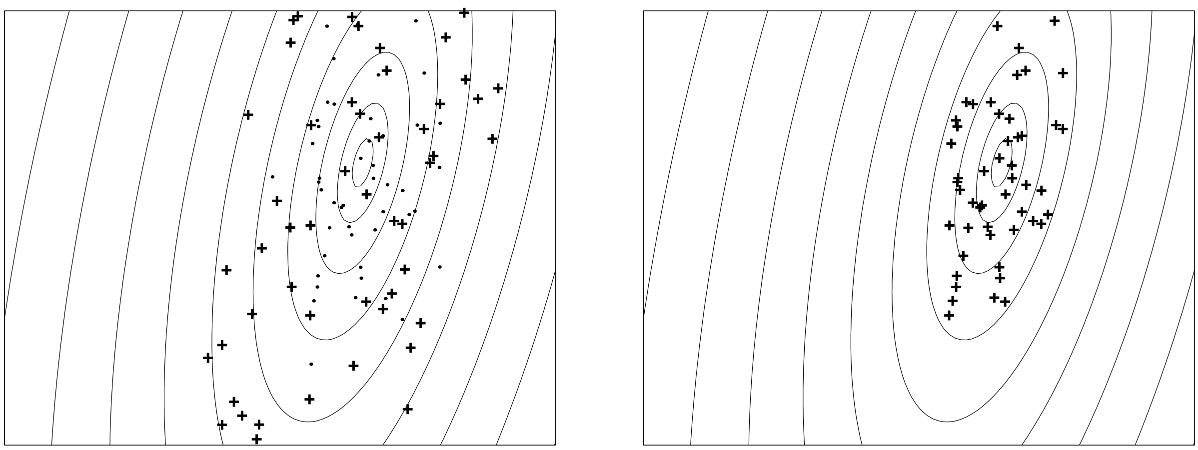


Figure 2.10: An elitist algorithm using parent selection—guaranteed, and perhaps reasonably rapid convergence to the optima.

2.9 Preservation of Diversity

Preservation of the diversity of a population in decision variable space is necessary to assure robust convergence in all but the most simple of optimisation problems. Techniques for the preservation of diversity have thus been proposed since very early in the development of EAs, and range from the stochastic selection of the next generation (as in EP), to the modification of the fitness function in order penalise individuals with many close neighbours.

When trying to preserve diversity, it is important to remember than diversity in itself is not the goal. As Spears [123] says:

> The general consensus, then, is that diversity for the sake of diversity is not the issue. Rather it is the *appropriate* use of diversity, to explore new areas while not destroying the information already learnt.

Diversity is preserved with a clear goal—to ensure that the decision variable space is searched thoroughly. Because exactly how to attain this goal is not always clear, methods used for diversity preservation are necessarily general.

It is worth noting that there are some conceptual differences between 'pure' diversity preservation, and diversity preservation for multi-modal optimisation. In the first case, it is assumed that there is only one interesting local optimum (the global optimum), and perhaps only even one local optimum. Diversity is preserved either to assure that the right local optimum is found, or, in the case of only one optimum, to ensure that the population does too converge to quickly so that it can continue to move toward the optima[14].

In the second case, multi-modal optimisation, there are several local optima, and ideally all of them should be found. This means that the diversity preservation scheme must not only preserve diversity, but a number of subpopulations, each one in the region of a local optima.

2.9.1 Parent Replacement

Goldberg [57] cites Cavicchio [15] as one of the earliest researchers to address diversity problems. In Cavicchio's work, a variant of a GA, a child would replace the worst of its two parents if it was better than that parent. This relies on the (quite reasonable) assumption that a child will closely resemble its parents, and therefore preserving at least one of the family will preserve the neighbourhood of the family in the space covered by the population. It is interesting to note that as early as 1970 Cavicchio's work added elitism to the GA and so, given strong enough mutation, guaranteed convergence[15]. Mafhoud [83] proposed a similar mechanism, deterministic crowding, where each 'couple' had two children, and each child competed with its closest parent.

2.9.2 Replacement of Nearby Individuals

De Jong [29] introduced the concept of *crowding* (again to a GA). Here, each new individual is compared to a randomly chosen subset of the population to find its nearest neighbour in the subset. As the algorithm in question was a binary coded GA, distance was measured as the number bits that differed between the two

[14]EA populations generally move across a search space at a speed proportional to the span of the population. Thus if a population converges very quickly to a small region, it will only be able to move slowly to the global optima, effectively getting stuck 'halfway down the hill'.

[15]and even more interesting to note that Goldberg doesn't remark upon the fact.

individuals. The new individual then replaced the nearest individual—without checking whether the new point was better than the existing one. This method was proposed as an 'improvement' to Cavicchio's work, but unfortunately, removed the elitism inherent in his method. Mafhoud [83] showed that though such a crowding maintained diversity, it was of little utility for solving multi-modal problems.

2.9.3 Sharing

Goldberg and Richardson [58] proposed a method called 'sharing' to preserve diversity, which has since been widely used in GAs. Here, in order to prefer individuals that 'inhabit' relatively sparsely populated regions of the decision variable space, the fitness values of individuals with many near neighbours are degraded. Distance between individuals is measured in decision variable space, rather than as a bit-difference count (though Goldberg suggested this could be used). For sharing, a sharing function $s(d)$ is defined for $d(\boldsymbol{x}_a, \boldsymbol{x}_b)$—the distance between two individuals, that specifies a penalty function for having nearby neighbours. The fitness of an individual $\boldsymbol{x}_a$ is degraded using the sum of all the sharing function values for the distances between $\boldsymbol{x}_a$ and all other points in the population[16]:

$$f_s(\boldsymbol{x}_i) = \frac{f(\boldsymbol{x}_a)}{\sum_{b=1}^{p} s(d(\boldsymbol{x}_a, \boldsymbol{x}_b))} \tag{2.5}$$

The sharing function is often a linear function of distance, taking a value of 1 when d is zero, and taking a value of zero for $d \geq \sigma_{share}$. Estimating σ_{share} however, can be difficult. Deb and Goldberg [35] proposed a method for automatically setting the parameter σ_{share}, but later investigations showed that the methods proposed were not sufficient to solve 'hard' multi-modal problems.

2.9.4 Labelling

Spears [123] proposed a 'labelling' algorithm to preserve separate populations. Spears found that the problem of calculating distances between individuals required by many of the above methods was to expensive (and subject to the 'curse of dimensionality', see below), and proposed identifying similar individuals by labels. Each member of the initial population is given a randomly chosen label (a number of binary tag bits—this was, after all, a GA). Parent pairs were then only chosen from individuals with the same tag, and the children inherited the same label, occasionally changed by mutation. Thus, there are effectively several sub-populations which evolve semi-independently, (still competing with each other for chances to produce children). By combining the labelling approach with a form of sharing, Spears was able to obtain results of a similar quality to those of GAs using sharing, for a much lower computational cost.

2.9.5 Restricted Tournament Selection

Harik [61] proposed 'Restricted Tournament Selection' as an improvement to crowding and to Mafhoud's work, which is intended to not only preserve diversity, but also solve multi-modal problems. Here, for each

[16] As in Goldberg's equation, the sum does *not* exclude the index a. I do not know whether this is intentional on Goldberg's part.

new individual, a number of members of the population are chosen at random, and the individual *closest* to the new point *competed* with the new point for inclusion in the next population. Thus, elitism is added to the crowding algorithm, and the assumption from Cavicchio's work that a parent is near its child is no longer necessary. This assumption, however, comes at the cost of calculating a distance metric between individuals.

2.9.6 The Struggle GA

The Struggle GA [60, 120], an 'ancestor' of the algorithm developed here, uses a variant of restricted tournament selection, where *all* the population is checked to see which individual is the closest to that being introduced. Like restricted tournament selection, the SGA's method not only preserves diversity, but can find several local optima.

2.9.7 Distance Measures

Most methods for the preservation of diversity (with the exception of those that assume that the parents of a new point are close) need a measure of the distance between two individuals. This can prove problematic. In some cases, the Euclidean distance between to individuals in decision variable space can be used. Often, however, a scaling must be defined on the space, so that the differences, between say, two turbines of several hundreds of kilowatts, and two design tolerances of a few thousandths of a metre, are roughly the same. Often, scaling information will be available in the limits of the search domain for each variable.

For some kinds of variables, though, this approach does not work at all. Not all integer variables, for example, are selecting a discrete number of something: some are representing design choices. So while it is reasonable to say that a plant that uses three furnaces is more like a two furnace plant than a plant with five, not much can be said about the relationship between plants using a gas turbine to supply power, those using an internal combustion engine or those buying electricity off the grid.

Another problem comes from the so-called *curse of dimensionality*. While it is reasonable to measure distances between individuals in two or three dimensions, as the number of dimensions rises, the distances between individuals are found to all be about the same—an individual that is distant in one dimension will probably be close in another. Thus, though a nearest neighbour will always be found using a Euclidean distance, it may not really reflect the similarities of decisions between one individual and another.

This means that in most problems of reasonable complexity, diversity preservation requires some knowledge of the decision variable space, which must be embodied in a distance function. The distance function may be a subset of the decision variables over which to calculate Euclidean distance (the subset might even include the objectives), or it might be a set of weights for each of the decision variables (and objectives). If foreknowledge of the problem space is detailed, it could even include non-linear functions of the decision variables.

Methods do exist [87] for automatic characterisation of the decision variable space, and these can potentially find appropriate weights for decision variables for a weighted Euclidean distance. However, the author is not aware of any published work in this domain, and the subject is not treated in this work.

Binary coded EAs[17] can use another simple, though not necessarily appropriate, method for measuring the distance between individuals. The *Hamming distance* between two binary strings is defined as the number of bit positions where the two strings differ, and so offers an easy measure of the distance between two binary-coded individuals. The measure suffers from all the problems normally associated with binary encoding, though. For example, individuals that differ in only the *most* significant bit of a binary-coded variable will be considered to be the same distance apart as variables that only differ in the *least* significant bit. This is not appropriate for most real-variable optimisation.

2.10 Parallel Evolutionary Algorithms

For the majority engineering problems, most of the computational effort spent by an EA will be calculating the objective function values for each individual. The elapsed time taken to perform an optimisation can be reduced by spreading these calculations over a number of processors—performing the objective function evaluations in parallel.

In a generational EA, where, say, 100 new individuals are generated and then evaluated at each iteration, it is clear where and how parallelism can be implemented. Given M individuals to evaluate and N similar processors (or computers) on which to evaluate them, we send M/N individuals to each processor and wait for the objective function values to return, before continuing with the selection of the next generation.

This can be considered one of the major advantages of EAs in terms of computation time. Although an EA will generally take many more function evaluations to solve an optimisation problem than a more analytic method (if such a method is available), a parallel EA, given sufficient resources, may take less *elapsed* time to solve the problem. Most analytical methods, on the other hand, follow a single point around the parameter space, and require that point t be evaluated before it can be decided where point $t + 1$ will lie, and thus it is difficult to parallelise the objective function evaluations.

Parallelising objective functions not the only means of parallelising an EA. Gordon and Whitley [59] also reviews 'island model' parallel EAs, where each processor runs it own, essentially independent EA, with some interbreeding between processors, and 'cellular model' EAs where each individual runs on its own processor, and only interacts with individuals that are 'nearby' in the network topology. However, only parallelisation of objective function evaluations is treated.

Levine et al. [76] describe PGAPack, a parallel binary GA implementation using MPI for communication between processors. Function evaluations are sent to 'slave' processors while the rest of the GA machinery runs on a 'master' processor. PGAPack can be run in either generational or steady-state modes, but does not run continuously—a number of individuals are generated, and then evaluated, with pauses between, as the 'GA work' (parent selection and generation of new individuals) is performed.

Bäck et al. [6] parallelise ES using PVM [125] and a steady-structure but the algorithm retains the function evaluation—EA work model, and so the remote processors stop for short periods as in PGAPack.

[17] I know I said I wouldn't mention them, but it happened

Cantù-Paz [13], reviewing parallel EA architectures, briefly makes the distinction between *synchronous* and *asynchronous* (where processing of objective functions does not stop of the EA WORK) EAs, but goes on to say that most parallel EAs are synchronous, and considers asynchronous algorithms no further in his analysis.

The problem with a synchronous parallel EA is that at the end of each generation, *all* the computers in the network, except the master, stop while the master performs selection generates the next generation. This is not a large loss if communication latencies between the processors are low, and generation of a new population does not take much work.

The second problem, is that, if either calculation of the objective function takes a variable time (as often the case in simulation problems) or all the computers do not run at the same speed, much time can be wasted at the end of each generation.

The first problem is worse in multi-objective problems than single objective ones. As will be seen in the next chapter, selection and creation of a new generation in multi-objective problems implies finding dominance relations for the entire population, and then ranking it. This process can take some time, during which the other processors rest idle.

The algorithm developed in this thesis, described in Chapter 4, is asynchronous, and overcomes both of these problems.

2.11 Summary

Evolutionary algorithms are used on energy system-type optimisation problems (and a wide range of other problems) because such problems pose a number of problems for more conventional optimisation methods. The problems are generally non-linear, discontinuous—even disjoint, and have several local optima. To complicate the situation further, the decision-maker using the algorithm probably wants more than just one optimal solution to a problem, instead preferring a range of solutions that describe the regions of several interesting local optima.

An advantage of using an EA for optimisation is that it is relatively easy to ‘prepare’ a model for optimisation, whereas some other methods may require posing the model in a particular form. If the problem is to be solved only once, the increased elapsed time for optimisation may well be payed for in a reduced problem preparation time. Similarly, EAs have the advantage that they will, given enough time, solve almost any optimisation problem, meaning that changes in the complexity of the model will not necessitate changes of the optimisation algorithm. This, however, is something of a two-edged sword—decision-makers must always ensure that they are using an appropriate optimisation technique, and not find themselves using EAs to solve linear problems.

Once it has been decided to solve a problem with an EA, some ground rules can be set about what can be expected of the function to be optimised, and how to go about optimising it. On one hand, as the problem is assumed to be impractical to optimise by more conventional methods, there is no advantage to be gained from too precise estimations of the function's global properties—as illustrated by Thierens and Bosman [127]. On the other hand, the problem is being optimised, and so it is not completely random, and some assumptions (noted by Sharpe [122]) can be made, notably, that points in the search space with high fitness indicate that points of even higher fitness can be found in their neighbourhood.

By investigating the behaviour of a very simple algorithm—the population-based algorithm—considerable insight can be gained into how EAs work. Even without specifying how new points are added to the population, two convergence proofs can be derived. Both proofs apply only in the domain of discrete problems—which covers most optimisation performed on computers—but which makes them quite different from continuous optimisation proofs. The first presented shows that, given a sufficiently strong mutation operator (implying that diversity must be preserved), and elitism, that a PBA is guaranteed to converge to the optimal set in finite time. The second shows that over all possible optimisation problems, the average performance of any optimisation algorithm can be no better than random search. However, the size of the 'all possible optimisation problems' space is so much larger than 'problems likely to be optimised', that the proof is largely irrelevant. It does, however, teach common sense—it is very difficult to construct an algorithm that works well on a wide range of optimisation problems.

The proofs are developed very early in the description of the PBA, and as such, they are essentially proofs about the performance of random search. EAs, by taking advantage of Sharpe's assumptions, can probably[18] do better on functions where these assumptions hold.

One way of taking advantage of these assumptions is to generate the new points in the PBA in a manner that makes them resemble the existing population. This method relies on the algorithm being elitist—it is elitism that makes the existing population crowd around the the function's optima. Generating a probability model of the population and choosing new individuals from the resulting distribution is a rigorous method of implementing this strategy. However, building the probability models is complex, and 'extra detail' provided by such a model is of little value.

Using crossover and mutation operators that mimic evolutionary processes proves to be a simple, effective and flexible method of generating a new population that resembles the existing one.

This description of an EA turns conventional wisdom on its head. An EA is viewed as a distribution estimation problem, which uses evolutionary processes as a cheap way of performing the estimation. It is *not*a simulation of evolution—rather, evolution is a simulation of a population-based algorithm[19].

As an aside, it is shown that elitism is of central importance to the performance of an EA, and that parent selection is of less consequence, despite the central role it plays in some earlier algorithms.

Preservation of diversity is a significant problem for EAs—not so much caused by the algorithms themselves, but more a symptom of the problems which are solved with EAs. Convergence and diversity preservation

[18]no proofs have been developed yet

[19]However, natural evolution is a little different from the optimisations performed here, in that there is no fixed objective function to optimised. It is more of an optimisation in a continuously changing environment. In this case, where an individual's 'objective function value' or fitness (or more critically, rank) can change relative to others with time, elitism is of considerably less importance, as a once-best individual will soon be out of date, not because better individuals now exist, but because the definition of best has changed.

are two somewhat conflicting goals—preserving diversity means concentrating less on the best individuals found, and slowing convergence. Methods used to preserve diversity must try, in some measure, to balance these two goals.

A number of times in this chapter, multi-objective optimisation has been mentioned in passing, without too many specifics. Multi-objectives poses even more problems for optimisation algorithms, and EAs are very suited to overcome these problems. The next chapter describes multi-objective optimisation in more detail, and shows how EA techniques can be applied to solving such problems.

Chapter 3

Multi-Objective Evolutionary Algorithms

3.1 Introduction

Many real-world optimisation problems have several objectives. A manufacturer will have a requirement that a product can be produced cheaply, but may also have requirements about quality levels or the product's life-span. In the energy systems domain, the problem is classic: plant costs—construction, maintenance and running costs—must be as low as possible, but the plant must have a minimal environmental impact.

Often, the objectives for a design problem will be conflicting—a cheap industrial plant will quite likely be highly polluting, while emissions abatement equipment is costly—and so no single optimum can be found. This is the domain of multi-objective optimisation (MOO)—how to weigh up conflicting objectives, how to assure that minimal requirements are met, and, in the ideal case, how to illustrate the trade-offs between objectives, and find solutions that cannot be unequivocally improved.

Unfortunately, multi-objective optimisation is difficult, and many methods have been invented in order to 'dress up' multi-objective problems as more tractable single-objective problems. On investigation it is found that many apparently single objective problems are better solved as multi-objective problems.

For example, the Shanxi Coke Industry problem (presented in §6.1) has two objectives that are entirely monetary—plant construction costs and coke transport costs. However, transport costs are per unit of production while construction costs are a one-time cost that must be paid off over several years. The comparison of these two costs requires calculating the future value of the plant construction costs over the life time of the plant, and the trends of these costs oppose each other (in this case the more expensive construction options lead to lower transport costs). As the study was for a local government, and performed at a very early stage in planning, both the plant life times (or payback times) and interest rates are ill-defined. Furthermore, since the study was for a government institution, one cannot discount the possibility that interest rates can be controlled, and that tax breaks and interest free loans can be given in order to encourage the kind of development the state prefers. Indeed, in cases like this, the question may not be of how to build the most economic plant, but of what regulatory changes are necessary to induce coke producers to work in the manner the state wishes.

In this example, three advantages of multi-objective optimisation can be seen:

- a multi-objective optimisation will illustrate the trade-offs between opposing objectives, even when those objectives could be combined;
- a multi-objective optimisation will reduce the number of uncertain variables (interest rates or plant lifetimes in this case) that are involved in the optimisation;
- a multi-objective optimisation will respond better to unforeseen questions even after the optimisation has been performed—in the case the regulatory questions.

Clearly, methods for ‘real’ multi-objective optimisation would prove useful in a wide range of situations.

3.1.1 Definition of a Multi-Objective Optimisation Problem

Before continuing any further, multi-objective optimisation problem (MOP) will be formally (and informally) defined. For the formal part, a definition is taken from Van Veldhuizen and Lamont [131]:

> In mathematical terms, a MOP minimizes (or maximizes) the components of a vector $\boldsymbol{f}(\boldsymbol{x})$ where $\boldsymbol{x}$ is an n-dimensional decision variable vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ from some universe $\mathcal{S}$. Or, in general,
>
> $$\text{minimize} \quad \boldsymbol{f}(\boldsymbol{x}) \quad = \quad (f_1(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})) \tag{3.1}$$
>
> $$\text{subject to} \quad g_i(\boldsymbol{x}) \quad \leq \quad 0, i = 1, \ldots, q, \quad \boldsymbol{x} \in \mathcal{S}. \tag{3.2}$$
>
> A MOP thus consists of n variables, q constraints and m objectives, if which any or all of the objective functions may be linear or non-linear. The MOP's evaluation function $\boldsymbol{F} : \mathcal{S} \rightarrow \mathcal{F}$, maps the decision variables to vectors.

Thus, given an optimisation problem with n parameters, we wish find a way, or preferably a number of ways, of adjusting the parameters $\boldsymbol{x}$ such that values of the objective functions are acceptable, as good as possible, or in some sense, optimal. If just one solution is found to the MOP, and its performance is found to be acceptable, then the criteria have been fulfilled. Ideally, however, a number of solutions will be found, and these will illustrate the trade-offs between the objectives[1].

3.2 Early attempts at Multi-Objective Optimisation

A number of methods exist for tackling MOPs. That which is probably the best, and which this work addresses—Pareto-optimisation—is presented in the next section. Here, a number of other approaches are reviewed, in order to present the history of MOO, and to illustrate some of the pitfalls of these methods.

[1]There will be, of course, some MOPs where the objectives are non-conflicting, and there really is only one optimal solution.

3.2.1 Weighting and Environomics

A common method for solving multi-objective problems is to transform them into a single-objective problem by weighting the objectives relative to some measure of their importance. (3.1) becomes:

$$\text{minimise} \quad F(\boldsymbol{x}) = \boldsymbol{w}^T \left(f_1(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}) \right), \tag{3.3}$$

where $\boldsymbol{w}$ is a vector of weights for each of the objectives. The MOP is thus transformed into the optimisation of the length of a vector in objective space, its direction defined by $\boldsymbol{w}$ (Figure 3.1, which shows an imaginary trade-off between cost and quality). Often, the elements of $\boldsymbol{w}$ have a physical interpretation.

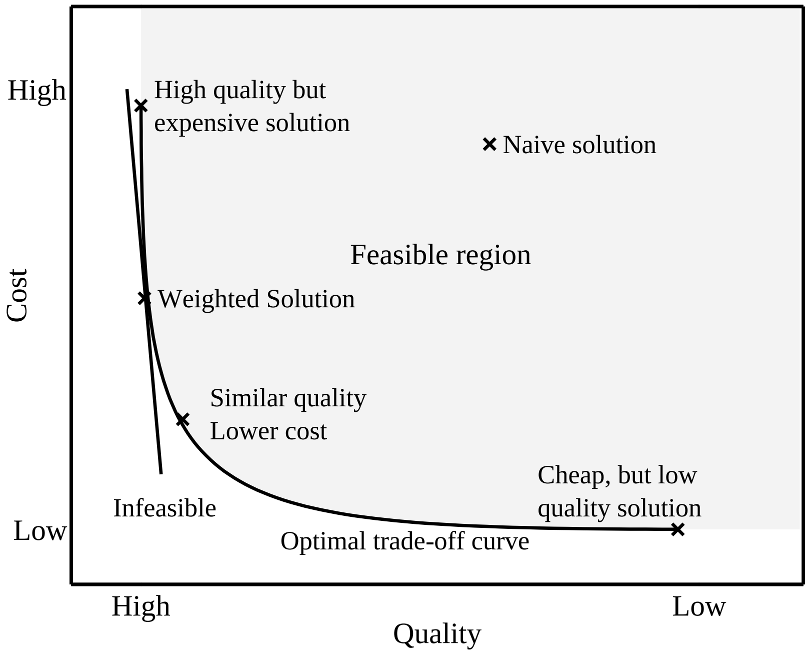


Figure 3.1: Optimising with Weighted Sums. Weighting finds that part of the optimal trade-off curve that is tangent to the plane defined by the weight vector). Several other points of interest on the trade-off curve are marked.

Environomics is a method for taking a two objective problem—the economic costs of a plant and the pollutants it emits—where the element of $\boldsymbol{w}$ corresponding to the pollutant objective is a 'cost of pollution'. Unfortunately the cost of pollution is poorly known, and any number of methods can be used for deriving a value for it. Curti et al. [24] derive a value based on existing levels of the pollutant in the environment, and on an estimation of the level of pollutant that the environment can tolerate. From this, a single objective optimisation is performed. Note that in Curti's work, $\boldsymbol{w}$ was a non-linear function of $\boldsymbol{x}$—a non-linear function of the amount of pollutant produced. Nothing implies that $\boldsymbol{w}$ must be constant.

Pelster et al. [102] uses a similar approach, imposing a cost of $\mathrm{CO_2}$, and adding the cost of $\mathrm{CO_2}$ emitted to the total cost of an installation. Pelster goes on to experiment with the value of the cost of $\mathrm{CO_2}$, in order to see at what level it becomes economic to install emissions abatement equipment. Olsommer et al. [97; 98] used a very similar approach for optimising a waste incineration plant.

Two problems associated with the weighting approach are clear from the examples: poorly defined factors must be included in the optimisation, reducing the value of its results, and only one solution to the problem is produced for each optimisation run—investigation of the solution space requires multiple, computationally expensive (as was the case in both Pelster's and Olsommer's work) optimisation runs.

A third problem is perhaps more insidious: a change in the manner in which the problem is viewed. Ideally, for both problems, the final product of a study would be a trade-off curve between plant cost and pollution, showing all the options. Instead, this trade-off curve will be presented in terms of a fictional (and confusing) factor, and this, along with the paucity of information available due to the difficulty of running the optimisation multiple times, will not give as clear a picture of the situation as possible.

Molyneaux [89] presents an in-depth comparison of the environomic approach used in Curti's work, and a multi-objective approach used to solve the same design problem. The study concludes that multi-objective optimisation provides more information about the system being studied, in less time.

3.2.2 Constraints

An alternative approach to the 'aggregation' described above is to transform some of the objectives into constraints. Here, the problem is optimised for just one of the objective, subject to the constraint that the other objectives must be better than some minimum requirements c_k. (3.1) thus becomes two equations:

$$\text{minimise} \quad f_1(\boldsymbol{x}) \tag{3.4}$$

$$\text{subject to} \quad f_k(\boldsymbol{x}) \leq c_k, k = 2, \ldots, m, \quad \boldsymbol{x} \in \mathcal{S} \tag{3.5}$$

This approach (illustrated in Figure 3.2) suffers from the same problems as aggregation by weighted sum: little known factors are introduced into the optimisation (in this case the constraint levels), and the optimisation finds only one optimal point per run. There is, however, another disadvantage. Because of the nature of constrained optimisation, combined with the probable shape of the trade-off curve, the optimal point found will always lie on the constraint. If the trade-off curve has a sharp corner (as in Figure 3.2) and the constraint does *not* lie near the corner, there is a risk of either finding solution that emits up to the limit, while cleaner solutions would only cost a little more, or finding a very expensive solution that fulfils the constraint, when a cheaper solution would only pollute a little more. Weighting methods suffer much less from this problem, as illustrated in Figure 3.1.

3.2.3 Acceptability Functions

Another method of reducing a problem to a single objective, which combines both the aggregative and constraint approaches, is to optimise a product of the objectives, or more generally, the product of non-

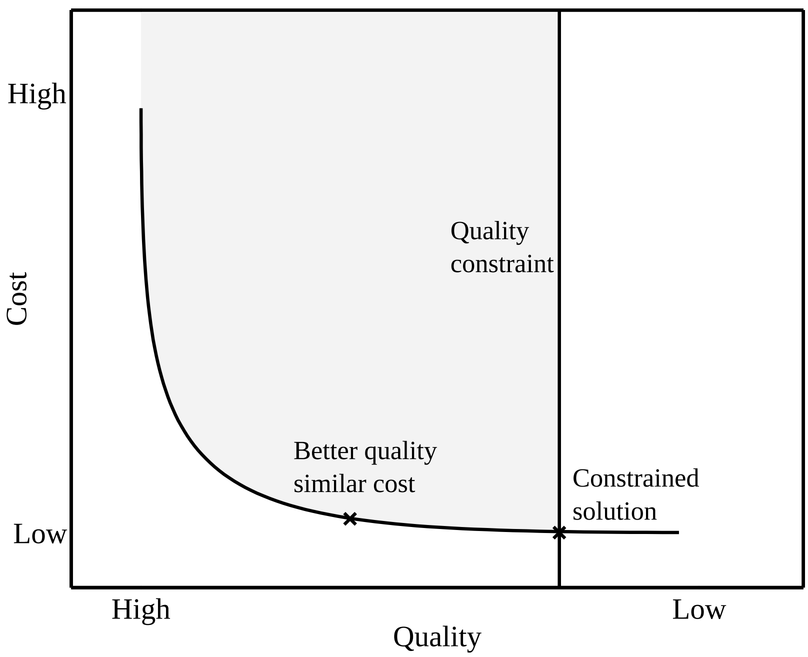


Figure 3.2: Optimisation with Constraints. Correct choice of constraint levels is even more critical than choosing weights.

linear functions of the objectives Kim and Wallace [68]. (3.1) thus becomes:

$$\text{minimise} \quad F(\boldsymbol{x}) = \prod_{k=1}^{m} a_k(f_k(\boldsymbol{x})), \tag{3.6}$$

where each a_k is a normalised 'acceptability function'. The acceptability functions can thus represent constraints on the values of objectives, reflect their value, or even indicate values above which there is no point improving that objective. Often the a_is are piecewise linear, between an unacceptable value, and a value that is completely acceptable, as illustrated in Figure 3.3

Acceptability functions again present the same problems as weighting, though it could be argued that an acceptability function allows for a better representation of poorly-known weights, or that it is a method of implementing 'soft constraints'. However, unlike the two methods presented earlier, acceptability function approaches offer no guarantee that the solution found by the optimiser lies on the 'optimal' trade-off curve. Though the solution found will do a good job of satisfying the acceptability functions, it is possible that there exist solutions that are both cheaper and cleaner than the solution found using acceptability functions.

3.2.4 Multi-Objective Optimisation and Parameter Studies

Often, when poorly-defined weights or constraints are used in a single-objective optimisation, a range of values for the parameters are used. In these 'parameter' studies, the cost of pollution may be varied to see its effect on the optimal system design, or pollution limits or budgetary constraints may be varied on order to

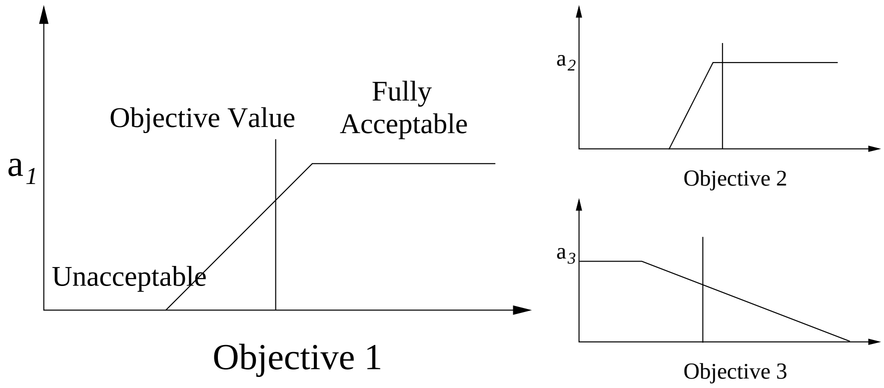


Figure 3.3: Acceptability Functions. The objective is the product of several piece-wise linear acceptability functions. The acceptability functions have regions indicating unacceptable objective values, a region where the objective becomes more acceptable, and an objective level that entirely sufficient.

see what effect the have on other objectives. Thus, parameter studies can provide information about optimal solutions over a range of conditions.

Parameter studies and multi-objective optimisation are entirely equivalent. For any parameter study, a multi-objective problem can be found that will provide the same information. Equally, any multi-objective problem can be turned into a parameter study.

The information found by treating the Shanxi problem as a two-objective problem can also be found either by adding the transport and construction costs in a weighted sum, and varying the weight, or by optimising only for construction costs, with a constraint on maximum transport costs, and varying the constraint[2]. In another problem concerning district heating systems, Molyneaux [89] studies the effect of varying one of the problem's input parameters—the network temperature—by adding the temperature as an objective. By both minimising and maximising temperature, the comportment of the network can be seen over the full range of temperatures.

Parameter studies are particularly effective for a number of analytical optimisation methods, where, given the current optimum, other optima for different weight and constraint values can quickly be found. When using EAs, though, this is not necessarily the case. The final population from one optimisation run may help in finding other optima, but more often than not, the entire population will have converged too tightly to the current optimum, and will have difficulty moving. Multi-objective optimisation, on the other hand, takes advantage of the parallel nature of EAs, and each individual can find a different trade-off between the objectives, while making the most of information on regions other individuals may explore.

3.2.5 Summary

The discussion methods for reducing multi-objective problems to single objective problems has frequently referred to a 'trade-off curve', but no concise definition of what this means has yet been presented. The next

[2] or vice-versa—optimising for transport costs and constraining construction costs

section will present just such a definition—that of Pareto-optimality, and will precisely define terms relating to Pareto-optimality that will be used in the rest of this work.

The introduction to multi-objective optimisation will then be essentially complete. The chapter will continue by presenting early methods for MOO *by evolutionary algorithms*, and then present ranking concepts, explain the relation between ranking and partially ordered sets (as discussed in §2.5), and then present more recent work in MOEAs—algorithms that use elitism, and the methods that are needed to manage the 'elite' population. Other topics relating to Pareto-optimisation by EAs, particularly ranking algorithms, and methods for discriminating between population members once they all have the same rank, will also be discussed.

3.3 Pareto-Optimality and Pareto-Optimal Frontiers

Methods for reducing multi-objective problems to single objective problems have been used for years, but a definition of an 'optimal trade-off curve'—the ideal result of a multi-objective optimisation—has been available for even longer. Pareto-optimality, presented in Pareto [99], gives a precise meaning to 'optimal trade-off curve' by providing a definition of optimality for a multi-objective problem:

Definition 1 (Pareto Optimality of a Vector) *A vector $\boldsymbol{u} \in \mathcal{F} \in \Re^m$ is said to be Pareto-optimal in $\mathcal{F}$ if $\nexists \boldsymbol{v} \in \mathcal{F}$ such that $v_k \leq u_k \forall k = 1, \ldots, m$ and $v_k < u_k$ for at least one k.*

The definition applies to a vector in the objective space, but it is easily extended, given an objective function, to parameter space:

Definition 2 (Pareto Optimality in Search Space) *Given a search space $\mathcal{S} \in \Re^n$, an objective function space $\mathcal{F} \in \Re^m$ and an objective function $\boldsymbol{f} : \mathcal{S} \to \mathcal{F} = \{f(\boldsymbol{x} : \boldsymbol{x} \in \mathcal{S})\}$, a vector $\boldsymbol{x} \in \mathcal{S}$ is said to be Pareto-optimal in $\mathcal{S}$ and $\boldsymbol{f}$ if $\nexists \boldsymbol{y} \in \mathcal{F}$ such that $f_k(\boldsymbol{y}) \leq f_k(\boldsymbol{x}) \quad \forall k = 1, \ldots, m$ and $f_k(\boldsymbol{y}) < f_k(\boldsymbol{x})$ for at least one k.*

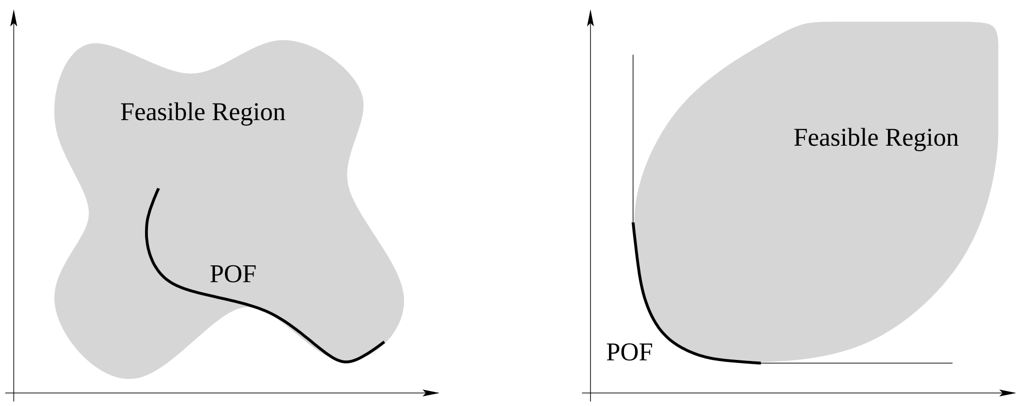


Figure 3.4: The Pareto-optimal front. Left: the feasible region in search space, with the location of the POF marked. Right: the feasible region in objective space, the POF is the bottom left corner of the region.

We can interpret the definition as meaning that a point is Pareto-optimal if there is no other point in the search space that is better in *all* objectives. Alternatively, it means that if a point is Pareto-optimal, moving from that point any other feasible point will make at least one objective worse. The concept of Pareto-optimality leads to the concept of a 'Pareto-optimal front' (POF) which is the set of all the Pareto-optimal points in a search space. A feasible search space, its corresponding attainable objective space, and the POF are illustrated in Figure 3.4.

The POF is not necessarily smooth or continuous, and can be disjoint. Nor is it strictly concave or convex. A more complex POF, resulting from a number of disjoint feasible regions, is shown in Figure 3.5.

Figure 3.5: A disjoint Pareto-optimal front. Three disjoint feasible regions in variable space and objective space. One of the feasible regions is completely dominated by the others, while the other two regions are complimentary.

If the optimisation problem has any real parameters, the POF probably has an infinite number of members—it will be made up of a number of discrete continuous sections. However, if the problem has only integer parameters, the POF can be discrete, and have a finite number of members. This is illustrated in Figure 3.6.

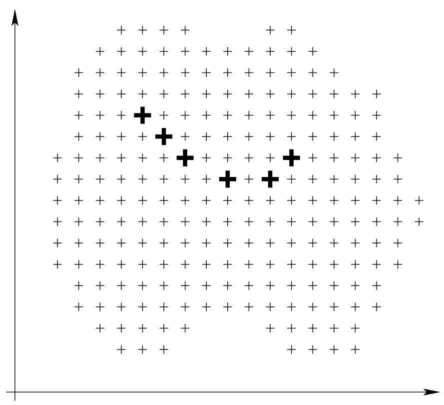


Figure 3.6: A Pareto-optimal front for an all-integer problem in decision-variable (left) and objective (right) space. The non-dominated points are marked with larger crosses.

To show the correspondence between the search space and the objective space, we use ‘Schaffer’s problem’ [114], a two objective optimisation problem in a single variable search space, that has been used for demonstrating Pareto-optimality, as an example, and as a test problem in any number of publications [64, 130, 124]. Schaffer’s problem has the form:

$$\begin{aligned} \text{minimise} \quad \boldsymbol{f}(x) &= (f_1(x), f_2(x)), \quad \text{where} \\ f_1(x) &= x^2, && (3.7) \\ f_2(x) &= (x-2)^2, && (3.8) \\ \text{subject to} \quad -1000 \leq x \leq 1000. \end{aligned}$$

The problem is almost trivially solved, as shown in Figure 3.7, which illustrates the relationship between the parameter and objective spaces, and shows the location of the POF in *parameter* space. Notably, Shaffer did *not* use this problem as a test problem, only a demonstration of concepts[3].

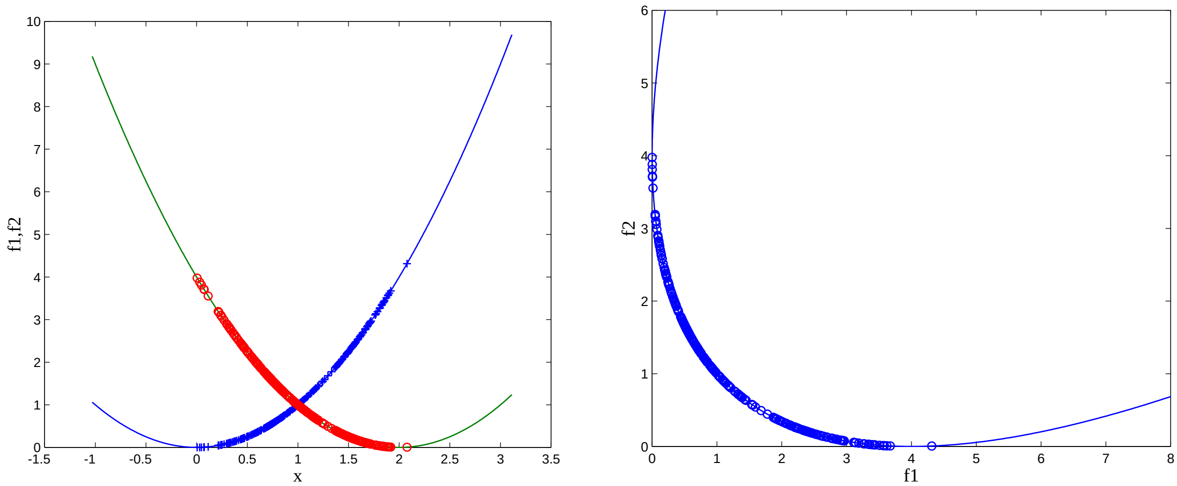


Figure 3.7: The Pareto-Optimal Front for Shaffer’s Problem. Left: the two objectives in parameter space. Right: The POF in objective space. The symbols are solutions found by a multi-objective optimiser.

3.3.1 Dominance and Non-Dominated Sets

‘Pareto-optimal frontier’ is a term that generally refers to the optimal (and possibly unknown) solution to a multi-objective optimisation problem, and in this work, to avoid confusion, this will *always* be the case.

Nonetheless, the concept of Pareto-optimality can be used during a population based optimisation to separate better individuals from the rest of the population. The extension to the population of a PBA is trivial, and in fact the Pareto-optimal part of a population is simpler than the POF of a MOP, as it is *always* discrete, and always has a finite number of members.

[3]Since then, several authors have used it as a (somewhat simple) test problem.

Because of this, the Pareto-optimal part of a population, not matter how converged, will only be an approximation to the POF[4], and thus it is important to make a clear distinction between the two.

For any given population, some individuals will be better than others in a Pareto sense. Specifically, following the definition of Pareto-optimality, an individual with an objective vector $\boldsymbol{v}$ is better than another with an objective vector $\boldsymbol{u}$ if $\boldsymbol{v}$ has at least one element strictly less than the corresponding element of $\boldsymbol{u}$, and all the remaining elements of $\boldsymbol{v}$ are less than or equal to the respective elements of $\boldsymbol{u}$. This is termed *dominance*. More formally:

Definition 3 (Pareto Dominance) *A vector $\boldsymbol{v} = (v_1, \ldots, v_m)$ is said to dominate a vector $\boldsymbol{u}$ if and only if*

$$\forall k \in \{1, \ldots, m\} : v_k \leq u_k \land \exists\, k \in \{1, \ldots, m\}, v_k < u_k$$

Given two individuals denoted by $\boldsymbol{x}$ and $\boldsymbol{y}$, and their objective function values $\boldsymbol{v}$ and $\boldsymbol{u}$, we say that $\boldsymbol{x}$ dominates $\boldsymbol{y}$ if $\boldsymbol{v}$ dominates $\boldsymbol{u}$. By convention, this is denoted by $\boldsymbol{x} \prec \boldsymbol{y}$.

Pareto-dominance can be weakened by removing the requirement that $\boldsymbol{x}$ be clearly better than $\boldsymbol{y}$:

Definition 4 (Weak Pareto Dominance) *A vector $\boldsymbol{v} = (v_1, \ldots, v_m)$ is said to dominate a vector $\boldsymbol{u}$ if and only if*

$$\forall k \in \{1, \ldots, m\} : v_k \leq u_k$$

Weak dominance is denoted with $\preceq$. Thus $\boldsymbol{x} \preceq \boldsymbol{y}$ reads '$\boldsymbol{x}$ weakly dominates $\boldsymbol{y}$'

A individual $\boldsymbol{x}^*$ is said to be *non-dominated* in a population if there exists no individual in the population which dominates it:

Definition 5 (Non Dominated Vectors) *An individual $\boldsymbol{x}^*$ is non-dominated in a population of I individuals if and only if*

$$\forall i \in \{1, \ldots, I\} : \boldsymbol{x}^i \not\prec \boldsymbol{x}^*$$

The set of non-dominated vectors in a population is termed the non-dominated set (NDS), and can be thought of as that population's approximation to the POF. As a PBA converges to the solution of a MOP, the NDS of the population will converge to the POF of that MOP.

3.3.2 Terminology

The meaning of 'Pareto-optimal frontier' varies widely between publications, and even within a publication, terminology can be confusing. For example, Horn [63] defined P_{online}, $P_{offline}$ and P_{actual} to be respectively the NDS of the current population, the best NDS found so far by the population[5], and the POF. These are transformed to $P_{current}$, P_{known} and P_{true} in Van Veldhuizen and Lamont [131], with largely the same

[4]Except in the rare case of perfect convergence to the solution of an entirely integer problem of manageable size.
[5]The algorithm did not use elitism, and thus could recede from the POF.

meanings. Some confusion can result from this changing notation, and even more from the presence of several ‘Pareto-optimal objects’ (the current, known and true fronts).

To avoid this confusion, in this work, the POF will *always* refer to the true Pareto-optimal front of a MOP, and NDS will *always* refer to the non-dominated set of individuals in the current population. As the algorithm presented in this work is elitist, there is no need for an ‘offline’ or ‘known’ object—a store of the best solutions found so far.

3.4 Dominance and Partially Ordered Sets

Goldberg [57] proposed one of the earliest methods for ranking a population using Pareto-dominance. First, the NDS of the entire population is found, given the rank 1 (the best), and temporarily removed from the population. The NDS of the remaining population is found, given a rank of 2, and temporarily removed. The process continues until the entire population is ranked.

Thus the population is sorted into a ‘partial order’ of groups of points that are mutually indistinguishable in a Pareto sense—and so Rudolph’s results can be applied to an algorithm using Goldberg’s ranking (elsewhere [124] referred to as non-dominated sorting). That the symbol used for the operator in Rudolph’s proof ($\preceq$) is the same as that used for the weak dominance relation is no coincidence.

In fact, a population arranged by non-dominated sorting is considerably ‘more’ than partially ordered. Indeed, just because an individual finds itself in the second rank does not imply that it is dominated by all the individuals in the first rank—in fact it will probably be indistinguishable from most of them. All that a Pareto EA requires in order for Rudolph’s results for convergence to part of the POF to apply is that it uses the weak-domination operator to choose at least one elite individual. If more elite individuals are chosen, and they occupy different regions of objective space, the algorithm can converge to a larger part of the POF. However, in the case of real-valued problems, it is difficult[6] to design an algorithm that converges to the entire Pareto-optimal set.

Thus any number of ranking algorithms or fitness assignments based on dominance can be used to build guaranteed convergence algorithms. As will be seen in the discussion of the convergence behaviour of MOEAs, and of the management of large non-dominated sets, and subsequent test results, most of the ranking algorithms are equivalent for the large part of an optimisation run, and fulfil Rudolph’s criteria for convergence to the POF, within the limits of the resources of the computer on which they are run.

3.5 Multi-Objective Evolutionary Algorithms

Having defined the conditions for optimality in a MOP, we will now take a step back, and examine the history of solving MOPs by Evolutionary Algorithms. Early attempts did not use the concept of Pareto-optimality at all, instead using methods such as ‘objective switching’ and automatic weighting. Even when a method

[6] and impractical, though not actually, on a finite precision computer, *impossible*, given sufficient resources.

for distinguishing individuals based on Pareto-optimality was proposed in [57], it wasn't until Fonseca and Fleming [50] that an algorithm using these techniques was published.

For a complete and thorough review of the history of MOEAs, the reader is referred to Coello Coello [19; 20] and Deb [31], which contains more details about certain algorithms than the original authors published.

3.6 Non-Pareto Algorithms

Although there were some earlier attempts at MOEA, the work of Schaffer [114] first recognised the need to return a set of solutions approximating the POF—earlier algorithms tried to find just one Pareto-optimal point. Schaffer's Vector Evaluated GA (VEGA) works by selecting different parts of the population based on different criteria. At each generation, the population is split into M groups, one corresponding to each objective, and roulette wheel selection is applied to each group separately. The resulting groups are then recombined to form the parent population, and combination and crossover are applied to this population as in a standard GA. Thus individuals are selected based on only one objective, but it is possible that a child will be created from parents that are good in different objectives. Because selection is based on individual objectives, the VEGA prefers individuals toward the 'edges' of the POF—which are often the least interesting solutions as, though they perform very well in one objective, they are poor in some other. Coello Coello [19] notes that the VEGA approach can be shown to be equivalent to using a weighted sum approach, and though the VEGA probably provides a more diverse range of solutions, it suffers from many of the same problems.

Kursawe [74] used a 'lexical ordering' of individuals for selection in a variant of ES. In lexical ordering the population is ordered by each objective, in order of 'importance' of the objectives. In Kursawe's work, a slightly different scheme is used, as a real lexical ordering on real-valued objectives will almost inevitably result in a population ordered by only the most important objective. Thus, selection consists of as many steps as there are objectives. At each step, the population is sorted by a randomly chosen objective (the probabilities of choosing a particular objective can be specified) and the worst individuals are removed. After the selection process is finished, the remaining population is used as parents for the next generation. It is worth noting that Deb [31] contains considerably more detail about the algorithm than the original work.

3.6.1 Modern, Pareto and Elitist Algorithms

Pareto-based EAs started to appear in the 1990s. Coello Coello [19] asserts that "Pareto ranking is the most appropriate way to generate an entire Pareto front in a single run of the GA", but had already noted earlier that there was no efficient algorithm to check for non-dominance in the set of feasible solutions [18], and cited this as the major problem with Pareto-based EA[7].

[7] This situation has recently changed—a method now exists for ranking large populations, and will be discussed in the next chapter.

Fonseca and Fleming [50] described one of the first implementations of a Pareto-based MOEA. In the Multi-objective GA (MOGA), each individual is given a rank that is equal to one plus the number of points that dominate it. The individuals are then sorted by this rank, and a fitness applied linearly to the entire population from the best ranked to the worst. The fitness of individuals with the same rank is then averaged, and all individuals in the same rank are given this fitness. Roulette wheel selection is then performed as in a standard GA. The MOGA uses standard GA sharing (though it is usually applied in objective space, rather than parameter space) to preserve diversity. The ranking scheme will be discussed further in §3.7.2.

Srinivas and Deb [124] present the Non-dominated Sorting GA (NSGA). Here, the ranking scheme is based on the technique proposed by Goldberg [57], and discussed in §3.7.1. A fitness is assigned to each individual, based on its rank, and again standard GA sharing is used, though this time in objective space. The fitness assignment is designed so that the best individual in the second rank after sharing—which has had its fitness degraded the least by sharing—has a lower fitness than the worst individual in the first rank—the individual in the first rank occupying the most crowded region. Stochastic remainder proportionate selection was used by the original authors, and the rest of the algorithm is as for a standard GA. The NSGA generally provides good coverage of the POF, but is criticised because the ranking algorithm is computationally intensive (of order N^3) and, more recently, because it lacks elitism. Deb et al. [33] update the NSGA (to the NSGA-II), which uses a faster non-dominated sorting algorithm (order N^2), and adds elitism.

Horn et al. [64] introduce the niched Pareto GA (NPGA), which uses restricted tournament selection based on Pareto dominance. A pair of ‘competing’ individuals is randomly chosen, as is a pool of individuals to compare against (typically about ten individuals). Each of the pair is compared for dominance with the pool. If one of the two individuals dominates the entire pool, and the other does not, or one is non-dominated with respect to the pool, and the other is dominated, the better individual is declared the winner. If there is a tie, the ‘better’ individual is chosen using a sharing calculated for each of the pair, with respect only to the pool, not the entire population. The NPGA has the advantage that its selection scheme is extremely fast compared to most other Pareto-based methods, which generally require a ranking over the whole population.

In Zitzler and Thiele [139], Zitzler [137] the Strength Pareto EA (SPEA) is introduced. This algorithm uses another ranking scheme (discussed in §3.7.3), called the ‘strength’. Strength is designed to solve some of the problems of other ranking schemes and to provide a sharing measure directly in the ranking. More significantly, the SPEA introduces elitism to an MOEA for the first time. The SPEA is generational, with a fixed population size, and as in a GA, in principle, the population is replaced at each generation. However, a certain part of the population (about 20%) is set aside as an elite population, that survives from generation to generation. This elite population consists of individuals in the first rank—new individuals in the first rank are added to the elite population, and as individuals in the elite population are dominated, they are removed. However, the elite population is of a fixed size, while the number of individuals that can potentially be in the first rank is extremely large, so a method must be used for reducing the number of individuals in the elite population (the term ‘truncation’ is used). The individuals in the elite population are clustered[8] into small groups in objective space. The point closest to the centroid of each cluster is found, and all the other points in the cluster are removed. This truncation method has the disadvantage that extreme points of the NDS can

[8] For a detailed discussion of clustering see §4.6.

be removed, resulting in shrinkage of the NDS (this is discussed further in §3.8). In Zitzler et al. [138] the SPEA is improved (to the SPEA-II), which, among other improvements assures that the NDS cannot shrink by 'pinning' its extreme points. Because Zitzler's thesis contained performance comparisons with a large number of other algorithms, and because convergence data was made available on the Internet, the SPEA has been widely used as a baseline algorithm when the performance of new algorithms or of modifications to existing algorithms are measured.

The Pareto Archived Evolution Strategy (PAES) developed in Knowles and Corne [69; 70] is a $(1 + 1)$ ES that the authors propose as a 'baseline' algorithm for MOEA. The $(1 + 1)$ ES uses a single individual, from which a new individual is created by local mutation. If the new individual is better than parent, it replaces the parent. In order to describe the POF, a set of all non-dominated individuals found so far is kept—the archive. In the cases where the child dominates or is dominated by the parent, it is easy to decide which individual is kept for the next iteration. However, when they are not dominated with respect to one another a method is needed for choosing between them. Here, the ranking scheme from Fonseca and Fleming [50] is used to rank each of the two individuals with respect to the members of the archive—that with the lowest rank is kept. Knowles and Corne note that this detail is critical to the performance of the PAES—a number of other methods of discriminating between mutually non-dominated individuals were tried, but none worked nearly as well as the method finally used.

Andersson and Wallace [3] present a multi-objective version of the Struggle GA [60]. The algorithm ranks the population using the domination count ranking developed by Fonseca and Fleming [50], and otherwise remains similar to the original SGA. When a new individual is added to the population, its rank is calculated, and compared to that of its nearest neighbour in objective space. If the new individual has a better rank, it is added to the population, and the neighbour is removed. Otherwise, the population is left as it is. After some experimentation, the MOSGA was modified so that the distance measure between individuals took account of the distance between individuals in both parameter and objective space.

3.7 Ranking

A number of ranking schemes have been developed for MOEAs. They differ in both conceptual and computational complexity, and each have a number of faults. Three of these ranking methods are described in detail here.

3.7.1 Goldberg's Ranking

Goldberg [57] proposes a ranking scheme based on successive non-dominated fronts, as shown in Figure 3.8. Here, the non-dominated front of the population or region is found, and these individuals are given the best rank (1), and removed from the rank search. The non-dominated front of the remaining population is found, and this time, all individuals are given the next best rank (2). The algorithm continues until all individuals have been ranked, or a maximum rank limit has been reached.

Goldberg's ranking scheme only allows fairly coarse discrimination between points, and no account is taken of how many individuals actually dominate a chosen individual. This means that provides no discrimination between individuals in highly populated regions of objective space and those in relatively sparsely populated areas.

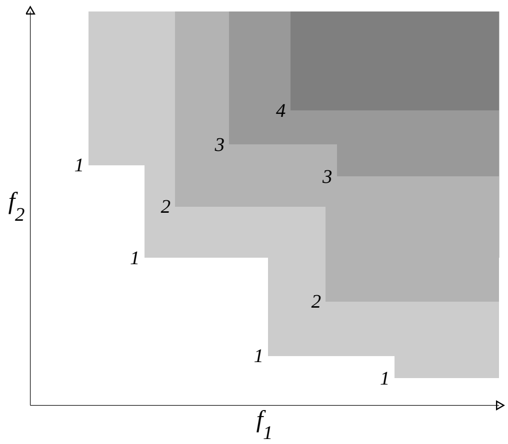


Figure 3.8: Non-Dominated Sorting

Deb et al. [33] propose a 'fast' algorithm for Goldberg's ranking. Here, every point in the population is compared with every other, to see which points dominate which. A list is also kept of how many points dominate a point, and a list is kept for each point, of which points it dominates. Once the entire population has been scanned, those points which have a domination count of zero are given a rank of one, and all of the points that they dominate have their domination count reduced. The points that now have a domination count of zero are in the second rank, and the process continues. The resulting algorithm is thus of order N^2. A variant of this algorithm is used in this work.

3.7.2 Fonseca and Fleming's Ranking

In Fonseca and Fleming [50]'s Multi-Objective GA (MOGA, see also de Fonseca [28]), the rank assigned to each point is the number of points that dominate that point plus one. Non-dominated points are assigned a rank of 1. This ranking scheme is shown in Figure 3.9.

The ranking allows a better distinction between individuals than non-dominated sorting, and is a little easier to implement.

3.7.3 'Strength' Ranking

Zitzler and Thiele [139] propose a third ranking scheme. Here, points on the non-dominated front are ranked according to the number of points they dominate, and dominated points are ranked according to the number

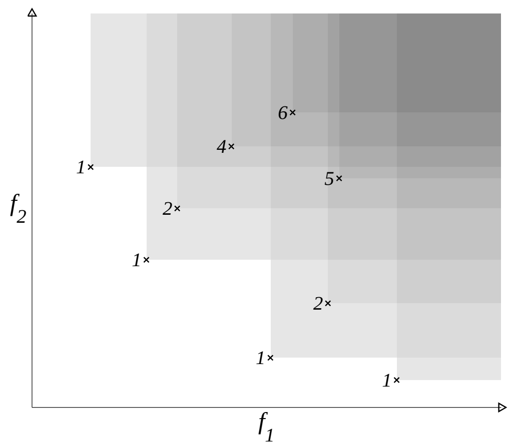


Figure 3.9: Fonseca and Fleming's Ranking Scheme

of (and which) non-dominated points dominate them. Thus, the rank for a non-dominated point is

$$R_{nd}(i) = \frac{\sum_{j=1}^{N_d} dm(i,j)}{N_d + 1} \tag{3.9}$$

and the rank for a dominated point is

$$R_d(j) = 1 + \sum_{i=1}^{N_{nd}} dm(i,j) R_{nd}(i) \tag{3.10}$$

where $dm(i,j)$ is one if i dominates j, and zero otherwise.

Interestingly, it can be seen from (3.9) and (3.10) that the best non-dominated point is that with the highest strength less than one (that which dominates the most points), and the best dominated point is that with the lowest strength greater than one (that which is dominated by the least, and least dominating points). Strength ranking is shown in Figure 3.10.

3.7.4 The Nature of Rank

The objective function values of an individual are fixed once the individual's parameter values have been set, and distances between two individuals in objective space are constant, and can be used as the measure of the similarity of the two.

The rank of an individual, on the other hand, is not fixed by the individual's location in parameter space, but rather by the individual's position in objective space relative to the rest of the population. Since the population is continually changing throughout an optimisation run, so an individual's rank is changing. This is fairly obvious: early in the optimisation, a very good new individual will be created, and given a rank of one. As the population progresses toward the POF, the individual will be overtaken by others, and its rank will decrease. Furthermore, the difference in rank between two individuals is not fixed, and cannot really

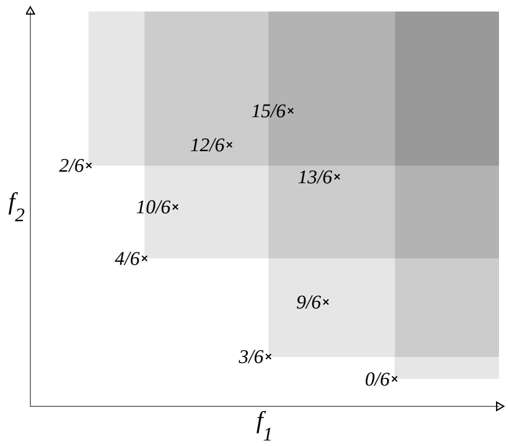


Figure 3.10: Strength Ranking

be used a measure of the relative merits of the two. Clearly, a dominating individual is better than those it dominates, but it can not be said that a just because B is one rank lower than A, and C is ranked three lower than A, that B is better than C. Consider an initial population containing just A, B and C, where B and C are dominated by A, but do not dominate each other. A has a rank of one, B and C have a rank of two. Two new individuals D and E are added. D is dominated by A, and dominates C, and E is dominated by D (and consequently A), and also dominates C. C's rank is now four, while B's is still two, even though C's objective relative to B has not changed at all.

The rank of an individual is an ambiguous measure of its importance, but ranking is nonetheless one of few viable methods for distinguishing between individuals in multi-objective problems. Its intransigent nature causes problems for most 'analytical' methods of optimisation—it does not make sense to think of the derivative of rank, or to assume it has a linear or low-order quadratic form and so try to interpolate or extrapolate it.

Fortunately, EAs are more than capable of optimising under these conditions. Indeed De Jong [29] pointed out that GAs were never intended for function optimisation, but for continuous optimisation in the face of a changing environment. Intentionally or not, most EAs will perform just as well as a GA at optimising in a changing environment, and thus the strange nature of rank should pose no problems for them.

As well as the intransigent nature of rank in general, each ranking method poses its own problems.

Non-dominated sorting, while it treats all non-dominated individuals equally, gives ranks that are highly dependent on population density to all other ranks. Thus an individual near (in objective distance) to the NDS in a crowded region of objective space, will have a worse rank than an individual the same distance from the NDS in a less crowded region. This is, however, justifiable—the non-crowded individual is probably so because it is difficult to find individuals in that region, so the individual's value should be recognised. Non-dominated sorting is illustrated in Figure 3.11.

Fonseca and Fleming's ranking and Strength ranking both incur lower computational costs than non-dominated sorting, and both reduce the relative effect of 'intermediate points'. However, both introduce a

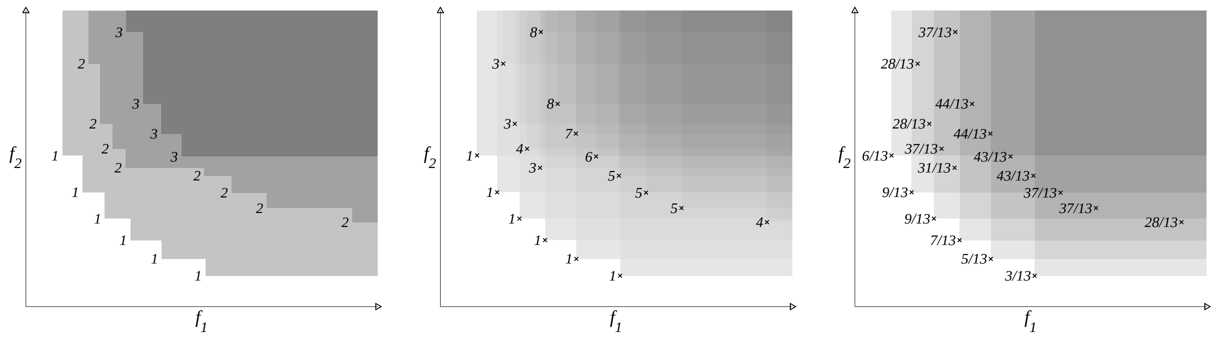


Figure 3.11: Problems with Ranking Schemes. Left: Non-dominated sorting. Individuals equally close to the NDS have a different rank depending on how crowded their neighbourhood is. Middle and right: Fonseca and Fleming's ranking and strength. The presence of 'intermediate points' has less effect, but points near the centre of the NDS get different ranks from those near the edges.

new effect—if points are reasonably uniformly distributed along the NDS, points at the ends of the NDS are ranked differently from those at the centres. In the case of Fonseca and Fleming's ranking, the NDS all gets the same rank, but of the individuals behind the NDS, those in the middle get lower ranks than those at the edges. In the case of Strength the situation is almost contradictory—points on the NDS are favoured when they are in the middle, while those behind the NDS are better off toward the edges. This is illustrated in Figure 3.11.

3.7.5 Relevance of Ranking

Experience shows that the effect of a particular ranking scheme on the progress of an optimisation may be very small. With modern elitist algorithms (particularly the very elitist algorithm presented here) the population rapidly converges to an almost entirely non-dominated state, where all three ranking schemes presented above give all individuals the same rank.

Throughout the tests presented in Chapter 5, only about the first quarter of an optimisation run was spent with more than one rank—even in cases where the convergence was not complete at the end of the run.

More important, perhaps, is how a population is managed once it is entirely non-dominated.

3.8 Thinning

Unlike single-objective optimisation, which searches for a single optimal point (or one in each local optima), multi-objective optimisation searches for an entire NDS. In theory, it is possible for an MOEA to find an infinite number of distinct individuals that can be considered optimal. In practise, as the population approaches the POF a large part of the population becomes non-dominated. At this point selection by rank no longer discriminates between points sufficiently well to keep the population size under control, nor to maintain selection pressure. This presents two problems.

The first is simply the burgeoning population size. Many MOEAs use a fixed 'elite' population size throughout, and thus are eventually be obliged to remove some non-dominated individuals in order to keep the elite population size within limits. In any case, the some of ranking algorithms proposed take time proportional to the square or cube of the population size and so the population size must be kept in check.

Even if an algorithm is capable of handling very large populations, a second problem can arise. An EA needs selection pressure to push the population toward an optimum, and in an MOEA the selection pressure is related the dominance relation. In a population which is near the POF and entirely non-dominated, there is no way to distinguish, by dominance, between individuals in the population. Further, most new *non-dominated* points[9] will not dominate any of the existing points. Thus after a certain time, without some kind of modification, the algorithm will stagnate near in the region of the POF, mostly generating new points that are dominated by the existing population, occasionally generating non-dominated points that only increase the resolution of the NDS in areas where it is not needed, and only rarely generating points that actually dominate members of the existing population.

The SPEA, as noted above, was the first MOEA to use elitism, with a fixed-size 'external archive' of points. A clustering method is used to reduce the size of the external archive[10]:

1. Put each member of the elite population into its own cluster
2. If the number of clusters is less than or equal to the maximum size of the elite population, go to 5.
3. Calculate the distances between each pair of clusters. The distance between clusters c_1 and c_2 is defined as the average distance in objective space between pairs of individuals on the clusters:
$$d(c_1, c_2) = \frac{1}{|c_1|.|c_2|} \sum_{i \in c_1, j \in c_2} d(i, j) \tag{3.11}$$
where $d(i, j)$ is the Euclidean distance between individuals i and j in objective space.
4. Find the two clusters with the minimal distance between them, and join them. Go to 2.
5. For each cluster, find the individual closest to the centroid of the cluster, and remove all other cluster members from the population.

Note that normally the size of the elite population will not be much larger than the limit and only a few clusters of more than one point will be formed, and points will be removed from those[11]. This method effectively resolves the problem of population control, and is designed to ensure that the population tends toward a good, even coverage of the Pareto-optimal front. However, it provides no selection pressure—there is nothing to suggest that the point nearest the cluster centroid is in any way better than the others.

The SPEA-II [138] corrected an important problem with the clustering method. In the original SPEA extreme points of the NDS could be removed, resulting in shrinkage of the span of the NDS in objective space.

Deb et al. [33] added elitism to the NSGA-II, and were obliged to handle similar issues. However, the

[9]Note that most *new* points will be dominated and will not make it into the 'living' population at all.

[10]For a detailed discussion of clustering methods, see §4.6.

[11]Unfortunately, Zitzler does not describe what is to be done when the cluster contains two individuals, as will often be the case, and so both are equally near to the centroid.

NSGA-II method of population control is more like sharing than Zitzler's archive truncation method. As well as its rank, each individual is given a *crowding distance*. This is a measure of the size of the hypercube surrounding the individuals that does not contain any other individuals of the same rank (that is, the hypercube with vertexes at the individual's nearest neighbours in the same rank). The algorithm starts with a population of $2N$, which is sorted by rank and crowding distance. The top N individuals are used to generate N new individuals, and these two groups are combined to form the $2N$ individuals of the next population.

In this work, a number or methods for 'thinning' the NDS are examined. These methods attempt to not only favour an even coverage of the POF, but also to prefer points that might be closer to the POF than others. These methods are detailed in §4.9, and results of tests are in §5.4.3.

3.9 Summary

Many optimisation problems are, on inspection, multi-objective. Even in problems where the objectives can be added (for example transport and construction costs), it can be interesting to see the trade-off between the two elements of the objective. Furthermore, the results of a multi-objective optimisation will lend themselves to answering more of a decision-maker's questions, both in terms of the results providing more information about optimal regions, and being useful for answering any unforeseen questions that may arise.

There are several well-established methods for reducing multi-objective problems to single objective problems. Weighting (and specifically in the case of this work, environomic) methods involve forming a weighted sum of the objectives. However, the weights must be chosen carefully. Requiring that some objectives be 'sufficient' by placing constraints on their values is also possible, but the choice of constraint values is even more critical than the choice of weights in weighting methods. Utility functions allow the user much more flexibility in the expression of their preferences regarding each objective, but such methods do not guarantee that a solution will be Pareto-optimal. All methods suffer from the disadvantage that an optimisation run results in just one optimal point, rather than a range of solutions from which the decision-maker can choose.

Parameter studies on the weights in a weighting method, or on the constraint values in a constrained method, can provide a range of solutions. This is equivalent to a multi-objective optimisation. Equally, for any multi-objective optimisation one can find an equivalent single-objective parameter study. Parameter studies are more appropriate when using analytical optimisation methods, while multi-objective optimisation works well with EAs.

Pareto-optimality forms the basis for methods for solving multi-objective optimisation problems, giving a formal definition of the desire to find points that represent a range of trade-offs between the two objectives—a set of points which can not be moved in the feasible region with reducing at least one of the objectives are searched for. Pareto-optimality leads to the definition of domination, which works between pairs of points, and non-dominated points, which are defined with respect to a population, while Pareto-optimality refers to the feasible region. The terms 'Pareto-optimal frontier' (POF), and 'non-dominated set' (NDS) are carefully defined to avoid confusion.

The dominance relation proves sufficient to define a partial ranking, and so Rudolph's results on convergence apply to optimisation algorithms based on dominance.

A number of EAs using Pareto-optimality in order to optimise multi-objective problems have been developed. These algorithms differ in how they use the dominance relation to define the 'rank' of an individual. The different methods have their advantages and disadvantages, but their overall influence on the performance of the algorithm may be limited, as the algorithms converge rapidly to an entirely non-dominated population.

The best of the MOEAs use elitism to store the set of non-dominated points found so far. So that the population size does not grow out of hand, methods are needed to discriminate between points of the same rank. The methods developed so far attempt to ensure a good coverage of the POF, but can not do much to prefer points nearer the (unknown) POF, and thus push the population toward convergence.

Chapter 4

The Queueing Multi-Objective Optimiser

4.1 Introduction

The queueing multi-objective optimiser (QMOO) is the MOEA developed at LENI for optimising energy system problems, and applied to a number of related domains. The nature of these problems has been discussed earlier, as have the algorithms used in the past to solve them, and the methods used to reduce what are essentially multi-objective problems to single objective problems.

Given LENI's past experience in optimisation, QMOO was designed with the following goals:

- QMOO should be a multi-objective optimiser, to avoid the complications associated with environomics;
- QMOO should attempt to find, as well as the global optimum, as many local optima as possible, in order to give the design engineer as much information about the problem as possible;
- QMOO should have as few critical control parameters as possible—it should not be necessary to run optimisations many times in order to tune parameters or representations;
- QMOO should converge rapidly (in terms of number of function evaluations required) to as many local optima as possible;
- Given that the objective functions of many energy system models take a long time to evaluate, QMOO should lend itself easily to parallelism.

Tests on test functions and real problems in subsequent chapters will show that these goals have been achieved. QMOO is a rapid and robust optimiser that requires little tuning, and which is able to find several local optima.

In order to achieve these goals, a number of choices were made early in the development of QMOO that have had a significant effect on the structure of the algorithm and the techniques used to provide rapid and robust convergence. The significant features of, and requirements on QMOO, and the reasons for the choices are briefly presented here. They will be explained in more detail in later sections.

4.1.1 Steady State

QMOO is a steady-state EA. Individuals are added to the population when they are ready, and are removed from the population when they are found to be lacking—creation and removal of individuals are completely separate processes, and there is no ‘formal’ control of population size. It was initially planned that the population would grow as large it wished in order to well-define the NDS and give a good approximation to the POF. This, however, proved computationally impractical[1], and in any case, it was found that certain methods of reducing the size of the NDS improved convergence.

Designing QMOO as a steady state algorithm later proved to have been a good choice—the steady-state structure, and the queue based algorithm used to implement it perform extremely well with respect to parallelism, the high (but varying) ‘population management’ requirements of QMOO, and experimentation with a number of sub-algorithms.

4.1.2 Elitism

QMOO, partly because it is a steady-state algorithm, is extremely elitist. Most other MOEAs are generational, with a current population, and an external elite set. QMOO, on the other hand, has a single population, which contains only the best individuals found so far. This makes QMOO converge fast, as evolution concentrates on the best individuals, but unfortunately poses problems for the preservation of diversity (in other algorithms, the non-elite population represents a significant source of diversity). Thus, in QMOO, other methods are pursued for the preservation of diversity, concentrating on preserving diversity where it is appropriate, rather than diversity for diversity’s sake. The most significant of these efforts is the dividing of the population into groups, but preservation of the tails of the NDS (§4.7.1) can also prove effective on some problems.

Initially, there was no parent selection in QMOO—because of the strong elitism, if an individual was in the population, it was considered good enough to be a parent. However, some problems required large populations in order to preserve diversity, which in turn caused them to converge slowly, because a poor individual had the same chance to be a parent as a good individual. With some preference for better parents, the diversity of the population could be preserved, with a minimal impact on convergence speed.

Elitism certainly contributes to QMOO’s convergence speed, and the diversity preservation problems it causes led to some unique, and important developments in the preservation of elitism.

[1] Late in the preparation of this document Everson et al. [43] published methods for making the handling of extremely large populations in MOEAs practical, but these were not experimented with.

4.1.3 Grouping

QMOO preserves diversity by dividing the population into groups in parameter space, and then letting these groups evolve somewhat independently—there is no competition between the groups, though there is some interbreeding. The grouping approach for diversity preservation differs from all methods presented earlier. Early versions of QMOO used the concept of 'dominatingly close' individuals to get a measure of the relative size of the parameter dimensions, an individual's neighbourhood, but this approach was not successful.

Grouping using clustering methods from statistical analysis was then tried, and worked very well. A number of clustering methods were tried, fuzzy c-means clustering proving to be the best compromise between speed and quality of results.

Identifying the groups in the population requires re-grouping the entire population from time to time—a time consuming task, and this must be allowed for in the QMOO algorithm. However, for some problems it is known beforehand that there is only one POF, and grouping is not necessary, so QMOO must be flexible enough to accommodate both the presence and absence of a grouping subprocess.

4.1.4 Computational Requirements

Because QMOO was designed with energy system optimisation in mind, where the objective functions will normally take some time to evaluate (times of the order of a second or more—the hybrid car model takes over ten seconds for an evaluation, and other models can take *much* longer), computationally expensive 'population processing'—such as the ranking and grouping algorithms, is justifiable. If complex analysis of the population can reduce the number of function evaluations needed for convergence or improve the robustness of convergence, this probably worthwhile.

This logic does not apply, however, to all types of problems, for example, the objective functions of the test problems in Chapter 5, are extremely quick to evaluate, and in terms of elapsed time (though not the number of objective function evaluations) it is probably more efficient to forgo complex population analysis, and accept that it will take slightly more function evaluations to converge, or even that the algorithm must be run several times to ensure that a good solution is found.

4.1.5 Multi-objective optimisation

QMOO implements Pareto-based multi-objective optimisation. This requires that the population be ranked from time to time in some manner (QMOO uses non-dominated sorting). Ranking algorithms are generally expensive, non-dominated sorting being no exception, and best only performed only occasionally if the population is large. This presents much the same problem as the grouping algorithm—an expensive operation must be performed from time to time on the whole population, requiring a flexible algorithmic structure. Furthermore, a number of ranking algorithms were used, the newer methods being faster than others. This lead to the development of an algorithmic structure that can easily cope with the changing requirements of 'population processing'.

4.1.6 Parallelism

Easy parallelisation of objective function evaluations was a very early goal in the design of QMOO. The requirements for parallelism were complicated by the need to run across a disparate network of computers of different speeds and availabilities, and by the need to evaluate objective functions for which the evaluation time varied depending on the input parameters. Once again, this required a very flexible algorithmic structure, able to cope with objective function results ‘arriving late’ and out of order from remote processors, and sometimes—when the remote workstations crashed or were turned off—not at all.

4.1.7 Summary

All these requirements lead to a unique algorithmic structure for QMOO, able to cope with all the problems above, and generalisable to other cases. The key to the flexibility is to no longer view the algorithm as a loop processing a population in generations or individuals one by one, but to view individuals as independent entities that must go through a number of processes in order to join the ‘main population’. Then, by placing the individuals in queues for each process the problems of needing to work in batches for efficiency (in the case of grouping), or one by one (say, in the case of producing new individuals), or of unknown processing times (as in the case of parallel objective function evaluations) are easily managed.

QMOO has fairly high computational demands per function evaluation, this being justified by the relatively complex objective functions for which it is intended. The algorithm is probably less suited to problems with very easy to evaluate objective functions. The basic architecture of QMOO can, in theory at least, accommodate both approaches: the ranking scheme could be replaced by Horn’s faster restricted tournament ranking [64], and grouping can be removed. However, the ‘variant’ of QMOO studied here is the one appropriate to energy system modelling, with computationally demanding population analyses.

Despite the queue-based architecture, externally, QMOO operates much like any other EA. The grouping methods complicate understanding of the algorithm a little, and for clarity, grouping is illustrated in Figure 4.1.

This rest of this chapter describes the QMOO algorithm in detail. First, QMOO’s approach to parallelism is detailed. This provides a logical base for explaining QMOO’s queue-based structure, which lists the processes an individual must pass through. This in turn leads to an explanation of the ‘generalised’ queue architecture in QMOO, which allows for easy redesign of the algorithm. Finally, each of the processes through which each of the individuals passes is described in detail, and each of the ‘sub-algorithms’, and any variants, are presented.

The QMOO algorithm is unique in a number of ways: not only does incorporate new techniques for rapid and robust convergence toward local and global POFs of a MOP, its algorithmic structure is completely new, and uniquely suited to energy system optimisation, where parallel evaluation of objective functions is often required, and to multi-objective optimisation, where expensive ranking processes may be necessary.

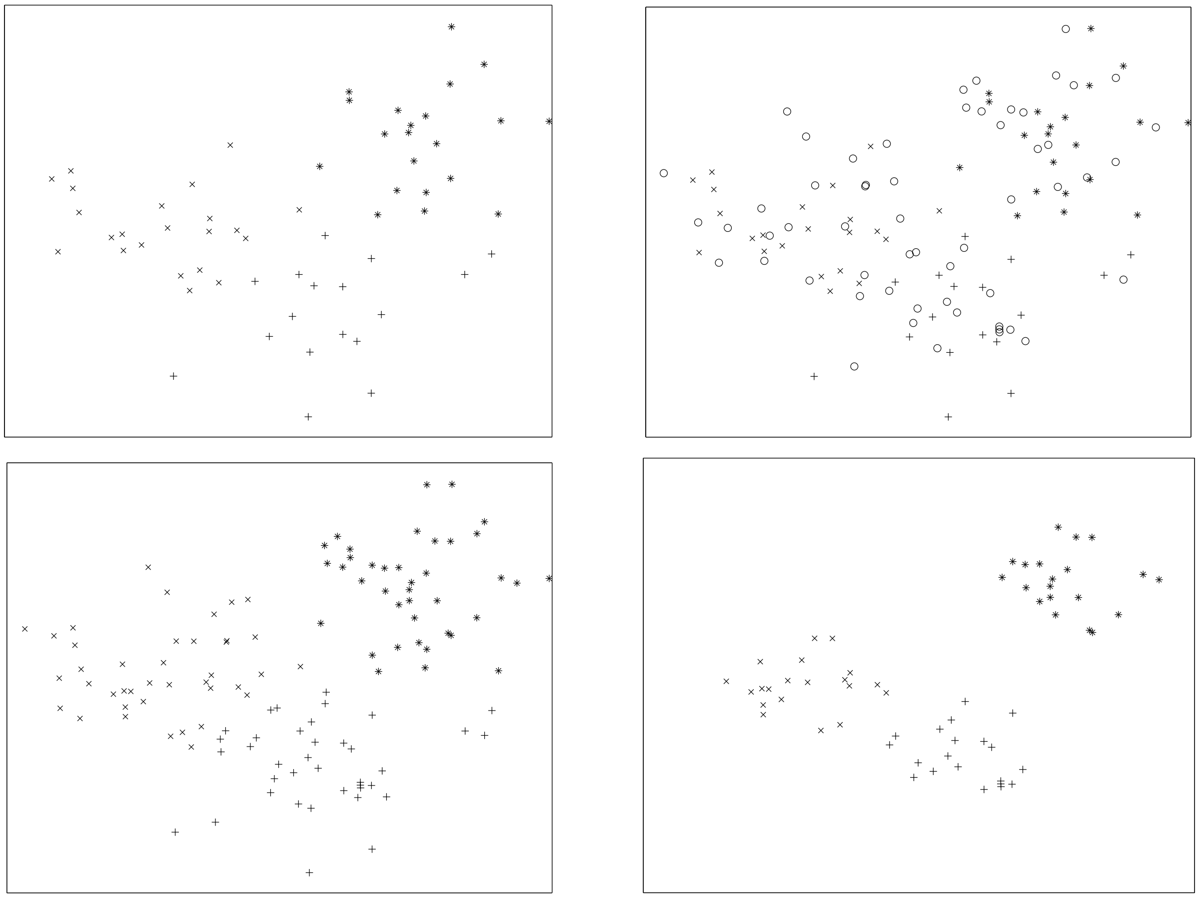


Figure 4.1: How QMOO works in decision space: (**Top left**) The 'current' population is divided into a number of groups (different symbols). (**Top right**) New individuals ('o') are generated (using EOC §4.4.1) in the general region of each of the groups, but they are not yet assigned to groups. (**Bottom left**) Once enough new individuals have been generated to make it worthwhile, the *entire* population is re-grouped—new individuals are assigned to groups, and some existing individuals in 'border regions' will change groups. (**Bottom right**) Each group is ranked separately, and thinning (§4.9) and tail preservation (§4.7.1) are applied. The worst individuals in each group are removed. Now, new individuals can be generated using the new groups.

4.2 Parallelism in QMOO

Parallelism with respect to generational (and some steady-state) single-objective algorithms was discussed earlier, and some of the problems related to parallelising steady-state, multi-objective EAs, particularly those where the time to evaluate the objective function can vary, were mentioned.

QMOO's approach to parallelism is largely the same as any other implementation—objective function evaluations are passed out to slave computers—but the queue-based algorithmic structure makes the parallelism much more flexible. All of the problems mentioned earlier are easily solved.

The interface used for communication between computers, based on the Parallel Virtual Machine (PVM) [125], is documented in Molyneaux [88].

4.2.1 Parallelising Objective Function Evaluation

Instead an individual having its objective function evaluated immediately after it has been assigned its decision variable values, the individual is put on a queue of individuals waiting to be evaluated, which the evaluation process empties from time to time. Once the individual has been evaluated, it continues the process of becoming a 'full member' of the population

Thus, any number of processors can pick an individual off the end of the queue, evaluate it, and return the individual, which now has an objective function value, to the main process. In this manner parallel processing of objective function evaluations is simply implemented.

Attention to a number of details is necessary, mostly due to the network on which problems were evaluated—an office network of computers of different speeds, often occupied with other tasks, and occasionally turned off. Further, the network latency in transferring a job from the 'master' (the main EA process) to a 'slave' (a processor performing objective function evaluations) was quite high. A final complication was added by the fact that one of the models optimised was old FORTRAN code, which tended to crash without explanation if some of the parameters, though well within constraints, were not exactly as it wished.

Thus:

- 'slave' processors keep their *own* queue of a couple of individuals to be evaluated. This means that the network latency for transferring an individual occurs while the previous individual is being evaluated, and the processor never stops working;
- the master checks that individuals have been returned from slaves after a certain time, and checks that the slave is responding. If not, it is assumed that the slave has been turned off or crashed, and its individuals are passed to other processors;
- if an individual is passed out more than three times to a slave to be evaluated, without ever receiving a correct response, it is assumed that the individual is causing the objective function to crash, and the individual is marked as infeasible;

This parallelism worked extremely well for QMOO, and reduced the elapsed time to solve large problems by a factor of about four—this on a network with six slave computers available, most of them only working some of the time. Brief experiments testing transfer times, and evaluation rates indicate that the network could be much larger. This, however, is highly dependent on the time required to evaluate the objective function. If it is low, the number of new individuals that can be generated and managed becomes a limiting factor[2].

[2] In the MATLAB version of QMOO, this limit is quite low—of the order of 100 evaluations per second. Preliminary tests show that a version written in C++ can manage up to 20,000 evaluations per second.

4.2.2 Parallelising Ranking and Grouping

In contrast to SOEAs, where the overhead of the algorithm is very small compared to the time taken for objective function evaluations, the algorithm overhead in an MOEA can be very high, due to the time taken to rank the population, and in the case of QMOO, to group it. Where possible, we would like these operations to be parallelised.

Ranking

The ranking algorithm used (described in §4.7), consists of two parts, first, adding a new individual to the dominance matrix, and secondly ranking (or re-ranking) the population. The first is relatively quick (complexity is of the order of the size of the population), but the second involves the entire population, and is of order N^2, and so it is best to re-rank the population as rarely as possible. Even in the *non*-parallel versions of QMOO, the ranking is delayed, in the same manner as the parallel version.

As with the parallelisation of objective function evaluations, once an individual has been evaluated, it is added to a queue of individuals to be added to the dominance matrix and ranked. When the queue is large enough the entire population is re-ranked. In the non-parallel version, all other processes stop while the re-ranking is performed, while in the parallel version, a 'snapshot' is taken of the dominance matrix and passed to the slave computer, so that the master can continue generating and evaluating new individuals (with the ranking) while the new ranking is determined.

In order to save on network transfer times for the dominance matrix, the matrix is transformed from a fully populated (0, 1) matrix to a list of all the indexes where one individual dominated another. However, network transfer times are still high, and no benefit is gained from parallelising ranking.

Grouping

Grouping can be parallelised in much the same manner as the ranking, except that it can be performed just after an individual has been created, and, in fact, at the same time as the individual is in the queue to be evaluated. However, clustering requires the entire population's decision variables to be sent to the slave computer—considerably more than the dominance matrix and objective function values that are required for ranking. This makes efficient parallelisation of clustering nearly impossible. Nonetheless, the 'queueing' approach to grouping, and the possibility of only regrouping occasionally, remains effective.

Network Latency

Unfortunately, on LENI's network at least, the time taken to send a population for ranking or clustering to a slave computer was prohibitively high, and in fact parallelising either had little effect on performance (as was the case for the ranking where only the dominance matrix was passed over the network) or actually slowed things down (as for the clustering, where the whole population had to be transferred).

There may be a number of avenues to remedy this, such as a 'dedicated' clustering slave that permanently stores the population, or even multiple processors on a shared memory machine, but these were not explored—the technique of only re-ranking or clustering when the queues were sufficiently long was found to work acceptably.

4.3 The Life of an Individual

Because of its queue-based design, it is difficult to describe the 'QMOO Algorithm' in the conventional manner. Instead, the processes through which an individual passes (effectively, the queues it is put into) are listed, and then each of the processes is examined in detail.

An individual goes through a number of processes during its 'life'. All of these processes (such as 'evaluation', 'grouping' or 'ranking') are only very broadly defined at this stage—the specific process, the algorithm used, or whether the operation is performed in parallel or not, is not specified.

Thus, the states of an individual are:

- **Creation** The 'main process' creates a new individual: A free space is found in the array storing individuals, and all its values are set to initial values.
  Once created, the individual is put on the queue to be 'assigned'.

- **Assignment** Parameter values are assigned to the individual: The process responsible for choosing parameters for an individual removes an individual from the queue, and gives it parameter values. If the algorithm is in the initialisation phase, the assigning process chooses parameter values at random from the ranges of allowed values. If the algorithm has finished initialisation, crossover and mutation operators are applied.
  Once parameter values have been assigned, the individual is put on the queue for evaluation, and, if the grouping does not depend on objective function values, for grouping.

- **Evaluation** The individual's objective function values are evaluated. If grouping depends on the objective function values, the individual queues for grouping, if not, and the individual has already been grouped, it queues for ranking.

- **Grouping** The individual is assigned a group. The individual passes itself to the group, so that the group can keep track of its variable space extents. If it has already been evaluated, it queues on the appropriate group for ranking.

- **Ranking** The individual is ranked. Once the individual has been ranked, it is marked as a 'full member' of the population and can be the parent of new individuals.

- **Removal** If at any stage, the individual is found, in some way, to be 'lacking' (if it is a duplicate of another point, its rank is too poor, or it is thinned), it dies, and removes itself from its group.

4.3.1 Regression

Both grouping and ranking are global operations that work on the entire population or group, thus, from time to time, a 'live' individual in the population will change groups, or change rank. In these cases the individual regresses back through the list of states. If the individual changes groups, it also needs to be re-ranked with respect to the new group, and for a short while is no longer in the 'live' population. The consequences of changing rank are smaller amounting mostly to bookkeeping.

Neither of the possible regressions are conceptually difficult. Some attention must be paid to the details to ensure that individuals are moved from group to group in a consistent manner, but this is only an implementation problem.

4.3.2 Duplicate Detection

As discussed earlier with respect to the no free lunch theorem, theoretically, algorithms that do not revisit search points could be better than those that do, and algorithms that revisit less than others may well have a performance advantage.

In practise as well it seems better that individuals with exactly the same decision variable values do not enter the population. Firstly, the duplicate individual will require the evaluation of the objective function at a point that has already been evaluated, wasting valuable processor time, and secondly, if a population of a limited size contains duplicates, a significant amount of diversity is lost, just because the second (and third...) copies of the individual could otherwise be elsewhere in decision variable space.

However, in a population of a significant size, and with a reasonable number of decision variables, finding out whether a new point is a copy of an individual already in the population is an expensive process, and only justifiable if objective function evaluations are extremely expensive. Fortunately, though, there are two simple ways of eliminating the duplicates in the population. Firstly, we note that the individuals in the population that a new individual is most likely to resemble are its parents. Thus, a quick comparison of an individual against it parents just after its decision variable values have been assigned will eliminate about 95% of all duplicates—*before* the objective function has been evaluated. Secondly, the ranking process involves comparing the objective function values of all individuals. At this point, if any individuals are found to have exactly the same objective function values, their decision variable values can be checked, and duplicates can be removed. This effectively and cheaply eliminates *all* the duplicates remaining in the population, but *after* the objective function has already been evaluated, when the computation time has been wasted. However, a small amount of diversity is been preserved.

One can wonder, though, whether in the case of an energy system problem, with a large number of continuous decision space variables, whether it is likely that duplicate individuals will be created. Experience shows that, depending on the operators used, and the form of the problem, that the answer is more often 'yes' than first expected. More importantly, in problems where duplicates arise, or when using operators that can create duplicates, loss of diversity can rapidly become a critical problem, with not just two, but many individuals occupying the same point in decision variable space. Since there are cheap and simple methods for keeping these duplicates in check, there is no reason not to use them all the time, even on problems where it is unlikely that duplicates will arise.

Note that the above methods do *not* prevent the algorithm as a whole from revisiting the same points. All that is assured is that two identical points do not exist at the same time in the population.

4.4 Assigning Parameter Values

Assigning parameter values is the 'combine and mutate' phase of a traditional EA. Individuals waiting in the queue to have values assigned to them are removed from the queue by the current 'assigner', which gives them gives them decision variable values. The assigner used is configurable, and can be changed during the algorithm. In QMOO, by starting with an 'assigner' that generates random points, the initial population can be generated in the same manner as the rest of the points, avoiding any initialisation code.

In the 'main loop' of the EA (after initialisation), parameters are assigned by crossover and mutation as in a conventional EA. The exact operators used can be chosen by the user, or be chosen from among a set of available operators by an evolutionary process—this process, termed 'evolutionary operator choice' (EOC) proved extremely effective on a number of problems, and will be discussed in §4.4.1.

The algorithm used for assignment is as follows:

1. Set g_1 to be a random group.
2. Choose a first parent p_1 from g_1, following the rules for parent selection established in §4.8.
3. Copy p_1's parameter values into c, the child, and mark p_1 as one of c's parents.
4. Pass c to the assigner. If the assigner needs to find a second, or any subsequent, parent, then choose a second parent p_2 randomly from the population less p_1:
   - Choose p_2 from the same group as p_1 β% of the time, and a randomly chosen other group the rest.
   - Again, prefer parents of better rank.
5. Check that c is not an exact copy of any of its parents, if it is, remove it from the population

Features of the implementation to note are:

- individuals are chosen by first selecting a group, and then an individual in that group. This is necessary to overcome issues that arise with groups of different sizes, and is discussed further in §4.6
- in step 3, p_1 is copied into c before any operators are used, the operators then work on c and p_2. This means that all operator families (random generation, combination and mutation) can have exactly the same interface.
- after an individual is assigned its parameter values, a check is performed to see if it has exactly the same parameter values as any of its parents. This is as noted in the section above on duplicate detection and removal.

In this work, β was always 90%, meaning that individuals nearly always combine with members of the same groups. It may be worth experimenting with this parameter, but it is quite likely only of secondary importance.

4.4.1 Evolutionary Operator Choice

The assigner used in this work implements a method of choosing combination and mutation operators from a pool of available operators. This process, EOC, was developed while working on the Shanxi problem. This problem, in contrast to the energy system problems presented here, which have mostly real parameters, is a combinatorial optimisation problem. A number of simple operators were developed to aid the resolution of the problem, and so questions arose as to when, or how much, to use each operator. These questions were simply answered by letting the choice of operator evolve with the population, and this approach proved very effective.

Any number of systems for choosing control parameters of EAs have been proposed. Eiben et al. [41] presents an extensive review. The best known example of using an evolutionary process to control parameters is the evolution of mutation radii in ES, and other methods attempt to choose population size, mutation rates, sharing radii and more. However, operators are generally chosen by the algorithm user, or even form part of the 'design' of the algorithm, and there is a belief (not necessarily false) that experimentation is needed with each problem to see which operators will perform best. If several operators are to be used for a single problem, a probability of using each operator is assigned by the algorithm user, again relying on experience or knowledge of the problem that the user may have.

Thus QMOO's approach to choosing operators is almost unique: the only other algorithm using evolutionary processes for operator choice known to this author is 'flexible evolution' [53].

The algorithm used for EOC (here including the details already covered above for clarity) is as follows:

1. Set g_1 to be a random group.
2. Choose a first parent p_1 from g_1, following the rules for parent selection established in §4.8.
3. Copy p_1's parameter values into c, the child, and mark p_1 as one of c's parents.
4. Choose a combination operator:
   (a) If p_1 was generated randomly (it was part of the initial population), choose a random combination operator.
   (b) Otherwise, use p_1's combination operator ϕ% of the time, a random operator the rest.
5. If the combination operator is not 'no combination',
   (a) Choose a second parent p_2 randomly from the population less p_1:
      - Choose p_2 from the same cluster as p_1 β% of the time, and a randomly chosen other cluster the rest.
      - Again, prefer parents of better rank.
   (b) Use the combination operator on c and p_2 to generate a new version of c
6. Choose a mutation operator:
   (a) If p_1 was generated randomly, choose a random mutation operator.
   (b) Otherwise, use p_1's mutation operator ϕ% of the time, a random operator the rest.
7. If the combination operator is not 'no combination', use the combination operator on c to generate a new version of c
8. Check that c is not an exact copy of either p_1 or p_2, if it is, remove it from the population

The intention of EOC is that if an operator is generating successful children, then those children should continue to try to use that operator. The success of a child is measured by whether it remains in the population—operators that produce *non-dominated* children will be successful. However, in a multi-objective problem, non-dominated children are not necessarily better than the existing population, they can just fit into gaps in the NDS. Thus, an operator that is very successful in generating new individuals among the known NDS but not dominating any of it may be preferred over an operator that occasionally generates a new individual that dominates part of the NDS. A number of other criteria for measuring the success of an operator can be imagined, but none were tested.

EOC assumes that different operators will work better on different problems, and even that different operators may be appropriate at different moments when resolving a problem. Tests in Chapter 5 show the first assumption to be true, and less formal tests have indicated that the second assumption is probably true also. However, using the 'fastest' operator all the time can often be dangerous, as one of the test problems will illustrate, converging rapidly can lead to problems with diversity that prevent total convergence, while slower operators will converge more completely. It could be hoped that EOC would do well in such a situation by using the rapid operator when it is still working, and switching to the slower operator later. However, this is not the problem. Once the rapid operator has done its work, the population can be left in a state that is not able to continue converging toward an optimum.

Thus, preservation of diversity, and good mutation operators are of utmost importance.

4.4.2 Mutation Operators

Combination operators were detailed in Chapter 2, as the operators used in this work are found in the literature. The mutation operators, however, are unique to QMOO. They are not, however, particularly revolutionary. The operators were designed to be used with EOC, to demonstrate the method—by controlling the use of the three operators, it is hoped that EOC is able to generate an appropriate mutation radius throughout the solution of a problem. However, no tests were performed to compare these fairly simple operators with others found in the literature.

There are three mutation operators: two 'local' operators, and one 'global'. The local operators cause small mutations in the region of an individual or the population. The global operator causes mutations that cover the entire search space—thus meeting Rudolph's requirement on mutation operators. Tests in Chapter 5 will show the effect of removing any of these operators—in the case of the global mutation operator, this is a very nice demonstration of Rudolph's results.

The local operators use a fraction of the standard deviation of either an individual's group's spread in decision variable space, or the entire population's spread as a mutation radius. The reason for using the standard deviation is simple: there is a simple formula for the variation of a population that can be used to recalculate the variance of the population every time an individual is added to or removed from a group or the population:

$$\sigma^2 = \frac{\sum_{i=1}^{N} x^2 - \frac{1}{N}\left(\sum_{i=1}^{N} x\right)^2}{N \quad \text{or} \quad N-1} \tag{4.1}$$

This makes tracking variances very simple—it is only necessary to keep totals of sum and sum of squares of the decision variables. Earlier mutation operators based their mutation radii on the *span* of a group or the population. This worked just as well as the standard deviation approach, but an inordinate amount or processor time was spend searching groups for their smallest and largest members in each decision variable. The variance calculated by this method quickly accumulates round-off error, and so it must be recalculated from scratch from time to time.

Normal Mutation

The normal mutation operator is a local mutation operator that changes all of the individual's variables to values chosen from a normal distribution with a mean at the individual's original values, and a variance of half of the individual's group's variance. The fraction of the group's variance used in the mutation (as with the fractions for the other operators) was not tested. However, tests do show that not using the operator has a significant negative effect on convergence.

Uniform Mutation

The uniform mutation operator is a local mutation operator that changes one of an individual's variables by choosing it from a uniform random distribution centred on the population centre, and over a range that is ten times the standard deviation of the entire population in that variable.

Global Mutation

The global mutation operator changes all of an individual's variables by choosing them from a normal distribution that is centred on the individual, and with a span that is one twentieth of the search space for each variables. This operator provides, in some form, the global mutation that is necessary to fulfil Rudolph's conditions[3].

4.5 Evaluating Objective Function Values

Evaluating objective function values is straightforward. Because a queue of individuals needing evaluation is available, the objective function can make the most of any benefits from evaluating several individuals in parallel (there can be a significant speed advantage in this case if the objective function is written in MATLAB), and the objective function is easily parallelised. Furthermore, the interface to the objective function is 'narrow'—the objective function needs to know very little about the internal workings of QMOO. This is even more the case because an individual's parameter and objective values are stored separately from the individuals, in an array containing values for the entire population.

4.6 Grouping

The earliest version of the QMOO, the 'Clustering Pareto EA' (CPEA) [90], lacked many of the features described here. Its distinguishing point was the method used for the preservation of diversity, and for finding multiple local optima.

In the CPEA, and QMOO, the population is analysed to find individuals that are grouped in decision variable space[4]. The groups then evolve as almost entirely separate populations, without competing with each other. Groups can interbreed, and more importantly, the population is re-analysed frequently, meaning that individuals can change groups, groups can merge, and new groups can be discovered.

This grouping allows QMOO to find approximations to multiple local POFs, even when one of the POFs completely dominates another, as shown in Figure 4.2, which illustrates the optimisation of the simple

[3] Even though a normal distribution theoretically covers the entire search space (and more), tests showed that with the 16-bit normal random number generator used in this work, the largest z generated is just over five standard deviations. Thus, about half the search space is covered (since $-5 \leq z \leq +5$ has a span of ten standard deviations), and so, with this mutation operator, QMOO is *not* guaranteed to be globally convergent!

[4] In fact, the grouping can be in decision variable space, objective space, both, or any subset of the decision variables and objectives.

function

$$f_1 = \sin(x) * (1 * x/20) \qquad (4.2)$$
$$f_2 = \cos(x) * (1 * x/20)$$

over the domain $0 < x < 20$.

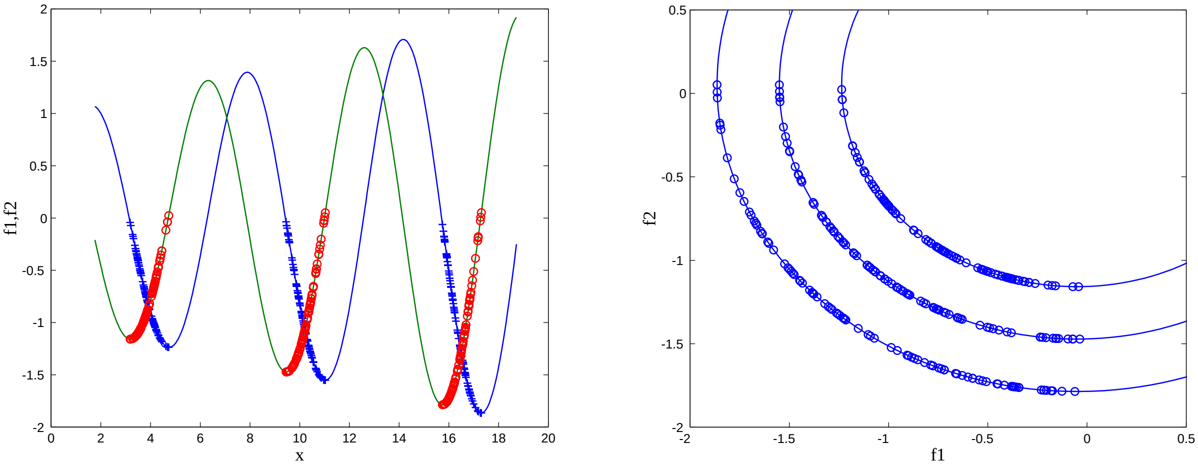


Figure 4.2: Multiple POFs in variable (left) and objective spaces (right). Three separate POFs (lines) and NDSs (shapes) are shown.

Another, more colourful example is that of Himmelbau's function, which has been used as a niching and multi-modal test problem by several authors [61, 83, 60]. The function has four equal height peaks, and we test to see if all solutions are found and maintained for a large number of generations. Here, to make a multi-objective problem, f_2 is a second Himmelbau function of half the size:

$$\begin{aligned} f_1(x,y) &= 5 - \frac{(x^2+y-11)^2+(x+y^2-7)^2}{200} \\ f_2(x,y) &= f_1(\alpha,\beta) \quad \text{where} \quad \alpha = 2x, \beta = 2y \end{aligned} \qquad (4.3)$$

A population that has groups on all four local POFs is shown in Figure 4.3.

In the context of energy system design, the advantages of finding multiple local optima are clear, and have been stated earlier. Briefly, the results of an optimisation by QMOO give the decision maker a comprehensive view of the interesting regions of the model. If the QMOO has converged and found all local optima, there are no other model configurations that will be interesting to the decision-maker. Nor do the solutions found by QMOO present too much information. The decision-maker may decide to change the qualitative weighting on objectives, or might find that one of the local optima is more interesting than the global one, or might even use the output from QMOO to debug and refine the precision of the model.

Grouping not only helps to find multiple local optima, it is also an effective method of preserving diversity in the population, and assuring that the search space is thoroughly explored. This comes at a cost, however:

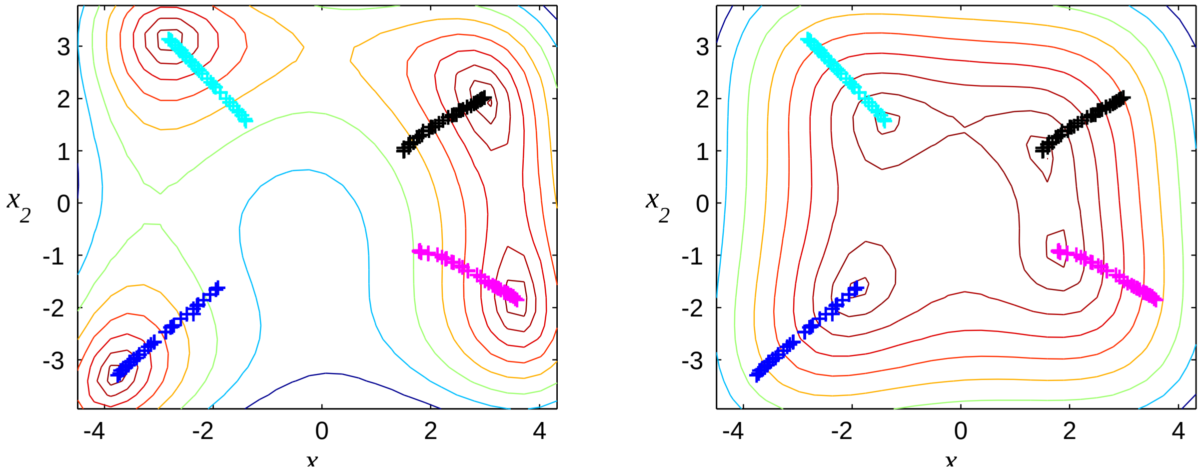


Figure 4.3: Contours of the Himmelbau functions and a final population. Four separate groups cover the four POFs.

convergence is slowed. In cases where there *are* multiple local optima, this is more than acceptable—it is better to take a little extra time to find several regions of interest, than it is to find just one. In cases where there is only one optimum, though, better convergence behaviour would preferred if possible.

It must be clear that if the slower convergence results in a more robust convergence to the optimum, due to its preservation of diversity, then slower, more robust convergence is better. However in cases where diversity preservation by grouping is slowing convergence without enhancing robustness, either because the faster method is converging well also, or because neither is doing particularly well, grouping becomes preservation of diversity for diversity's sake, or the preservation of an inappropriate type of diversity.

One example of this is the district heating problem (§6.6.1). Here, clustering was necessary to find a full range of solutions, and indeed several distinct local optima are present. However, clustering was also needed to find the entirety of some of the POFs. This is an inappropriate use grouping to preserve diversity. The problem was resolved by adding tail preservation (§4.7.1) to QMOO, which aids convergence to the entirety of a single POF—a much more directed method of diversity preservation.

A critical element of the grouping algorithm is that *groups*, rather than individuals, are given equal chances to breed. This is to ensure that all groups can converge at a reasonable rate, rather than having one large group that hogs nearly all the process time. If *individuals* are given equal chances to breed, at some early stage one group will become larger than the others, and evolve quicker, as, thanks to its larger population, it will get more chances to breed. As long as the large group continues to converge at the same *rate per function evaluation* as the other groups, this does not pose a problem. But this will clearly not be the case forever, and when the NDS of the larger group does get close to the POF, its convergence will slow. Close to the POF, most new points generated in the large cluster will not make it into the population, a few will be non-dominated—usually because they manage to squeeze 'between' two other top ranked points, adding resolution to the NDS where it is not needed—and only a few points will manage to dominate points already in the group. So, without some kind of 'population control' (§4.9) on the group, it will continue to grow—taking more and more process time, to achieve less and less. Meanwhile, the other, smaller groups will receive less and less process time, and, because they have not yet converged—so new individuals that dominate several others will be more common—they will not manage to grow larger, and ‘win’ the process time they need to converge completely.

The simple remedy to this problem is to give the same number of breeding opportunities to each group, rather than to each individual.

In summary, grouping is an excellent method for finding multiple local optima, and works well for the general preservation of diversity. However, when specific needs for diversity can be identified, it is better to attempt to fulfil those needs, rather than use grouping.

Several methods were tested for identifying the groups.

4.6.1 Hierarchical Clustering

Hierarchical clustering [4, 126] is an *agglomerative* clustering method. Initially, all the data points start as separate clusters and the two clusters that resemble each other the most are joined. The process is repeated until the desired number of clusters remain, or some other measure of cluster quality has been attained. Different methods of hierarchical clustering have different methods for judging the similarity of the clusters—termed the *linkage* method. Several methods are available, and all of the following were tried in the CPEA, but as hierarchical clustering was not used in the final versions of QMOO (fuzzy c-means clustering proved equally good, and much faster), they were not tested rigorously.

- In *single* linkage clustering, the shortest distance between a point in the first cluster and a point in the second cluster is taken as a measure of similarity. This method is simple, but it tends to form long thin clusters;
- *Complete* linkage, on the other hand, uses the longest distance between any two clusters. This, again is a simple method, now preferring rounder clusters, and proved more appropriate for this work—at least on the test functions used;
- *average* linkage uses the average distance between all pairs of points in the two clusters. This is the method used in the SPEA [137] for the truncation of the elite archive, where the clustering is performed in objective space;
- *centroid* linkage uses the distance between the cluster centroids as the similarity measure;
- *Ward* linkage uses a weighted average distance between clusters as similarity measure. Ward linkage was found to be the best of the clustering methods for the test problems it was tried on, but it is also the most computationally intensive of the linkage methods.

Usually, clustering is performed until all individuals and subclusters have been combined into one supercluster. The output of this process is a ‘tree’ representing the hierarchy of the clusters. The ’leaves’ represent the individual data points making up the cluster, branches represent the joining of two clusters, and the branch joints represent the clusters formed. Once the tree has been formed, it can be traversed to find the desired clustering. It can be traversed until the desired number of clusters have been found, or some condition on the similarity of the clusters has been met.

Using a condition on the similarity of clusters *should* mean that hierarchical clustering can not only find the clusters, but also find the right number of clusters. In practise though, using the similarity measure did not work at all, and the number of clusters to find had to be specified in advance.

4.6.2 Mean Neighbour Value Clustering

Dugad and Ahuja [39] present an algorithm based on pairs of points' 'mean neighbour value' (MNV). Given one point in a population, all the other points can be sorted with respect to their distance from the first point. The rank of each point in this list is the point's 'neighbour value' with respect to the first point. The MNV of two points A and B, then, the sum of their NVs ($MNV(A, B) = NV(A) + NV(B)$).

The algorithm works by combining into clusters all points that can be reached by links with an MNV less than a limit set by the user. There is some bookwork to ensure the consistency of all the links between clusters of different density, but the algorithm is mostly straightforward and rapid. There is even the possibility of finding the correct number of clusters by varying the MNV limit, and, unlike the methods used for hierarchical clustering, this works well.

However, the clusters found by MNV clustering are nothing like those found by either hierarchical or fuzzy c-means clustering, and MNV clustering proved inappropriate for this work.

4.6.3 Fuzzy C-Means Clustering

Fuzzy c-means clustering (FCM) is a clustering technique that attempts to minimise the distance from each point to be clustered to the centre of its cluster. It does this both by moving the centres of the clusters, and by changing points from cluster to cluster. The clustering is *fuzzy* because points are not assigned to a single cluster, instead they have a degree of membership in each cluster. Thus, a point near the centre of a cluster might be 90% a member of that cluster, and a few percent a member of every other cluster.

This makes FCM a continuous optimisation problem. We wish to minimise the weighted distance between points and cluster centres:

$$J(\boldsymbol{U}, \boldsymbol{V}) = \sum_{k=1}^{n} \sum_{i=1}^{c} u_{ik}^{m} d^2(\boldsymbol{x}_k, \boldsymbol{v}_i) \tag{4.4}$$

where $\boldsymbol{x}_k$ is one of the n data points, $\boldsymbol{v}_i$ is one of the c cluster centres, $\boldsymbol{u}_{ik}$ is the membership degree of point k in cluster i, m is weighting exponent for the fuzzy membership function (generally given a value of 1.25), and $d^2(\boldsymbol{x}_k, \boldsymbol{v}_i)$ is the square of the distance between a point and a cluster centre.

This objective function can be minimised by an iterative process where the membership weights are updated to:

$$u_{ik} = \frac{(d_{ik}^2)^{1/1-m}}{\sum_{j=1}^{c}(d_{ij}^2)^{1/1-m}} \tag{4.5}$$

and the cluster centres are placed at the weighted centre of their members:

$$\boldsymbol{v}_i = \frac{\sum_{k=1}^{n} u_{ik}^m \boldsymbol{x}_k}{\sum_{k=1}^{n} u_{ik}^m} \tag{4.6}$$

Note that the distance from cluster centres to points always appears squared in these equations, there is no need to perform the full calculation of the Euclidean distance (as many implementations of FCM do).

FCM is much quicker then hierarchical clustering, and though the clusters it identifies are not exactly the same, the results seem to have the same quality. FCM is the only clustering method used for the results in this thesis.

4.6.4 Number of Clusters

In general, the algorithms used for clustering need to be told the number of clusters present in the data, and will find that number of clusters. This would be acceptable if the decision-maker knew something about the form of the objective function beforehand, but often this is not the case.

Hierarchical clustering methods can theoretically also find the number of clusters in a data set, but in practise this does not work. Molyneaux [89], working on a district heating network problem that required several clusters, developed methods for determining the number of clusters with FCM, and these methods are reported to work quite well. They are not discussed here.

4.7 Ranking

The ranking process is responsible for ranking the members of a single group. The entire process is complicated by the need to preserve the tails of the group (described below), but a particular ranking algorithm is quite simple, and needs only a list of objective function values for each individual. Thus the process can be broken into two parts—the 'overall process' that manages the group and the tail regions, and a the ranking algorithm proper, which may be called several times on a single group, in order to rank both the main front and the tails.

4.7.1 Tail Preservation

An MOEA, after an initial phase in which the population moves rapidly to the region of the POF, will spend much of its time with an entirely non-dominated population. This is particularly true of QMOO, where there is only an elite population. This makes preserving the diversity of the NDS is critical to robust convergence.

In studying the convergence of the SPEA both Zitzler et al. [138] and Everson et al. [43] both noted that

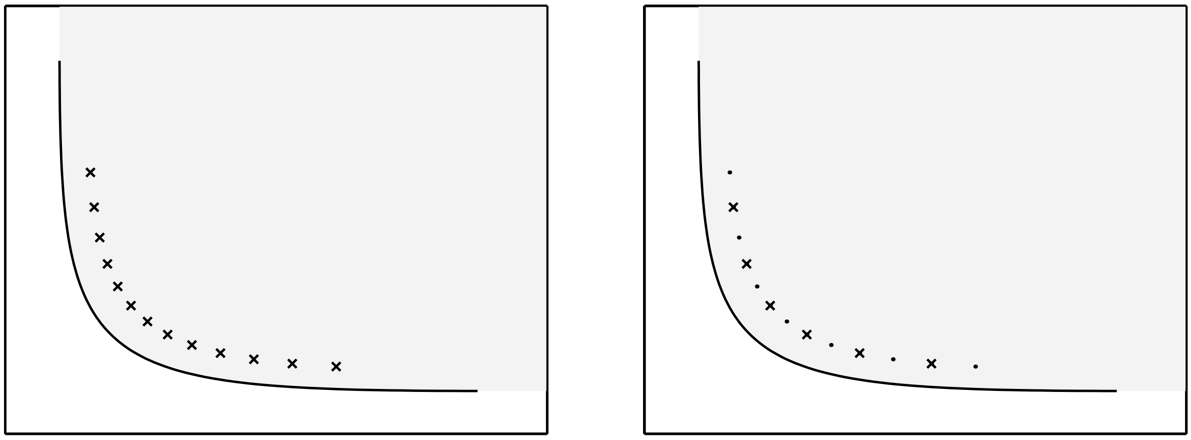


Figure 4.4: Truncation leading to shrinking of the NDS. Left: the population before truncation. Right: the population after (old population shown with dots)—the coverage of the POF by the NDS is reduced

the 'truncation' method used in the SPEA for reducing the size of the elite population could result in the shrinking of the NDS (Figure 4.4). A new truncation method was developed to ensure that the extreme points of the NDS were always retained, resulting in the SPEA-II. Everson also suggested 'pinning' of the extreme points of the NDS, but went on to propose algorithmic structures to handle extremely large NDSs, and thus dispense with truncation altogether (these methods are discussed in §4.7.4).

QMOO's thinning methods assured from an early stage that the NDS could not recede. However, particularly in the district heating problem problem presented in Molyneaux [89], the problem proved not to be of shrinking NDS, but of an NDS that failed to grow fast enough. Specifically, it was noted that certain regions of the POF were only explored when the number of groups was sufficiently high, despite the fact that the groups in question were joined in both parameter and objective space. The fact that these regions were explored when the number of groups was sufficiently high (as illustrated in Figure 4.5), could be seen as an advantage of grouping. However, grouping is intended as method of finding distinct optima, and while any other benefits are clearly useful, other methods for enhancing the exploration of single POFs were investigated.

Once QMOO's population is entirely non-dominated, (i.e. the first rank is sufficiently large) individuals that would be in the second rank can no longer enter the population. This is reasonable—best results are obtained by working with the best individuals, and diversity is preserved by grouping. However, if (as illustrated in Figure 4.6), the NDS does not yet cover the entire POF, some second ranked points will fall in the 'tail region' of the NDS. These points represent valuable information about how to find the remaining parts of the POF. Thus, a scheme that preserves these tail regions could considerably improve robustness of coverage of the POF in cases where the entire POF is hard to find.

The method for preserving tail regions is conceptually fairly simple (though its implementation requires some careful bookkeeping). Individuals are ranked as normal, and 'tail boxes' are defined at the ends of the NDS (Figure 4.7), having, say, 5%, of its linear dimensions. Individuals that are not in the first rank

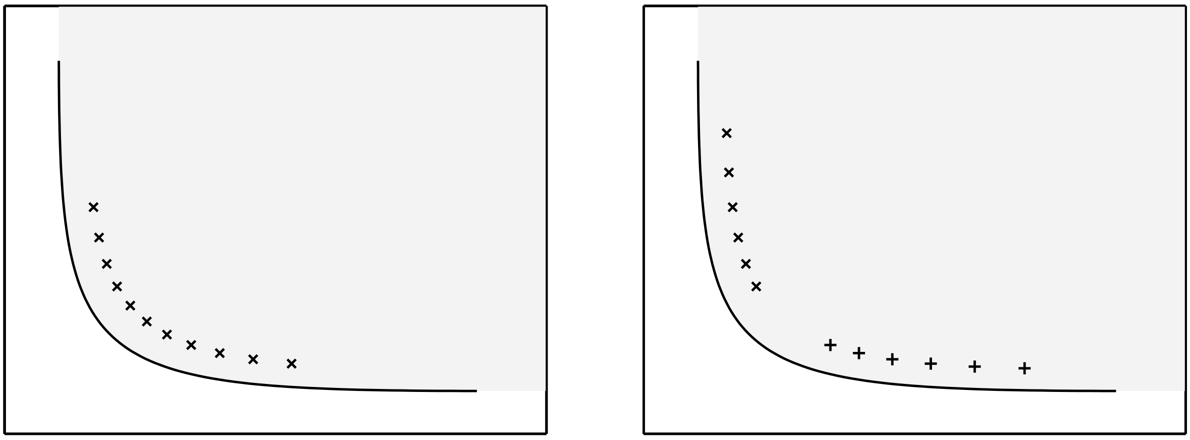


Figure 4.5: Lack of exploration of the POF in QMOO, and the effect of grouping. Left: NDS without groups. Right, with two groups ('$\times$'s and '$+$'s), more of the POF may be found, but not very efficiently.

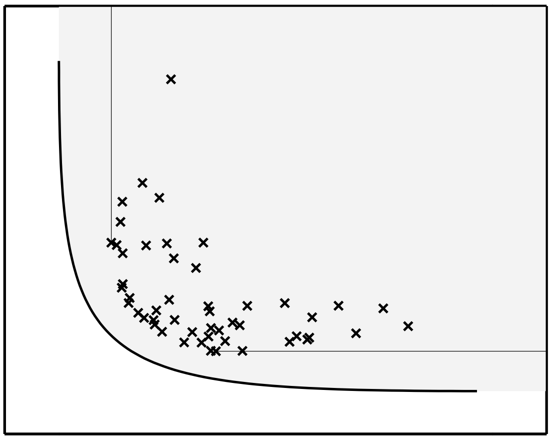


Figure 4.6: Dominated individuals in the 'tail regions' (behind the lines) of the NDS contain information about the full range of the POF.

are then examined to see if they fall into the tail regions. Then the tail regions are ranked in exactly the same manner as the main non-dominated set, except that for each region, one of the objective directions has been reversed. This results in a total population that stretches beyond the normal limits of the NDS. The individuals in the tails are, indeed, sub-optimal, but they may nevertheless contain information about where optimal individuals in as-yet undiscovered regions of the POF may lie.

The 'tail boxes' are quite small relative to the size of the NDS and the population. In tests, the tail boxes had 5%–10% of the linear dimensions of the NDS, and a corresponding proportion of the population of the group. Typically this meant that a tail contained one or two individuals. This is sufficient to aid the exploration of the ends of the NDS, and not so large that time is wasted on what are, essentially, sub-optimal points.

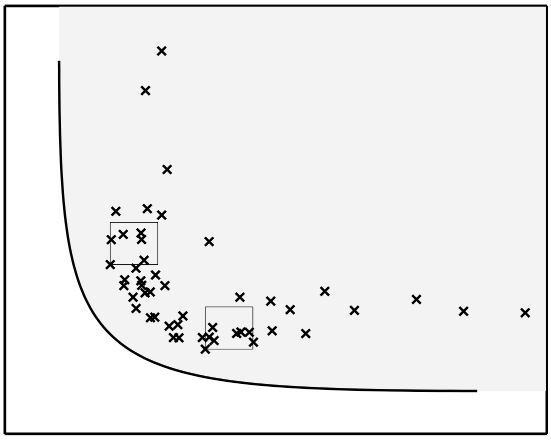


Figure 4.7: Preservation of tail regions. Individuals in the two boxes at the ends of the NDS are preserved, and ranked preferring ‘left and up’ (left box) and ‘right and down’ (right box).

Tail Preservation with More than two Objectives

Tail preservation has only been implemented for two objectives at this stage. Extending the method involves some complications. The most significant being that, though with two objectives the tail regions can be easily defined by two ‘boxes’, with any more objectives they become awkward to define regions ‘around the edge of’ a three or more dimensional NDS. At the cost of a little more computation, the problem can be easily resolved—an entire group can be ranked in all of the necessary directions. Any individual in the first rank of a ‘tail direction’ gets attributed to a tail, while all other points are treated as part of the main front. Clearly, the size of the tail regions must be limited, otherwise they will become larger than the main front, so acceptable limits for each objective should be imposed on individuals in tails.

The obvious problem with such an approach is the computation time required to rank entire groups several times—ranking is already expensive enough. Here, though, there is hope that the methods presented in Everson et al. [43] for rapid ranking of populations will make this a non-issue.

4.7.2 The Dominance Matrix

In this work a variation of Deb’s fast non-dominated sorting algorithm was used. The difference is that the determination of the dominance between pairs of individuals is decoupled from the ranking algorithm by use of a ‘dominance matrix’. The reason for this is simple: the NSGA is a generational algorithm, re-ranked at each generation, and so dominance relations must be determined for the entire new population at the same time as the rank. QMOO, however is not generational, many pairs of points will remain in the population for a long time. Thus it is better to determine the dominance relations between pairs incrementally as points are added and removed, while ranking is performed on the whole population.

An advantage of the dominance matrix approach is that the ranking algorithm used is decoupled from the population structures—the ranking algorithm needs only the dominance matrix as in input. This means that the ranking algorithm can easily be changed, however this was not exploited in this work.

A dominance matrix for n individuals is an $n \times n$ matrix D where d_{ij} is one if i dominates j and zero if it does not. Clearly, d_{ii} is always zero. D is generated incrementally, as shown in the following short example.

With a population of one, $D = 0$. A second individual, dominated by the first, is added, and thus, by inspection:

$$D = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}. \tag{4.7}$$

A third individual is added. It is compared to the first, and is found to dominate it. Now, there is no longer any *need* to compare the new individual to the second, as it dominates the first, and so must dominate the second. However, testing showed that with a low number of objectives, it was faster to compare every individual to every other, rather than 'short-circuit' the algorithm by using such inferences. Hence the final dominance matrix:

$$D = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix} \tag{4.8}$$

A separate dominance matrix is kept for every front (main and tails) of every group in the population. This limits the size of D, and makes both creating the dominance matrix and ranking faster, but means that an individual that leaves one group and then re-joins it later will be compared twice to the other group members. This, however, is very unlikely to happen.

The fast non-dominated sorting algorithm proceeds as described by Deb et al. [33]. The counts of the number of times that an individual is dominated are generated at the same time as the dominance matrix. Note that the dominance counts used by the fast non-dominated sort are already Fonseca and Fleming's ranking—demonstrating how much simpler their ranking scheme is.

4.7.3 Ranking Algorithm

The algorithm used to rank a group g, preserving its tails, is as follows:

1. For each member i of the queue of individuals to be ranked:
   (a) Add i to the ranker's data structures, allowing a dominance matrix to be created.
   (b) If, in adding i to the ranker, i is shown to have exactly the same objective function values as another individual remove i from the population.
2. Call the ranker to rank all members of the main front of g. Initially, this will be all members of g. The ranker should rank members of the population at least up to r_{min}, and ensure that at least G_{min} individuals have been ranked. Past these limits, the ranking should give individuals a rank of -1—indicating that they are so poor they should be removed from the population.
   This step may effectively be a 'no-operation' if the algorithm performed the ranking in 1.
3. Calculate the sizes and location of the 'tail boxes' using the span of the main front in objective space.
4. For each member i of g not in the first rank:
   (a) If i is currently in the main front, but falls in a tail box, remove it from the main front, and put it in the corresponding tail.
   (b) If i is currently in marked as in a tail, but no longer falls in that tail box (due to the new size of the main front found in 1) remove i from the population.
5. Call the ranker to rank the tails.
6. Remove members of g with a rank of -1 from the population

Thus, a group will never be reduced to less that G_{min} members by the ranking algorithm, its size being between G_{min} and the number of individuals required to ensure that there are at least r_{min} ranks.

4.7.4 Dominated and Non-Dominated Trees

Recently Everson et al. [43] published methods for a very rapid ranking of very large populations (they show results for up to 20,000 individuals—with the methods used in this thesis, populations are rarely larger than 200 individuals). The methods use 'dominated trees' and 'non-dominated trees' to structure a population, much like a heap used in a sorting algorithm.

Use of these methods would improve the performance of QMOO, and enable the use of larger populations, however, population size will also be limited for multiple-group problems by the computational requirements of the clustering algorithm. More significantly, the methods would make tail preservation with more than two objectives practicable, and make some thinning methods easier with more than two objectives.

The methods became available too late for use in this work, however, it is hoped that they can be incorporated in the near future.

4.8 Parent Selection

Early versions of QMOO had no parent selection scheme. It was assumed, given the high level of elitism, that if an individual remained in the population, it was sufficiently good to be a parent. Thus, no distinction

Rank	1	2	3	4	5
No. of individuals	9	12	8	11	10
Proportion	0.18	0.24	0.16	0.22	0.20
Cumulative Probability	0.18	0.42	0.58	0.80	1.00

Table 4.1: Selection by rank

was made between individuals of different rank when choosing which should be parents.

However in the Shanxi coke industry problem large populations were required in order to preserve sufficient diversity to converge to the full POF. Keeping a large population (and thus a large number of ranks), slowed convergence, because (say) a fifth-ranked individual had the same chance of being a parent as a top-ranked individual.

In order to remedy this problem, a method for parent selection was implemented. The method is necessarily based on rank, as objective function values do not give enough information to distinguish between individuals. The method was chosen for practicality and tunability, rather than for any theoretical reasons. Furthermore, using the tuning parameter, preference for better ranked parents can be turned off completely.

When the population contains several ranks of individuals, the proportion of each rank in the population is calculated, and then the cumulative probability of an individual being in rank r or better, as shown in Table 4.1.

Now, if a random number is chosen from a uniform distribution between 0 and 1, the corresponding rank is found, and a random individual is chosen from that rank, a selection mechanism that gives no preference to better ranks is obtained. However, if the random number is raised to a power greater than 1 before choosing the rank, a selection mechanism where parent preference can be controlled by the exponent is obtained.

The resulting parent selection mechanism is extremely simple and suitable for a multi-objective optimisation. However, it has no more theoretical basis than any of the other selection mechanisms presented in §2.8. This should not be viewed as a problem: the mechanism's advantage is that it has a tuning parameter, which can be experimented with, and which could probably be evolved with individuals.

4.9 Thinning

During initial work on QMOO emphasis was placed on the speed of the population analysis, and on the fair sharing of breeding opportunities between clusters, in the hope that the algorithm could 'cope' with large populations. Unfortunately, the ranking and grouping algorithms used are slow, and without controlling population size, especially when there are more than two objectives, the population will always grow until it becomes impractical to handle.

In order to resolve this problem new techniques for thinning the NDS with two objectives were developed. These methods not only keep the population size under control, they also encourage good coverage of POF, and continue to maintain selection pressure in entirely non-dominated populations.

The thinning methods are applied to each group individually, rather than to an entire population, so while one group may be close to a minimum viable size, another may be being actively thinned.

Without thinning, the size of the population in QMOO is controlled by two parameters—the minimum viable group size, G_{min}, and the minimum rank, r_{min}[5]. As many ranks are retained as necessary to ensure that the group has at least G_{min} individuals, and ranks better than or equal r_{min} will never be eliminated. r_{min}, as shown by tests, should nearly always be one. If r_{min} is one, once the number of non-dominated individuals has exceeded G_{min}, the entire group is non-dominated. At this point the dominance relation can no longer be used to provide selection pressure[6] and the size of the group can grow unchecked.

Once the group reaches a maximum manageable size, G_{max}, some non-dominated individuals must be removed. The individuals to be removed are chosen based on their location in objective space relative to the rest of the group. The best individuals to remove are those that offer little more information about the form of the POF (effectively, those that are in a crowded region in the NDS), and individuals that are far from the POF.

There exist methods for finding the first set of individuals, and with two objectives, these methods are almost trivial. However, the second class of individuals can only be guessed at, and the guesses that can be made are only really practicable when there are very few objectives.

Thus, we begin by discussing methods for thinning in with two objectives, and then go on to discuss how these methods might work with more.

4.9.1 Dominated Volume Thinning

One method for choosing points to remove from the NDS is based on removing points that add the least to the volume dominated by the NDS. Zitzler et al. [138] introduce the concept of non-dominated volume as a measure of the performance of an algorithm—the larger the volume dominated by the NDS generated by an algorithm[7], the better the algorithm. Clearly, the maximum possible dominated volume is that dominated by the POF. The non-dominated volume measure is discussed in greater depth in §5.2.

If non-dominated volume is a reasonable measure of the performance of an algorithm, it seems to logical to base a thinning method on removing individuals that contribute the least to the non-dominated volume. Figure 4.8 shows an NDS, and marks the volume that each individual contributes—effectively, the volume that would be lost if the individual was removed.

Much of the volume dominated by the NDS is not attributed to an single individual—the removal of a single individual would not result in a loss of the volume, and so most individuals only make a small contribution. The individuals at the edge of the NDS, though, contribute an infinite volume (thus assuring that they are never removed).

[5]and, of course, the number of groups N_g.

[6]Except to exclude most new individuals from entering the group, and in the rare case where a new individual dominates some part of the group.

[7]Or the smaller the volume it leaves non-dominated.

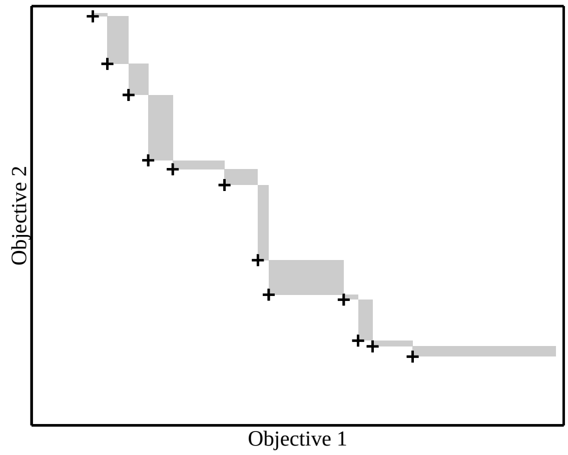


Figure 4.8: Contributions of individuals to dominated volume. Note that the two extreme individuals contribute an infinite volume (up and to the right).

Calculation of the volume dominated by each individual is easy when there are only two objectives. The members of the group can be sorted in one or other of the objectives, and traversed, calculating the volume dominated by each individual:

1. Set $0_{2,prev}$ to the second objective of the individual with the lowest first objective
2. Traverse the list of individuals, sorted by the first objective, starting with the second, and finishing before the last
   (a) Set O_1 to the first objective of the current individual, O_2 to the second objective.
   (b) Set $O_{1,next}$ to the first objective of the next individual.
   (c) Set the volume dominated of the current individual to $(O_{1,next} - O_1)(0_2 - O_{2,prev})$.
   (d) Set $O_{2,prev}$ to O_2.

The individual with the lowest contribution to the dominated volume is then removed, and the contributions of its two neighbours are updated. The process is repeated until the group size is less than G_{max}.

4.9.2 Quadratic Thinning

In quadratic thinning, we assume that a least-squares quadratic fitted though a subset of the population will be a reasonable approximation to a curve parallel to (but dominated by) the true POF. This is illustrated in Figure 4.9. The assumption is reasonable if the POF is reasonably smooth and the subset of the population is spread over a small part of the NDS. Such a subset could be found by clustering, as the SPEA. However, as the entire group is non-dominated, and because there are only two objectives, it can be sensibly ordered. Thus N_q consecutive points in this ordering can be chosen as the subset for interpolation. Here, N_q needs to be considerably more than the three points required to fit a quadratic, but only a small fraction of the group, in order to ensure that the quadratic interpolation is valid.

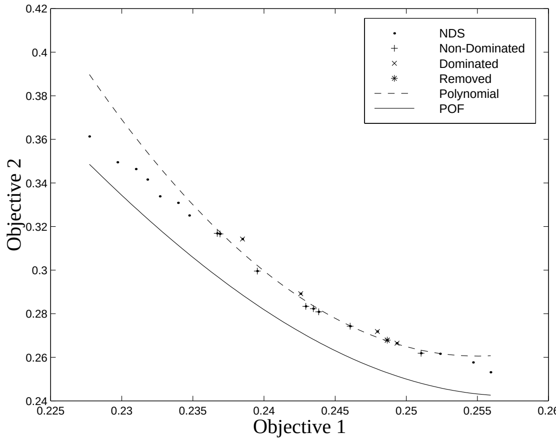


Figure 4.9: Quadratic interpolation in objective space through a subset of the population. The point removed is the dominated point with the closest neighbours. The quadratic is a reasonable, but not excellent, approximation to the POF.

Having chosen the subset and fitted the quadratic, we can separate the subset[8] into those individuals dominated by the quadratic, and those not dominated by it. The dominated individuals become candidates for removal—and so the technique applies selection pressure when the entire population is non-dominated.

The choice of the subset over which to interpolate and of which individual dominated by the quadratic to remove are made in a manner so as to encourage an even coverage of the POF, replacing the sharing features of many other fitness functions.

There are benefits to working with a non-dominated set of points in only two dimensions. Since the points can be are ordered, given a population of m points, we can find $m - n + 1$ subsets of n consecutive points, and measure the size of the sets in both objective dimensions. Given that all the subsets contain the same number of points, a good subset to choose in order to encourage a good distribution of points will be that which covers the smallest section of the non-dominated front.

The simplest way to measure the ‘spread’ of a subset is by a scaled Euclidean distance between the first and last points in the subset. Finding an appropriate scale is easy—the span of the group’s NDS can be used.

Choosing the smallest subset makes it reasonable to fit a quadratic to non-dominated front, and when a point is removed from that subset, it also reduces the density of the most densely populated part of the NDS, encouraging good coverage.

The same logic is applied to choosing which of the dominated points to remove from population. Here, the span of just three points—an individual dominated by the quadratic and its immediate neighbours—is used and the dominated individual with the nearest neighbours is eliminated.

The choice of the individual to remove takes no account of ‘how far’ the individual is from the fitted quadratic. One could imagine a scheme where the individual that is the furthest behind the quadratic is eliminated, hopefully adding even more selection pressure. However, the quadratic fit is only a rough ap-

[8]But *not* entire group—we do not *extrapolate* the quadratic.

proximation to the real POF, and calculating the shortest distance from an individual to the quadratic is awkward, and introduces scaling problems. Attempting to improve coverage, on the other hand, gives a clear indication of which point to remove.

The process described eliminates a single individual from the group. The process must be repeated until the population is a manageable size. The resulting algorithm has two parameters—how many points are necessary to perform a quadratic interpolation (typically 8 or 9 in two dimensions), and over how large a fraction of the group it is reasonable to assume the quadratic is valid (about a tenth in the case of just one group, and a quarter or a fifth if there are several groups).

4.9.3 Other Thinning Methods

A number of other thinning methods were tested, as a basis for comparison with the method described above, though one, 'random quadratic thinning', worked surprisingly well.

The methods used are:

- removing a random individual—which should result in poor POF coverage and slow convergence.
- removing the 'most crowded' individual, as measured by Euclidean distance between neighbours. This should result in even coverage of the POF, but will not help convergence much.
- An individual chosen in the manner of quadratic thinning, except that the subset for interpolation and the dominated individual to remove are chosen randomly. This should help convergence, but result in a poor coverage of the POF.

4.9.4 Thinning with More than Two Objectives

In optimisation problems with more than two objectives, the techniques described above for thinning lose some of their elegance and practicality.

Once there are three objectives, there no longer exists a simple ordering of individuals in the NDS—neither sorting by one or two of the three objectives can help—and so distances from each individual to every other ($n(n-1)/2$ distances, as opposed to $n-1$ when we can sort), must be calculated. Fitting low-order polynomials in more than two dimensions requires an exponentially larger number of points as the number of dimensions rises. Calculating the contribution to the volume dominated by a single individual becomes impractical—methods for such a calculation exist, but they are much slower than the two-objective case, and too slow to be used regularly.

On the other hand, if there are only three objectives, quadratic thinning is certainly practical, and possible. The subset to which the polynomial is fitted must be larger, and needs be found by a clustering method. It is difficult to find a measure of cluster size in order to find the most crowded cluster, and equally so to find the most crowded individual dominated by the quadratic. However other approaches, much like those used in the SPEA can be found. Molyneaux [89] implemented such a method (a three-dimensional quadratic fit using clustering) for work on the district heating problem.

The SPEA's truncation method works in higher dimensions, and is shown to be effective in Zitzler et al. [138]. The work of Everson et al. [43] shows that keeping the entire NDS in the working population is also effective.

4.10 Summary

QMOO is a steady-state, multi-objective, multi-modal evolutionary algorithm developed at LENI for optimising problems arising in energy system studies. QMOO uses an algorithm flexible queue-based structure which makes it easy to parallelise, and efficient when running in parallel, and which makes easy to add and remove complex analyses of the population.

QMOO, and its predecessor, the CPEA, use clustering methods to identify groups of individuals in decision variable space, and then let these groups evolve essentially independently of each other. The CPEA used a hierarchical clustering method to identify the groups, while QMOO, after some experimentation, uses fuzzy c-means clustering, which offers results of similar quality for much less computation time. Grouping allows QMOO to find multiple local optima, and gives it a higher chance of converging to the global optimum.

Evolutionary operator choice was used to solve all the problems presented in Chapter 6, and on most of the problems in Chapter 5 (the exception being some of the tests on EOC). EOC means that the best operator for a problem (or moment during a problem) can be used without any input or testing for the user. However, in some cases the best operator for a particular moment is not the best operator for overall convergence, as using fast operators can lead to a significant loss of diversity. EOC was initially developed as a method of including domain-specific operators in the Shanxi coal problem, but it has proved beneficial on other applications and some test problems.

Both QMOO's elitism, resulting from the algorithm's steady state design, and use of EOC result in diversity preservation problems. These are partly addressed by grouping, but careful choice of the set of mutation operators is also necessary. In order to improve the performance in the specific case of full POF discovery, NDS tail preservation was developed. Tail preservation was initially developed to improve NDS discovery in the district heating problem presented by Molyneaux [89], but tests show it also improves convergence on a range of test problems, with only a small loss of convergence performance on problems where tail preservation is inappropriate.

MOEAs spend much of their time with an entirely non-dominated population, and once this stage is reached, ranking methods are no longer effective at maintaining selection pressure. However, even if methods have recently been developed to rank very large populations, QMOO's grouping methods mean that the population size must be kept under control. Existing methods for thinning or truncating the NDS attempt to provide even coverage of the POF, but do not maintain selection pressure. Two methods for both maintaining selection pressure and providing even coverage are developed. However, though the methods work well and efficiently with two objectives, they are not easily transferred to problems with more objectives.

Chapter 5

Test Problems and Results

5.1 Introduction

The techniques for improving the convergence of QMOO were developed in order to improve performance on the 'real world' problems discussed in Chapter 6. However, they prove useful in the general case and to illustrate this they are tested here on a small suite of test problems.

The features tested are evolutionary operator choice (as compared to single crossover operators), a variety of thinning methods, and tail preservation, with two different tail sizes.

Grouping is *not* tested here—there is a lack of test problems available that would give meaningful results. Results showing that the CPEA always found the two optima of a deceptive problem were published in Molyneaux et al. [90], but the problem used only had two variables, and will not be presented again here. Despite the lack of problems that will test QMOO's multi-modality, real problems often perform better when groups are used. A number of examples are discussed in Chapter 6.

The general test procedure is relatively simple. A number of variants of QMOO, each with a different implementation of a feature, are tested on all the test problems. The tests are run for several different problem sizes (number of decision variables) and different population sizes. The tests are run 100 times, for 100,000 function evaluations[1], and convergence, as measured by normalised non-dominated volume (explained in the next section), is tracked throughout.

The average performance over 100 runs is presented in the results. By comparing the convergence curves from the runs for each of the versions of a feature, the effect of the different implementations on each problem can be seen.

5.2 Performance Measurement

Zitzler [137] introduces the concept of a 'volume dominated' approach for measuring an algorithm's con-

[1] Generally too many for the small problems and too few for the larger problems.

vergence to the POF. In [138] the measure is developed into a normalised non-dominated volume. Here, the volume between the non-dominated front and a 'worst' point is calculated. This volume is then normalised by dividing it by the volume between the worst point and a point composed of the best possible values in each objective—the 'utopia' point.

The resulting measure can be used throughout an optimisation to follow convergence. It is a very useful indicator of performance, as dominating a large volume implies both converging well to the POF, and discovering it in its entirety. Following dominated volume throughout an optimisation, as will be seen, provides a lot of insight into comparisons between different 'versions' of QMOO.

Calculation of the volume dominated by a population in two dimensions is trivial, the population is sorted in one of the objective directions, and then the algorithm walks through the population adding the contribution of the points. With more than two objectives, calculating dominated volume is more difficult[2].

In order that the non-dominated volumes can be compared to results of other researchers, the same utopia and worst points for each problem must be used. These points, provided by Laumanns [75], are listed with each problem, and are the same points used in the comparison of four algorithms by Zitzler et al. [138].

5.3 Test Problems

Many test problems already exist in the literature. Notably Van Veldhuizen and Lamont [131] and Deb et al. [33] proposed test suites[3], and Zitzler [137] proposed a number of test problems based on work by Deb [30]. Zitzler's work is particularly useful, as he made a large number of final populations for his test problems available on the Internet. This test data is used when comparing the performance of QMOO to other algorithms.

Three of the test problems used here to test the features of QMOO were used also in Zitzler et al. [138], and so performance on these problems is compared to the four algorithms tested there.

5.3.1 ZDT6

This problem was originally presented by Zitzler [137], based on work in Deb [30], where it was solved with 10 variables for 25,000 function evaluations. Since Zitzler made a number of final populations available for download, the problem has been widely used for comparison between different algorithms (for example Knowles and Corne [70]). Its form is:

$$\begin{aligned}\text{minimise} \quad \boldsymbol{f}(\boldsymbol{x}) &= (f_1(\boldsymbol{x}), f_2(\boldsymbol{x})), \quad \text{where} \\ f_1(\boldsymbol{x}) &= 1 - e^{-4x_1}\sin^6(6\pi x_1), && (5.1)\\ f_2(\boldsymbol{x}) &= g(\boldsymbol{x})\left[1 - (f_1(\boldsymbol{x})/g(\boldsymbol{x}))^2\right], && (5.2)\end{aligned}$$

[2] Zitzler et al. have made an algorithm for calculating dominated volumes in higher dimensions available on the Internet.
[3] Both, unfortunately named 'MOP', though they are different.

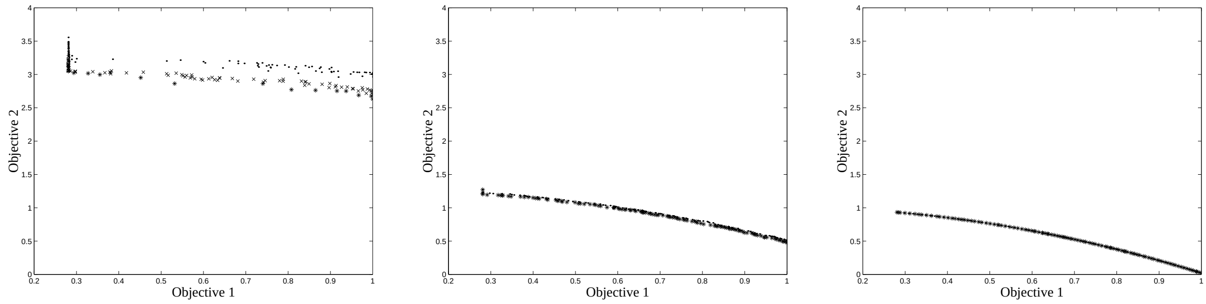


Figure 5.1: Convergence on ZDT6. Populations at 15,000, 40,000, and 100,000 function evaluations. Convergence is across the full POF. The difficulty lies in converging 'downward' to the POF.

$$\begin{aligned} g(\boldsymbol{x}) &= 1+(n-1)\left(\frac{\sum_{i=2}^{n} x_i}{n-1}\right)^{0.25}, \\ \boldsymbol{x} &\in [0,1]^n \end{aligned} \tag{5.3}$$

The problem's utopia point is $(0,0)$, and its worst point is $(1,10)$. The best normalised non-dominated volume possible for the problem is around 0.3025.

The first objective is a function of x_1 only, so the full range of values of f_1 is easily found, and further, the entire range of values of f_1 have a corresponding point on the POF, effectively making the problem a parameter study in x_1.

Convergence toward the POF is generally across all values of x_1 (and hence f_1) simultaneously while f_2 gradually decreases across the range. This is illustrated by a number of intermediate populations as shown in Figure 5.1.

Figure 5.2 shows that at the POF, all variable values except x_1 are zero. This means that the entire population is working toward the same solution. It seems unlikely that real MOPs are like this, where one would expect all variables to change across the POF. Some operators that can exploit this feature of the problem, by simply copying values from one individual or variable to another, and thus perform extremely well on this problem, while they are unlikely to do so on others.

Nonetheless ZDT6 is not an easy problem to solve, given the complexity of f_2, and the non-linear form of f_1.

5.3.2 'Enhanced' ZDT1

ZDT1 is the first problem from the same series as ZDT6. Its original form is:

$$\text{minimise} \quad \boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x})), \quad \text{where}$$

$$f_1(\boldsymbol{x}) = x, \tag{5.4}$$

$$f_2(\boldsymbol{x}) = g(\boldsymbol{x})\left[1-(f_1(\boldsymbol{x})/g(\boldsymbol{x}))^2)\right], \tag{5.5}$$

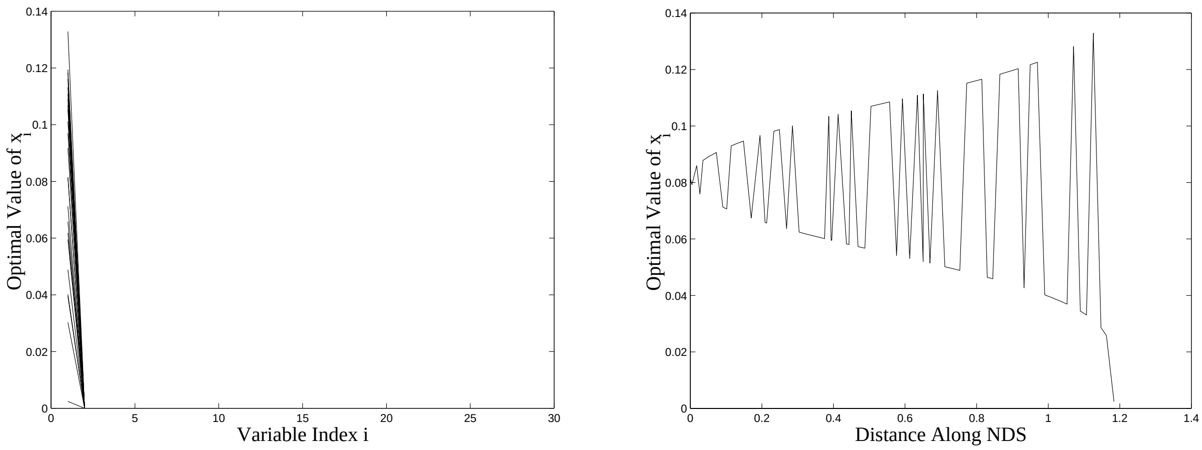


Figure 5.2: Solutions for ZDT6. Left: solutions for each variable (each line represents an individual). Only x_1 takes a range of values, all other variables are zero (making the graph a little unclear). Right: solutions along the NDS (each line represents a variable). In fact, the lines representing $x_{2...n}$ are all at zero. The only line on the graph shows the relationship between x_1 and f_1.

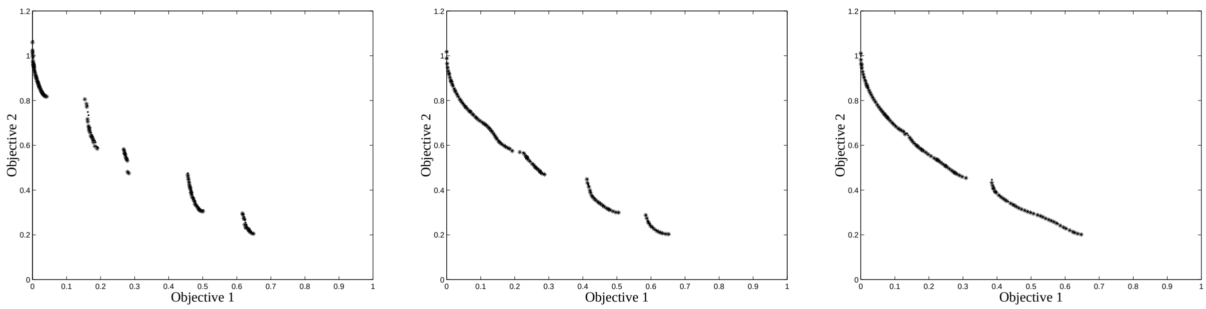


Figure 5.3: Convergence on ZDTE1. Convergence 'down to' the POF is easy, the problem lies in finding the whole POF, which stretches across $0 \leq f_1 \leq 1$. In this case, the POF is found no further than to 0.7 after 100,000 function evaluations. Populations at 10,000, 40,000, and 100,000 function evaluations.

$$\begin{aligned} g(\boldsymbol{x}) &= 1 + 9\frac{\sum_{i=2}^{n} x_i}{n-1}, \\ \boldsymbol{x} &\in [0,1]^n \end{aligned} \tag{5.6}$$

ZDT1 suffers from the same 'problem' as ZDT6—on the POF *all* of $x_{2..n}$ are zero. To counter this, the sum in (5.6) is replaced with:

$$s = \sum_{i=2}^{n} \left[x_i - \frac{1}{2}\left(1 + \sin\left(30\frac{i}{n}(1 + f_1)\right)\right)\right]^2 \tag{5.7}$$

to give an 'enhanced' ZDT1, ZDTE1. The optimal values for $x_{2..n}$ now vary in a non-linear manner with the variable index and f_1, as shown in Figure 5.4. Thus, though nearby individuals and variables will have similar optimal values, no individual can copy optimal values from another.

This makes ZDTE1 considerably more difficult to solve. f_1 is simpler than for ZDT6, meaning that convergence ‘down to’ the front is rapid, but the optimiser now has trouble finding the POF over the full range of values of f_1 (or x_1), as shown in Figure 5.3. Here, not only does the NDS not extend to the full length of the POF, but but there are ‘holes’ in the NDS where the POF is particularly difficult to find. The easy to find regions of the POF correspond to areas where several variables take the same values.

ZDTE1’s utopia point is $(0, 0)$, and its worst point is $(1, 10)$. The best normalised non-dominated volume possible for the problem is around 0.035.

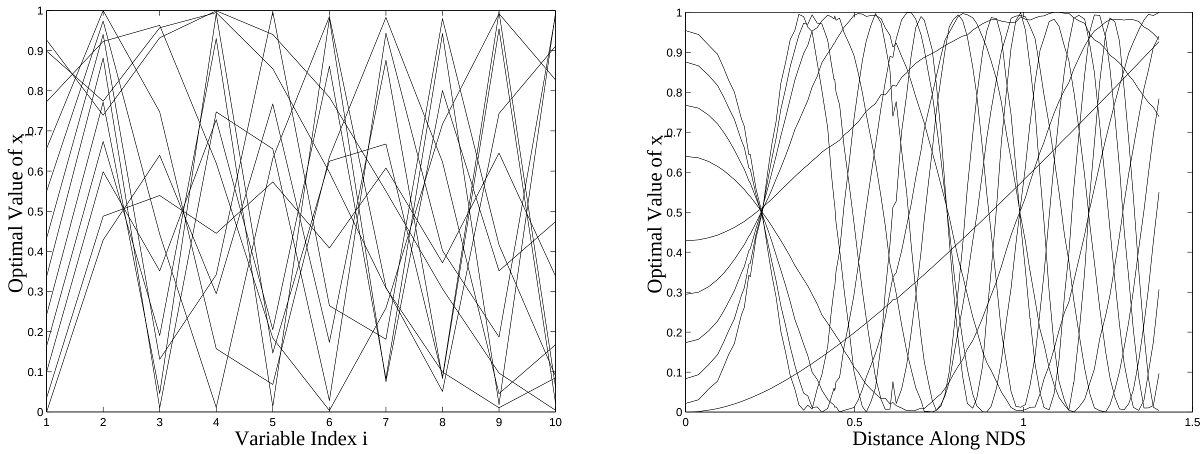


Figure 5.4: Solutions for ZDTE1. Left: optimal values for each individual vary for each variable (a line represents an individual in the NDS). Right: optimal variable values vary across the front (a line represents the values a variable takes across the NDS).

5.3.3 Quagliarella and Vincini’s Problem

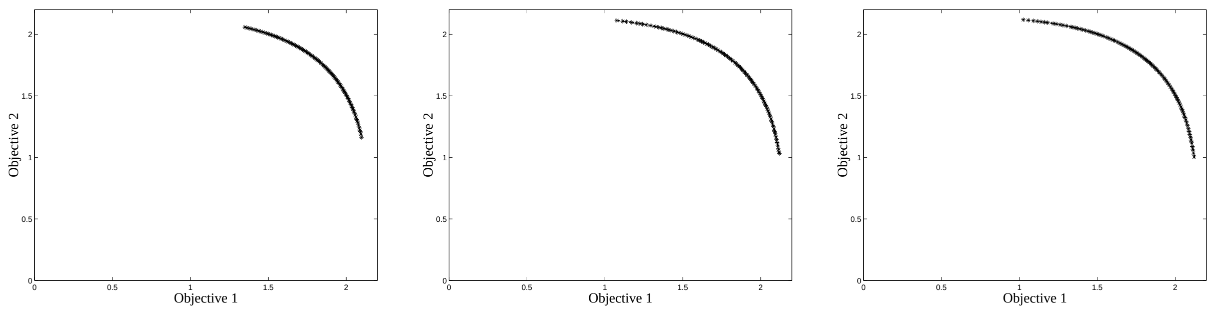


Figure 5.5: Convergence on Quagliarella and Vincini’s problem. Convergence to the middle of the POF is rapid, and then convergence continues by ‘growing’ the ends of the NDS. Note that the *true* POF extends to the left and bottom edges of the figure—well beyond the NDS discovered in this run. Populations at 10,000, 40,000, and 100,000 function evaluations.

This problem was first proposed by Quagliarella and Vicini [107], and later used by, among others, Zitzler et al. [138]. Its form is:

$$\begin{aligned}
\text{minimise} \quad \boldsymbol{f}(\boldsymbol{x}) &= (f_1(\boldsymbol{x}), f_2(\boldsymbol{x})), \quad \text{where} \\
f_1(\boldsymbol{x}) &= \left(\frac{1}{n}\sum_{i=1}^{n} x_i^2 - 10\cos(2\pi x_i) + 10\right)^{\frac{1}{4}}, \qquad (5.8) \\
f_2(\boldsymbol{x}) &= \left(\frac{1}{n}\sum_{i=1}^{n} (x_i - 1.5)^2 - 10\cos(2\pi(x_i - 1.5)) + 10\right)^{\frac{1}{4}}, \qquad (5.9) \\
\boldsymbol{x} &\in [-5, 5]^n
\end{aligned}$$

The utopia point for the problem is $(0, 0)$, and the worst point is $((20 + 6.5^2)^{0.25}, (20 + 6.5^2)^{0.25})$. The best non-dominated volume possible is around 0.55, but it is very rare that optimisers converge below 0.7.

Like ZDTE1, convergence to the POF is fairly rapid, but, as shown in Figure 5.5, the problem lies in discovering the full extents of the POF. As shown in Figure 5.6 there is a region in the middle of the problem where all variables are the same for an individual, and where optimal values change linearly with distance along the POF. However, at the 'edges' the picture changes completely, adopting a pattern much like that which will be seen in Kursawe's problem.

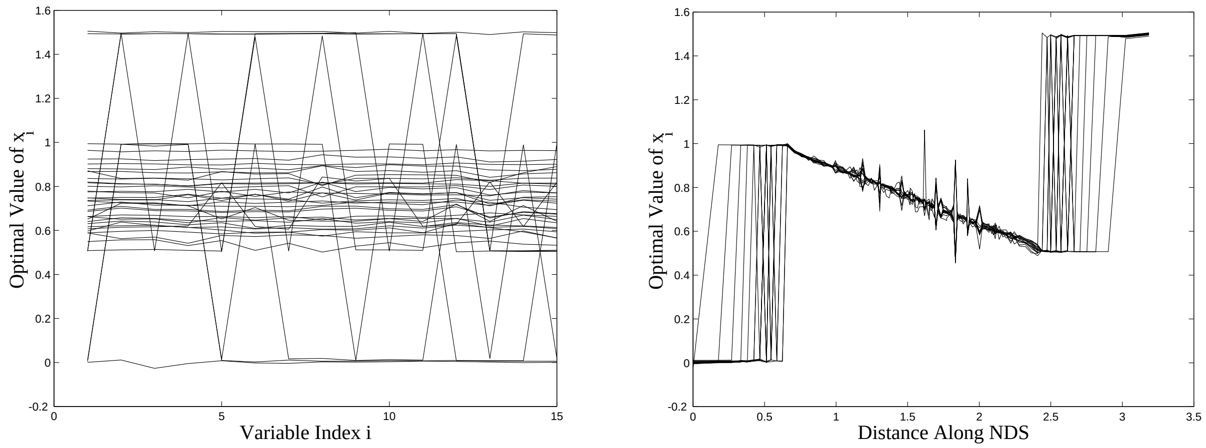


Figure 5.6: Solutions for Quagliarella and Vincini's problem. Left: each individual has *roughly* the same values for each of its variables. Right: all variable values change more or less linearly across the front.

The reason behind the problem's strange solutions can be seen when the two objective functions are plotted for one variable (Figure 5.7). The easy to find region lies between x values of 0.5 and 1, with objective function values as down to 1. After this, x values have to jump into the global optima for each objective at 0 and 1.5. Since at these points the *other* objective is worse than in the middle section, there is only a very small region in which non-dominated points can lie. Solutions in this region with several variables involve having some of the variables in each of the optima (and none in between), and finding an extremity of the POF means getting all variables into a global optima for one of the objectives.

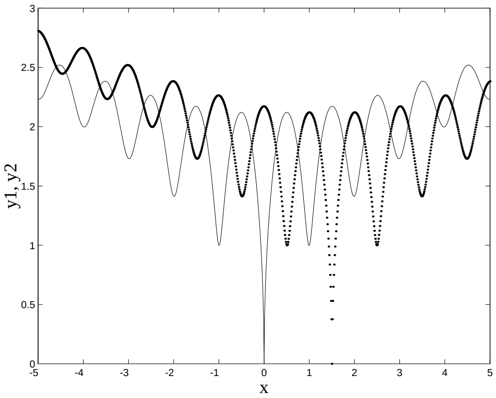


Figure 5.7: Quagliarella and Vincini's problem for one variable.

5.3.4 Kursawe's Problem

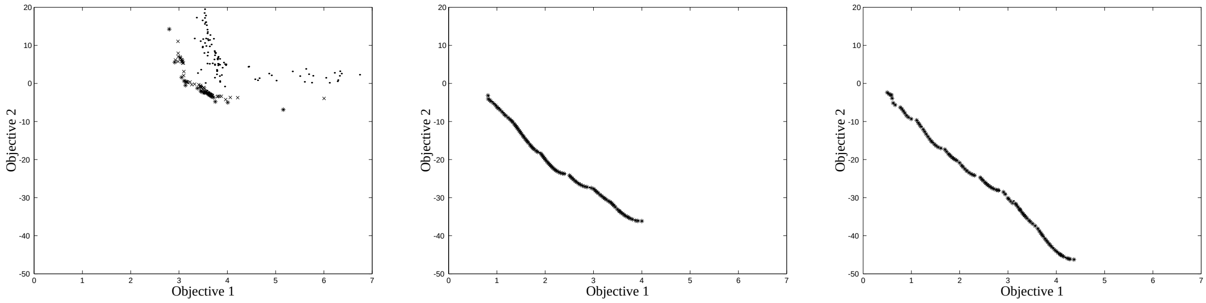


Figure 5.8: Convergence on Kursawe's problem. Convergence both across the POF and down to the POF is difficult. In this run the full span of the POF from $0 \leq f_1 \leq 5$ was not found. Populations at 13,000, 40,000, and 100,000 function evaluations.

This problem was originally defined in [74] as:

$$\begin{aligned} \text{minimise} \quad \boldsymbol{f}(\boldsymbol{x}) &= (f_1(\boldsymbol{x}), f_2(\boldsymbol{x})), \quad \text{where} \\ f_1(\boldsymbol{x}) &= \sum_{i=1}^{n-1} -10e^{-0.2\sqrt{x_i^2+x_{i+1}^2}}, && (5.10) \\ f_2(\boldsymbol{x}) &= \sum_{i=1}^{n} |x_i|^{0.8} + 5\sin(x_i)^3, && (5.11) \\ \boldsymbol{x} &\in [-1000, 1000]^n \end{aligned}$$

At least one group of authors [127] has used a different form, where the $\sin(x_i)^3$ term in f_2 is replaced by $\sin(x_i^3)$. Here we use yet another form, as used in [138] where:

$$f_1(\boldsymbol{x}) = \sum_{i=1}^{n-1} 1 - e^{-0.2\sqrt{x_i^2+x_{i+1}^2}}, \qquad (5.12)$$

The problem's utopia point is $(0, n-1)$, and the worst point is $(-n*3.5828, (n-1)*(1001^{0.8}+8.5828)-n*3.5828)$. The best non-dominated volume possible is around 0.0027, though optimisers rarely manage better than ten times this.

Kursawe's problem has many local optima, and so presents problems with diversity preservation problems that make it difficult for QMOO to solve. It also presents problems with discovery of the entire POF, as shown in Figure 5.8.

Variable values along the POF are a complicated pattern of switching between two values, as shown in Figure 5.9. On the 'left' side of the NDS, all variables are high (0), while on the 'right', they are all low (around -1.4).

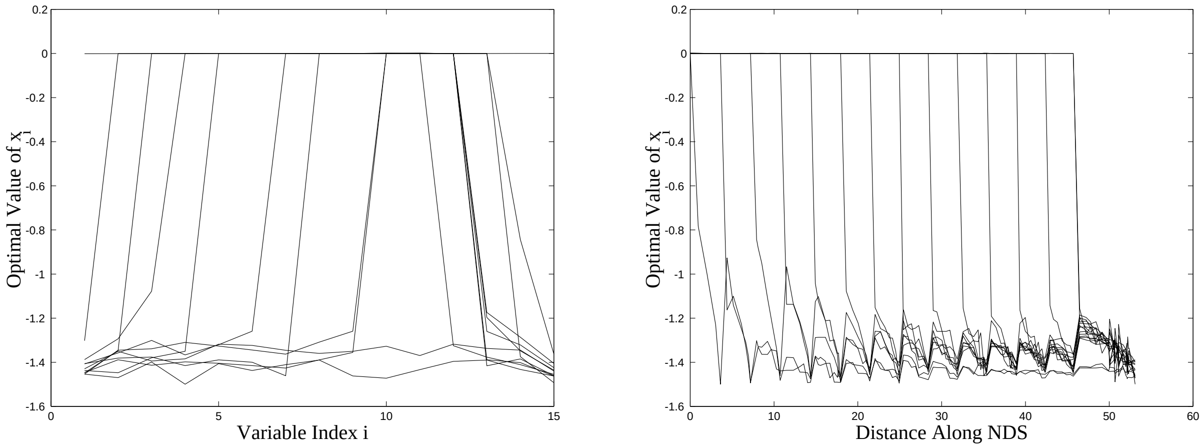


Figure 5.9: Solutions for Kursawe's problem. Left: each individual has variables switching between two modes. Right: at one end of the front, all variables are high, at the other, they are all low.

5.4 Tests

5.4.1 Test Conditions

The tests were run over a range of problem sizes and population sizes, and all parameters except for those being tested were fixed. The 'standard' test conditions are shown in Table 5.1.

5.4.2 Evolutionary Operator Choice

EOC was developed in order to control the use of 'problem specific' operators in the Shanxi coal problem. The performance improvements it offers also apply to some of the continuous test problems. In these cases, though, EOC is not controlling the use of special operators, but choosing between three 'standard' operators.

EOC is tested by solving the test problems with just one of the combination operators, and with EOC choosing between all three. Only the combination operators are tested—mutation operators are *always*

Test Problems	ZDT6, ZDTE1, QV, KURA
Mutation Operators	none, normal, uniform and global mutation (chosen by EOC)
Combination Operators	none, uniform, blend and linear crossover (chosen by EOC)
Population Limits	48, 64, 80, 96 individuals
Problem Sizes (ZDT6, QV, KURA)	15, 30 and 60 variables
Problem Sizes (ZDTE1)	10, 20 and 40 variables
Number of function evaluations	100000
Number of repeated runs	100
Thinning Method	dominated volume

Table 5.1: Experimental Conditions for Testing EOC

chosen by EOC. Experiments are performed in this manner simply to reduce the number of tests performed. The effect of EOC can be seen in the results on the combination operators.

The results show, in general, that EOC is sometimes better than any of the individual operators, and otherwise is second best or a very close third. Furthermore, no single operator has better performance across the full range of problems. Thus EOC is the 'best bet' if nothing is known about the problem beforehand. On some problems the finishing order of the single operators changes with problem size, while EOC's performance is generally more stable.

Full results are shown in §A.1.

ZDT6

EOC performs worst of all the problems on ZDT6, generally 'coming third' behind uniform and blend crossover, linear crossover always being last. As shown in Figure 5.10, initially, uniform crossover is the fastest to converge, but the rapid convergences stops—the algorithm 'bottoms out'—well above the lowest possible non-dominated volume, and then converges slowly. Blend crossover on the other hand, is initially slower than both uniform crossover and EOC. However, its convergence does not slow as much as uniform crossover, and it continues converging to the lowest possible non-dominated volume. EOC manages an early convergence nearly as fast as uniform crossover, but bottoms out even higher. It then continues converging at a slightly better rate than uniform crossover, but rarely overtakes it within the 100,000 evaluations.

Performance follows roughly the same pattern across the range of problem sizes. On the small problems, uniform crossover converges completely, as do EOC and blend crossover, and the 'finishing order' is given by early convergence rates. At large problem sizes, Uniform crossover and EOC bottom out, and blend crossover eventually overtakes them.

Blend crossover's performance is unaffected by population size, while uniform crossover improves with larger populations—both the early convergence, and the level at which it bottoms out. EOC's performance is even more affected by population size—at the largest population, it manages to overtake uniform crossover. Linear crossover's performance, on the other hand, gets worse with larger populations.

'Bottoming out' and performance improving with larger population sizes, especially with rapid early con-

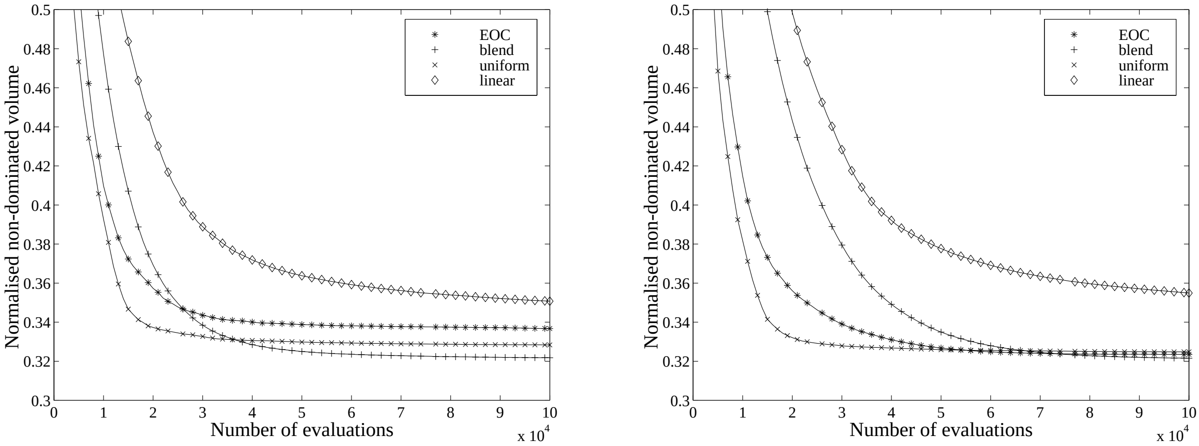


Figure 5.10: EOC convergence on ZDT6. Both problems with 30 variables. Left: with population of 48, the general pattern of convergence is clearly seen. Right: with population of 96, EOC 'just manages' to overtake uniform crossover.

vergence, suggest that uniform crossover and EOC are failing to preserve diversity on this problem. The effects are discussed further in the results for Kursawe's problem, which exhibits the same problems more clearly, and for which there are no questions about the form of the problem's solutions.

ZDTE1

EOC performs well on ZDTE1. EOC is consistently the best policy throughout the whole range of problem and population sizes. Not only does it converge to the lowest non-dominated volume at the end of every run, it often has the fastest initial convergence only being matched, and occasionally beaten, by uniform crossover—which, as on ZDT6, bottoms out and often finishes last.

Over 100,000 function evaluations, linear crossover nearly always finishes second on the smaller problems, though, because all operators are still converging in all the tests, it may well be overtaken later by blend crossover, which, though it has the slowest convergence for the large part of each run, continues converging at a higher rate.

For the largest problem size, however, patterns are reversed, and linear crossover sometimes performs worst—its early convergence is the slowest, though at low population sizes it does eventually overtake blend and uniform crossover. EOC performs best, again. Its initial convergence is faster, and if it bottoms out, it does so lower than the other operators, and its continued convergence is better.

All the operators are affected by the population size, though none in a very consistent manner. Even EOC does not consistently improve with a larger population size, and blend crossover, unaffected by the population size in ZDT6, sometimes prefers a smaller population, sometimes not.

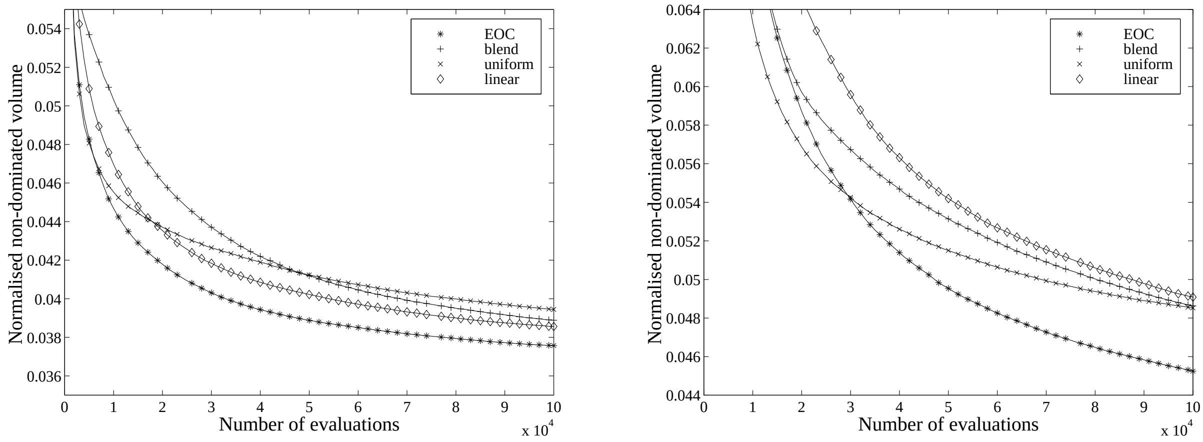


Figure 5.11: EOC convergence on ZDTE1. Left: 10 variables, population of 48: EOC converges fastest and longest, Linear crossover is second, but may be passed by blend crossover later. Right: 40 variables, population of 96: EOC wins again. None of the other operators are doing well, and linear crossover is worst on all counts.

Quagliarella and Vincini's Problem

The results for the Quagliarella and Vincini's problem are the most consistent over the entire range of problem sizes and populations, and all the results graphs seem to show the same picture at a variety of scales.

EOC is aways best, linear crossover is second, while blend crossover is a distant third, with uniform crossover not far behind. EOC is also the fastest early converger, with blend crossover converging second best at the start.

None of the methods converge much below a non-dominated volume of 0.7.

Kursawe's Problem

Kursawe's problem shows more clearly the same 'problems' that were encountered with ZDT6, and the patterns of convergence are much the same.

Very early on, blend crossover converges slower than uniform crossover and EOC, but very quickly these two bottom out, and blend crossover continues converging very quickly. Linear crossover converges slowest, but continues converging, and occasionally overtakes both uniform crossover and EOC. The general pattern is similar over problem sizes and population sizes.

At small population sizes, convergence starts to bottom out fairly early, while with larger populations, convergence bottoms out later, but continues at a lower rate, resulting in much the same final values over a range of population sizes.

What is happening? The symptoms are the same as for ZDT6—as is the probable explanation. Though

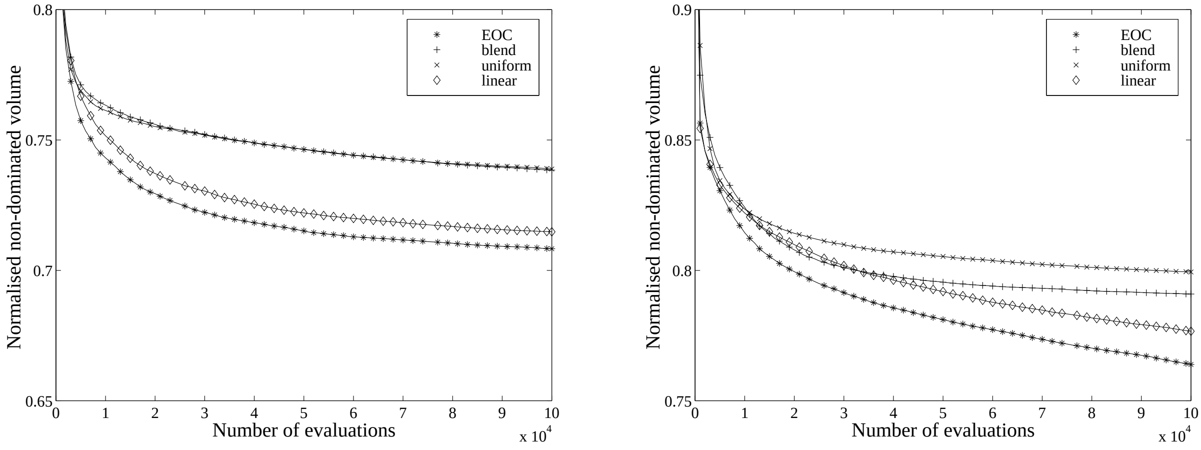


Figure 5.12: EOC convergence on Quagliarella and Vincini's Problem. Left: 15 variables, population of 48: EOC converges fastest and longest, Linear crossover is second, uniform and blend crossover look like they will never converge at all. Right: 60 variables, population of 96: exactly the same behaviour.

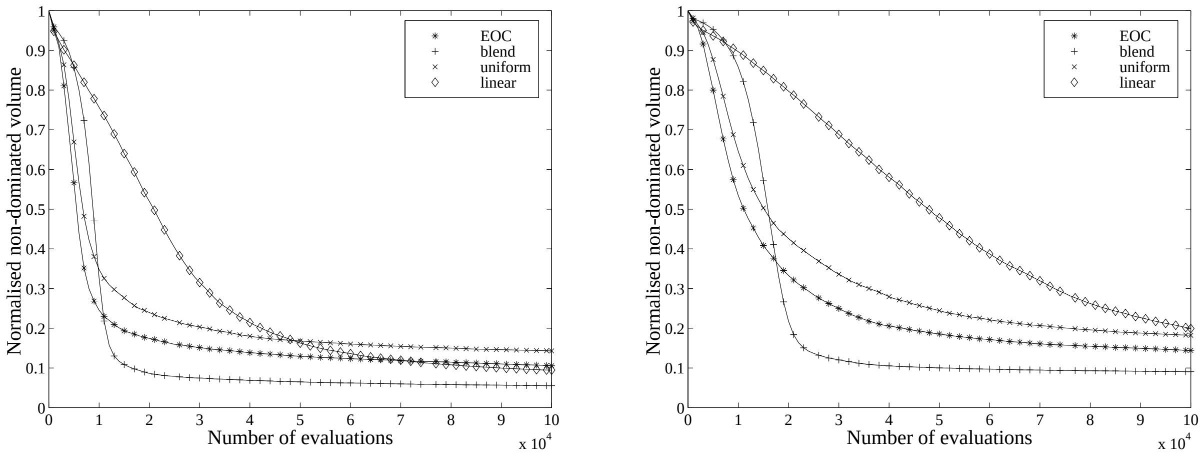


Figure 5.13: EOC convergence on Kursawe's problem. Left: 30 variables, population of 96: blend crossover is a clear best, Linear crossover is very slow, but continues converging to below both uniform crossover and EOC. Right: 60 variables, population of 96: a similar pattern is seen, except that linear crossover has not had time to overtake.

EOC and uniform crossover converge rapidly at first, in pursuing convergence, they cause the population to quickly lose the kind of diversity that is essential for solving this problem[4]. Blend crossover, and to a lesser extent linear crossover, on the other hand, preserve what is needed in the population, and continue to converge for much longer.

Why is EOC doing so poorly? Early in the run, EOC prefers uniform crossover, as it offers the best convergence, and even manages to outperform it. However, it soon finds itself in the same situation as uniform crossover, the population has lost what it needs to converge completely. Why, at this point, can EOC not

[4]Though we do not know exactly what that kind of diversity is.

switch to blend crossover and continue to converge? The problem is not with the operator[5], but with the state of the population. The diversity was lost earlier by the uniform crossover operator, and blend crossover can not 'bring it back'.

Mutation

Though EOC of mutation operators is not thoroughly tested here, brief tests illustrate the importance of each of the mutation operators, and the effect that mutation has on the performance of the crossover operators. Figure 5.14 shows the results of tests on Kursawe's problem.

Removing normal mutation has almost no effect on the convergence of any of the combination operators (the values are slightly worse). Less thorough testing has shown that normal mutation is important for other problems. Removing both uniform and global mutation has large effect on convergence for all operators except blend crossover. The effect of removing global mutation, while not as large as that of uniform mutation, can perhaps be taken as experimental validation of Rudolph's requirements for mutation operators.

Blend crossover is almost completely unaffected by the mutation operators, and on this problem, where it also performs the best, it would seem such a good and easy to use operator is far better than the other choices. However, it should not be forgotten that blend crossover is outperformed by other operators. Clearly, blend crossover shows that one can have a good combination operator that is insensitive to the mutation operators, but one can sometimes get better performance from other operators combined with the right mutation operators.

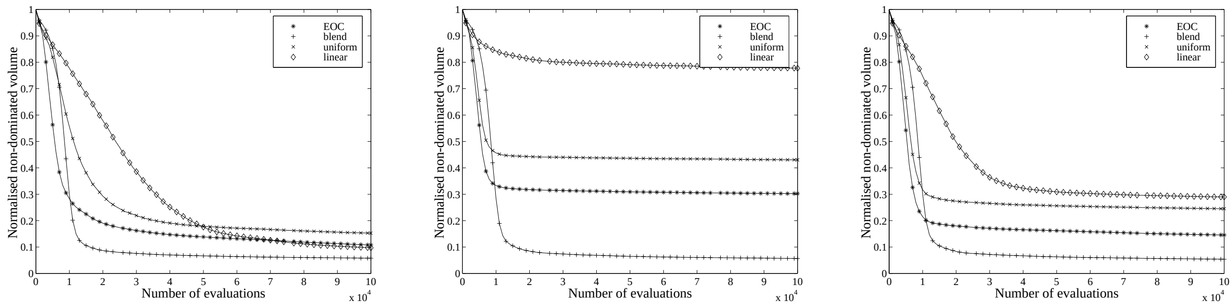


Figure 5.14: Removing mutation operators on Kursawe's problem. 30 variables, population of 96. Left: Removing normal mutation makes little difference to convergence for this problem. Middle: Uniform mutation, with a larger range, has a big effect on convergence, especially for linear crossover. Right: Global mutation also has a large effect on convergence. Note that blend crossover is almost entirely unaffected by the mutation operators.

Conclusions

EOC is not a panacea for all optimisation problems. The 'fastest' operator at a particular moment is not necessarily the best operator for full convergence to the optimum. Nonetheless, EOC provides excellent

[5] EOC does start to use blend crossover more at this point.

performance on some problems, and when it performs poorly it is rarely worse than a close third. Furthermore, it performs well on ZDTE1, this author's idea of what a 'real' problem might be like, whereas its poor performance on ZDT6 can perhaps be partly attributed to the problem's inconsistencies.

EOC does have the best performance over all the test problems, making it the best choice if nothing is known about the problem before optimising.

The brief investigation of mutation operators shows that they can have a large effect on performance. Given that no special attention has been given to the mutation operators, a through investigation may well yield even better performance.

5.4.3 Thinning

A number of thinning methods were tested and compared. Ideally the tests would have included a 'no thinning' option, where no non-dominated individuals would be removed. However, this proved computationally impractical, and it is only very recently that methods for managing large ranked sets have become available [43]. Testing of this option will be performed in the very near future.

In this work it is hypothesised that a good thinning method should improve convergence of an algorithm by maintaining selection pressure when ranking can no longer do so. To this end the dominated volume and quadratic thinning methods were developed. Here they are tested against a number of other methods, none of which are intended to be used as they are, but which are used here as bases for comparison.

The methods are:

- removing a random individual;
- removing the most crowded individual—the individual such that its left and right neighbours are the closest in scaled euclidean space;
- removing a random individual from a randomly chosen subset of the front, which is dominated by a quadratic interpolated through the subset. This is quadratic thinning without choosing the most crowded dominated individual.

Volume-dominated thinning was used as the standard thinning method in this work—in two dimensions it is very easy to implement. However, the tests show that quadratic thinning gives better results, and that random quadratic thinning sometimes gives excellent results, but at the cost of some convergence reliability. The other two methods generally perform worse than quadratic and volume dominated thinning. That random thinning performs reasonably well on some problems says more about the form of the problems than about the thinning method.

Full results are shown in §A.2.

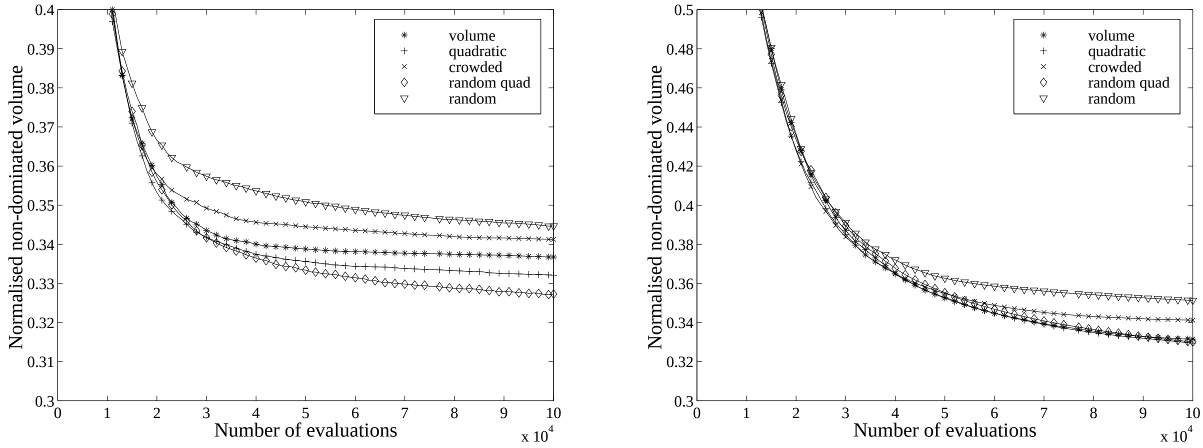


Figure 5.15: Thinning convergence on ZDT6. Left: 30 variables, population of 48: the finishing order remains the same over nearly all the problems. Right: 60 variables, population of 96: Volume-dominated thinning initially performs as well as quadratic, but quadratic thinning keeps converging for longer.

ZDT6

The performance of the four thinners on ZDT6 shows the same pattern that will be found in nearly all of the other tests: quadratic thinning is best, volume-dominated thinning is next best, crowded thinning is next, and random thinning is last. On ZDT6, however, random thinning does its best, coming only last, not a distant last. The reasons why will be discussed with the results for ZDTE1, where random thinning really does do badly.

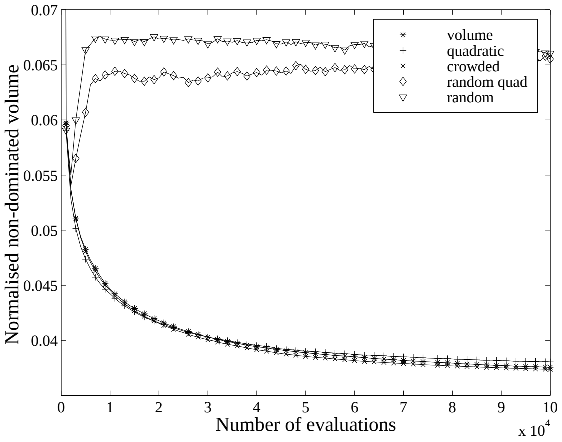
Random quadratic thinning also behaves as it does for the most of the problems: sometimes it performs the best on average by a large margin, other times, it does not perform so well, with few clear patterns in its performance.

The performance patterns for methods other than random quadratic thinning do not change much over the range of problem and population sizes.

ZDTE1

ZDTE1 proves to be the worst problem for both random thinning methods, while performance of all the other methods is similar—sufficiently so that there is no clear best. Crowded thinning is generally the worst of the rest, but on the small problem it performs best on two populations sizes. Quadratic and volume dominated thinning swap places over the range of problem and population sizes.

Random and random quadratic thinning's performance on ZDTE1 is extremely poor—the graphs show that once thinning starts, non-dominated volume actually increases. While ZDTE1 shares the same form as ZDT6, in the sense that the first objective is a function of x_1 only, and the second objective a function of all the other elements of $\boldsymbol{x}$, it differs in that the optimal values of $x_i \, \forall i = 2...n$ vary with x_1, and with i. Thus, every individual contains unique information about where the POF lies. For ZDT6, on the other hand, the optimal values of $x_{2...n}$ do not vary at all—they are all zero. Thus, removal of any point on the NDS from ZDT6 results in very little loss of information about the POF, while removing a random point from the NDS of ZDTE1 could result in the loss of hard-earned and important information.



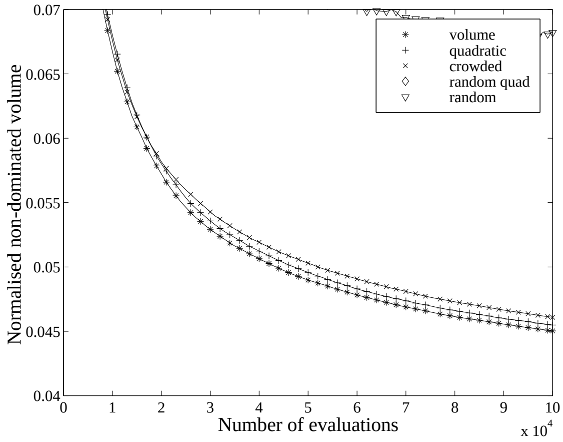


Figure 5.16: Thinning convergence on ZDTE1. Left: 10 variables, population of 48: random thinning performs appallingly, the others are roughly the same, crowded thinning being the best. Right: 40 variables, population of 64: results are similar, except that dominated volume thinning is best, while crowded is third.

The optimal values of $x_{2...n}$ do not vary randomly in ZDTE1, and so the information in an individual is useful to its neighbours, and to a certain extent, is also available in its near neighbours. Thus, the removal of points that have near neighbours results in the minimum possible loss of information. All three of quadratic, volume dominated and crowded thinning try to remove points with near neighbours one way or another, and all perform nearly equally well.

The two methods that prefer 'better' solutions, quadratic and volume dominated thinning, perform no better than crowded thinning. Though it is difficult to find the entire POF of ZDTE1, the single objective problem of finding $x_{2...n}$ given an x_1 is relatively easy. This means that removing points that have converged closer to the POF is not too serious a mistake—given information about the general area in which the optimum lies from nearest neighbours, convergence to the POF can be reasonably rapid. This also explains why random quadratic thinning performs as badly as quadratic thinning for ZDTE1—it conserves the wrong kind of points for this problem.

In ZDT6 on the other hand, converging to the POF with a fixed x_1 is relatively difficult. Thus, quadratic and dominated volume thinning perform better than crowded thinning, and random quadratic thinning performs well. This time, it is the distribution over the POF that does not need to be conserved.

One can imagine a 'ZDTE6', where both converging to the POF at a single point and converging to the whole POF are difficult, and here one would expect both extremely poor performance from random thinning, and a clear difference between crowded thinning and volume dominated and quadratic thinning[6].

[6] In fact, 'ZDTE6' was implemented and tested, but took a very long time to solve, even with small numbers of variables. This meant it was not chosen for the thousands of test runs performed here.

Quagliarella and Vincini's Problem

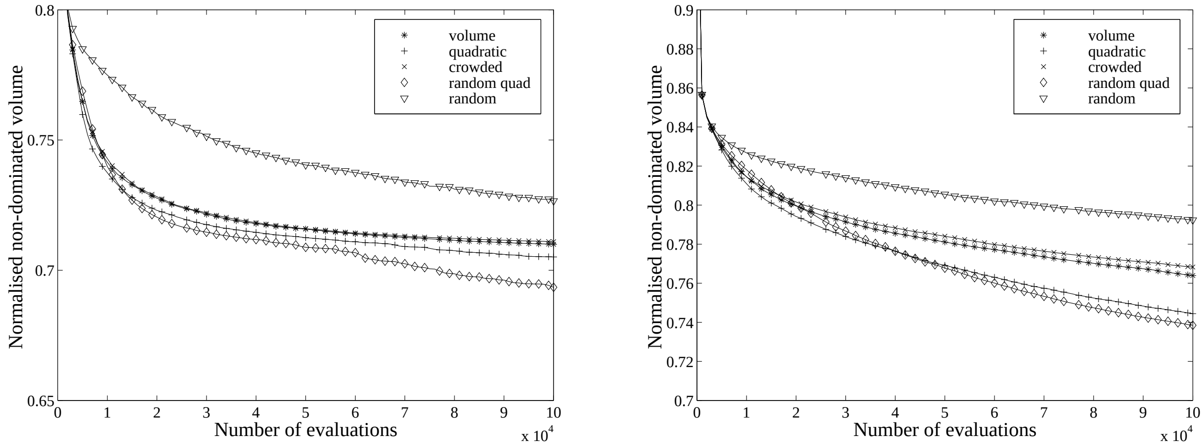


Figure 5.17: Thinning convergence on Quagliarella and Vincini's Problem. Left: 15 variables, population of 96: random quadratic thinning is best, while the others except random perform similarly. Right: 60 variables, population of 96: random quadratic is slightly better than quadratic, and both are better than volume dominated and crowded thinning.

Again, Quagliarella and Vincini's problem gives very consistent results over the whole range of problem and population sizes—even for random quadratic thinning. The 'finishing order' is always the same—random quadratic, quadratic, volume dominated, crowded, with random last—though on the small problem volume dominated catches or nearly catches up to quadratic thinning, but this is probably because quadratic thinning has almost completely converged. Random thinning performs better that it did on ZDTE1, though it is still by far the worst. The difference between the methods is large and clear.

Using the logic developed in the discussion of the results for ZDTE1, it could be concluded that in this problem, both converging to the whole front, and to a single point on the front, are relatively difficult. However, as this problem is slightly more like a real problem in that both objectives depend on all the variables, the situation is less clear-cut.

Random quadratic's excellent performance on this problem demands a little more investigation. Figure 5.18, shows the curves for volume dominated thinning and random quadratic thinning with a 90% error interval added. Worst case performance of random quadratic thinning is better than that for volume dominated thinning (though not for quadratic thinning), but its best case performance is far better. Thus, while the random quadratic thinning does perform better, this performance is comes at a cost of higher variability of convergence. As the benefit in this case is quite large, random quadratic thinning certainly deserves more investigation.

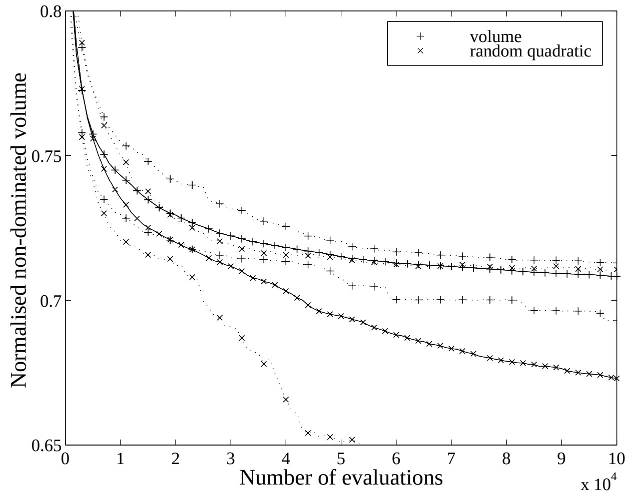


Figure 5.18: 90% limits for convergence for volume and random quadratic thinning. Left: 15 variables, population of 48: though random quadratic thinning is clearly better, its performance is highly variable.

Kursawe's Problem

Kursawe's problem is the only problem where crowded thinning outperforms dominated volume thinning, and on one problem, it even 'beats' quadratic thinning. Random thinning performs quite well on the large problem, though it is clearly worse on the smaller problems. Random quadratic thinning is remarkably well behaved, generally performing a little better than random thinning.

The good thinning methods do not change early convergence (this is not expected as they only start to work once the entire population is non-dominated), nor the level at which early convergence bottoms out. Instead, they have a positive effect on the rate of convergence after bottoming out, while in some cases random thinning nearly stops convergence completely after having bottomed out.

Across the range of problem sizes and population sizes patterns do not change much. For the larger problems, random thinning appears to do better, but this is only because the algorithms are still in the bottoming out region, and the better late convergence of the more effective methods has not yet had time to have an effect.

Conclusions

Paying attention to the thinning method always results in better performance than removing random points from the NDS, and quadratic thinning would seem to be the best choice of thinning method. It is a little surprising, given that performance is *measured* on non-dominated volume, that the method preserving the most dominated volume is not best. Crowded thinning performs well, given its simplicity—its performance probably coming from it being similar to volume dominated thinning—an individuals with two near neighbours that it does not dominate, clearly cannot contribute much to the dominated volume. The performance of random quadratic thinning is surprising. However it is a bit of a loose cannon across the range of prob-

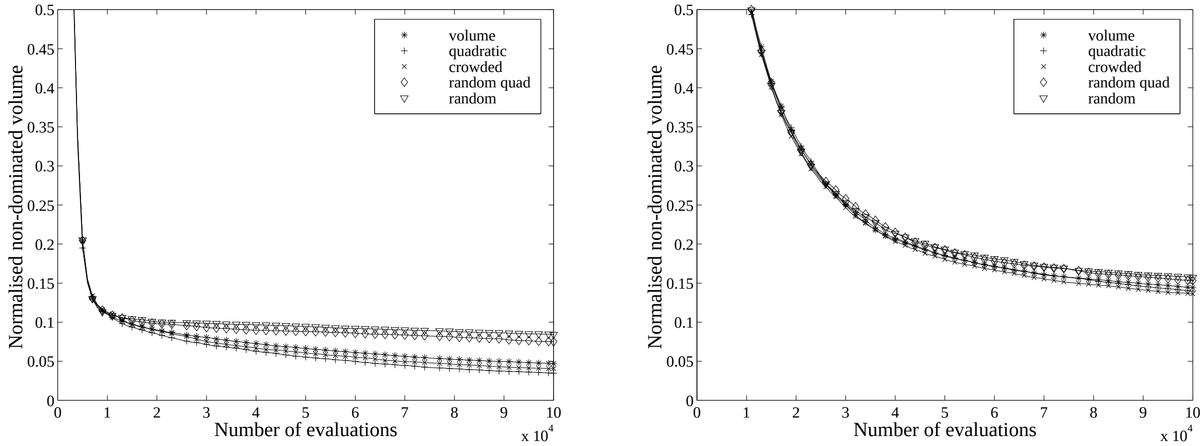


Figure 5.19: Thinning convergence on Kursawe's Problem. Left: 15 variables, population of 96: random thinning clearly performs worst, while crowded thinning outperforms volume dominated thinning. Right: 60 variables, population of 96: random thinning is as nearly as good as the other thinners. Crowded thinning is best (in this one case).

lems, and in some cases across the ranges of sizes and populations, mean that it is not the best method to use by default on a new problem. However, further study of its behaviour may lead to the development of better thinning methods.

The option that all non-dominated individuals be preserved was not tested, and the question remains as to whether the performance of the better thinning methods exceed that of preserving all individuals, or whether they are only approaching its performance.

5.4.4 Tail Preservation

Tail preservation was developed to improve convergence performance in the district heating problem, where discovery of the full span of the POF proved difficult. It also proved useful for the heat recovery problem, where it apparently improved convergence to low cost solutions. Three of the test problems in this chapter (ZDTE1, QV and KURA) pose problems for finding the full range of the POF, so here we test to see if tail preservation can improve convergence on these problems. The problem where the full range of the POF is easily discovered, ZDT6, is also tested, to see the cost of using tail preservation when it is not necessary.

Each problem is tested with 'tail box' sizes of 5% and 10% of the dimensions of the NDS, and equal proportions of the population, and compared to earlier runs without. Preserving the tail regions adds a few individuals to the population on average, so the population sizes were adjusted so that the average population over the 100 runs for each test was equivalent to the average population for the test being compared to.

Full results are shown in §A.3.

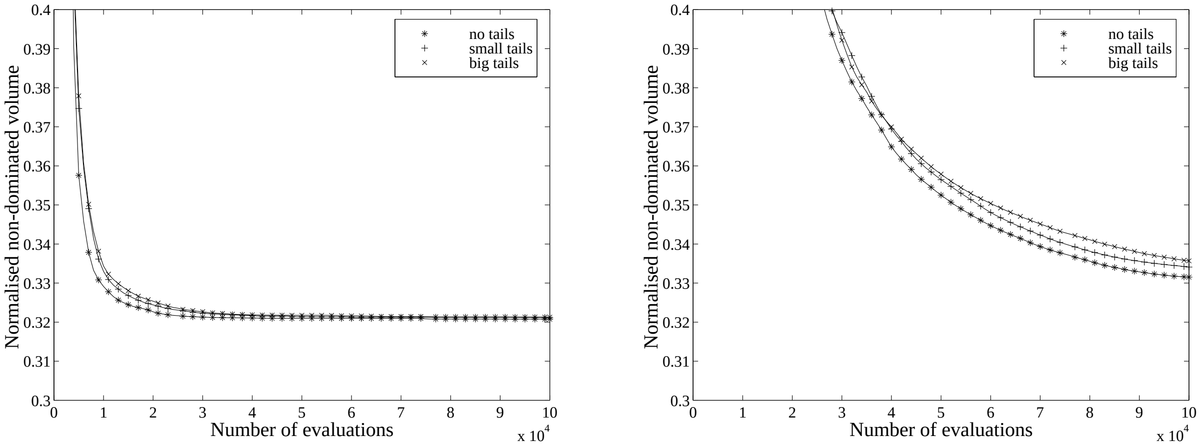


Figure 5.20: Tail preservation convergence on ZDT6. Left: 15 variables, population of 80: tail preservation slows early convergence, but has little effect on the effect on the final value found: it all depends on when you stop. Right: 60 variables, population of 96: here the cost in early stages is clearer, as none of the runs have converged completely.

ZDT6

As f_1 in ZDT6 is a function of x_1 only, and a relatively simple function at that, the entire range of values of f_1 are easily found. This means that tail preservation is ineffective on this problem, and worsens convergence performance, as it will preserve a few sub-optimal individuals that contribute no useful information to the population.

However, Figure 5.20 shows that the effect on convergence is not that large, certainly compared to the differences seen between different thinning methods, and different operators. The cost of a small tail region is not that high, and even the bigger region does not have a large adverse effect on performance.

The pattern does not vary much with either problem or population size. Larger problems are little more than ‘zooms’ on the smaller problems, and there is no clear pattern for different population sizes.

ZDTE1

Small tail preservation has a very small positive effect on convergence for ZDTE1. The improvement is of the same order as the difference between thinning methods, but much less than that between different operators. The larger tail region sometimes results in slightly better performance, occasionally doing better than small tails, and sometimes worse performance, with no clear pattern across problem population sizes Overall small tail preservation has a positive effect on convergence, while large tails seem to be neutral.

Given that ZDTE1 is a problem where discovery of the full POF is a problem, this seems surprising. Some of the ‘missing’ parts of the POF are not at its ends—the POF has holes in the middle, and tail preservation does not help with the discovery of these internal regions[7], but nor do they contribute a great deal to the

[7]Though it might if ZDTE1 was solved with grouping turned on, and each of the separate sections of the NDS was identified as

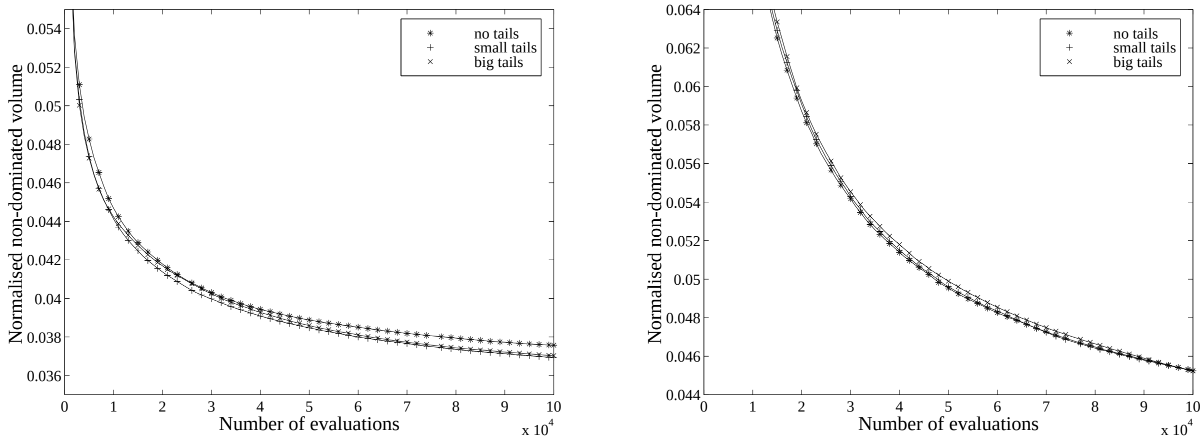


Figure 5.21: Tail preservation convergence on ZDTE1. Left: 10 variables, population of 48: tail preservation has a small positive effect on convergence over the whole run. Right: 40 variables, population of 96: there is very little difference between the two methods.

dominated volume.

Quagliarella and Vincini's Problem

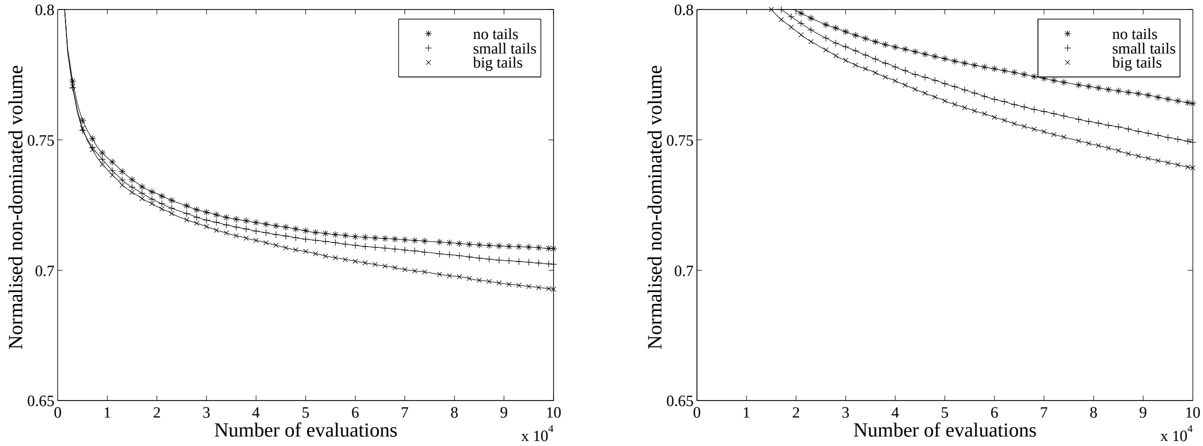


Figure 5.22: Tail preservation convergence on Quagliarella and Vincini's problem. Left: 15 variables, population of 48: large tail preservation offers considerable advantage. Right: 60 variables, population of 96: tail preservation is even better.

As always, the results for Quagliarella and Vincini's problem are very consistent. Large tail preservation is best, small tail preservation is next, and both are better than no tail preservation.

Given the form of the problem, and that the main difficulty for convergence is the discovery of the full POF, these results are not very surprising, but very reassuring. Tail preservation is performing well on problems

a group with its own tail regions.

were it would be expected to perform well.

The larger tail regions perform better than the smaller regions, in contrast to results for ZDT6 and ZDTE1, so tail preservation clearly offers better performance on certain problems.

Kursawe's Problem

Tail preservation offers a better performance improvement on Kursawe's problem than a good thinning method. Here, as for the other problems, neither the initial convergence rate, nor the level at which early convergence 'bottoms out' are affected much by tail preservation. However, the rate at which the algorithm continues to converge after bottoming out is greatly improved. Given that for EOC, the main difficulty with Kursawe's problem is the bottoming out, this is an excellent result.

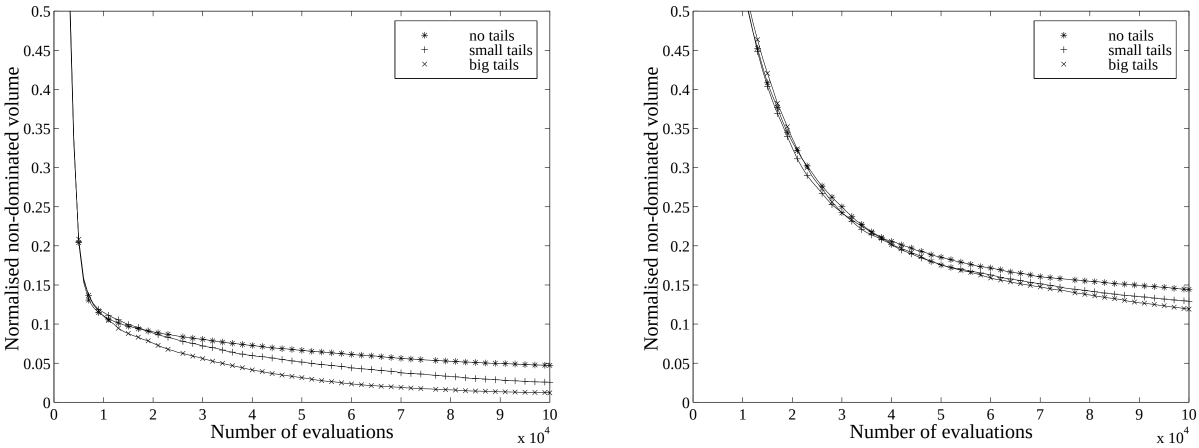


Figure 5.23: Tail preservation convergence on Kursawe's problem. Left: 15 variables, population of 96: tail preservation gives no advantage early on, and bottoms out at the same level, but critically, continues converging at a much better rate, with large tails performing best. Right: 60 variables, population of 96: in larger problems, performance is much the same.

At the lowest population size on the two smaller problems, tail preservation also has a slight positive effect on the level at which convergence bottoms out, though this is not as significant as the better late convergence. Across problem sizes, performance changes very little—the graphs of larger problems are little more than zooms of the smaller problems.

Conclusion

On three out of four problems, preserving small tails results in better performance than not preserving them. On two of the problems where discovery of the POF has been identified as a difficulty, performance is considerably improved by tail preservation. Even on a problem where tail preservation can clearly not help convergence, because discovery of the whole length of the POF is trivial, small tail preservation does not have a large negative effect on performance.

Large tail preservation offers large performance improvements on some problems, but over the range of problems, performs less consistently than small tails. Without knowing anything about a problem beforehand, preserving small tails is probably a good choice.

5.4.5 Comparison with Other Algorithms

QMOO's performance was compared to that of the SPEA in Leyland et al. [78]. Zitzler [137] kindly made final populations from the SPEA for a number of test problems available for download. These populations are the results of 30 runs of 25,000 function evaluations each the test problems ZDT1–6.

Test	Runs	E_w	E_b	E_{all}	E_{95}	Speed up
1	140	4339	4639	4694	6200	4.03
2	140	4342	4105	4384	5600	4.46
3	140	4166	4438	4476	5900	4.24
4	140	3842	3255	3884	6000	4.17
6	140	3398	3038	3406	5400	4.63

Table 5.2: Performance figures for QMOO and SPEA

Table 5.2 shows the number evaluations taken by QMOO to pass the fourth best population from the SPEA results (all but the best 10%) for problems 1–4 and 6.

E_w is the average number of evaluations for QMOO's worst individual to pass the SPEA's worst, E_b those required for QMOO's best to pass SPEA's best, E_{all}, those required for QMOO's worst to pass SPEA's best, E_{95} is the number of evaluations for 95% of QMOO's runs to pass both E_w and E_b, and the speed up is the number of times faster QMOO is based on E_{95}.

In Zitzler et al. [138] convergence curves based on non-dominated volume were published for the SPEA, PESA, NSGA2 and SPEA2. Among the problems treated were ZDT6, Quagliarella and Vincini's problem, Kursawe's problem. The problems were run 30 times with 100 variables for 1,000,000 objective function evaluations, and the average normalised non-dominated volume over the runs is presented.

Here we compare the convergence curves with curves obtained under the same conditions by QMOO. QMOO is run with what, from the test results above, would appear to be good 'default settings'. The default settings are a population of 96, EOC of mutation and combination operators, small tail preservation, and quadratic thinning. The performance of this combination is a fair estimation of the performance of QMOO.

As shown in Figure 5.4.5 QMOO's performance on Quagliarella and Vincini's problem is excellent. After 100,000 function evaluations the normalised non-dominated volume is 0.75. The best of the other algorithms (SPEA2 and NSGA2) get to this point at around 300,000 function evaluations. QMOO remains about 3 times as fast as the other algorithms, and converges to a value considerably lower than the other algorithms (though still far from the limit of 0.55).

Performance on Kursawe's problem is less spectacular, but still very good (Figure 5.4.5). QMOO converges

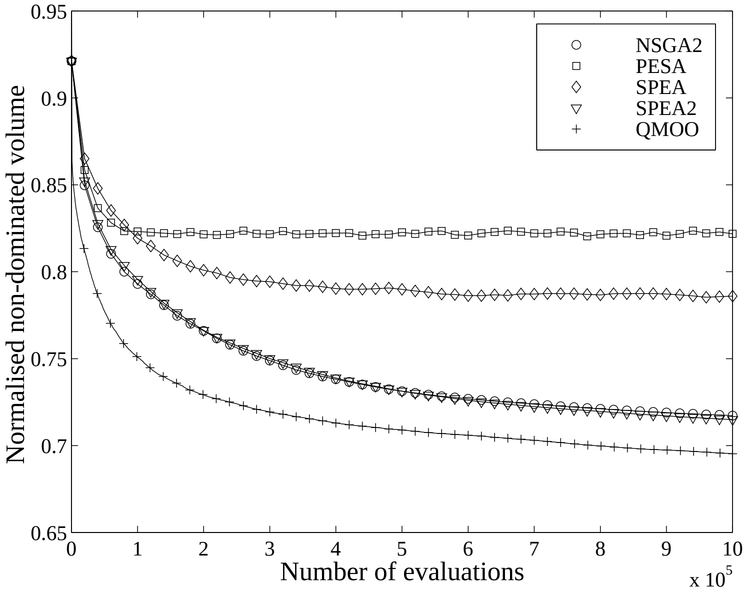


Figure 5.24: Performance of several algorithms on Quagliarella and Vincini's problem.

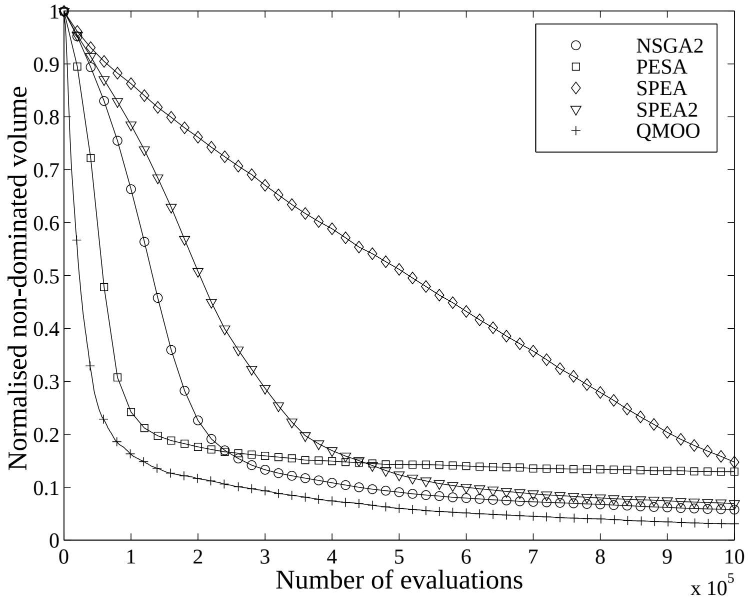


Figure 5.25: Performance of several algorithms on Kursawe's problem.

faster initially than PESA (more than twice as fast), and then manages keep ahead of NSGA2 for the rest of the run, converging to a slightly lower non-dominated volume.

In contrast to the other two problems, QMOO does not do very well on ZDT6 (Figure 5.4.5). Though QMOO initially converges very fast, it stops converging early, and finishes with a larger non-dominated volume than any of the other algorithms—this is the probably the penalty paid for using tail preservation on

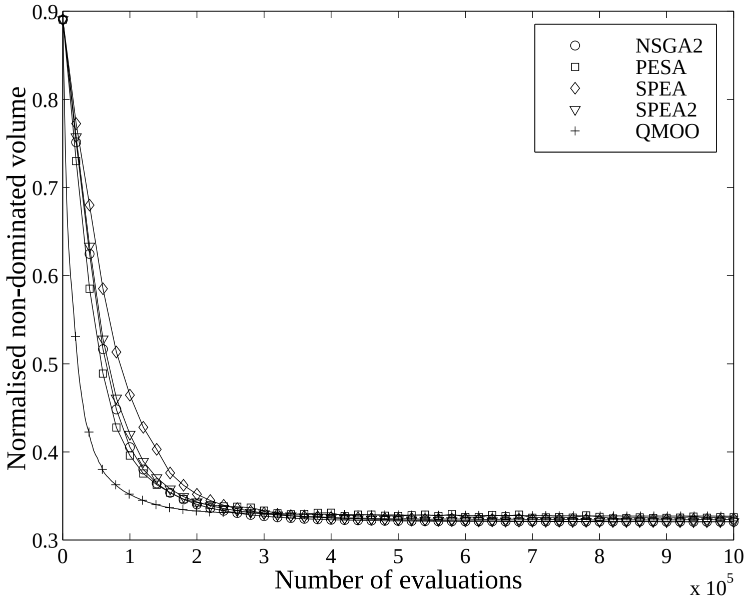


Figure 5.26: Performance of several algorithms on ZDT6 problem.

a problem where it is clearly not useful.

QMOO's performance on test problems is at worst not much worse than any other MOEA available today, and at best, it is much better. Quite likely, with some tuning (in this case, use of large tails and blend crossover where appropriate) QMOO could do better, but this runs against the initial goal that QMOO require very little tuning.

Some of these performance gains are attributable to the innovations in QMOO, but others are probably related to QMOO's extreme elitism. It would be interesting to compare algorithms using combinations of QMOO's features, and different methods of managing elite individuals, to see where the large performance gains lie, to learn about methods of managing elitism, and about the diversity preservation problems that seem to be caused by strong elitism.

5.5 Conclusions

The techniques developed for improving the convergence performance of QMOO generally do so on the problems tested, though not always over the full range of problems.

EOC is probably the least successful of the innovations, only offering performance advantages on two of the four problems. However, over the range of problems, it performs better than the single operators. A brief investigation of the mutation operators shows that the performance of some of the combination operators (including EOC) is depends strongly on the mutation operators. Given that the mutation operators were not particularly carefully chosen, a study may well lead to better performance for EOC.

Good thinning methods offer much better performance than less well chosen methods, and quadratic thinning results in better performance than any other method except random quadratic thinning over every problem. The thinning methods illustrate the difference between problems where converging ‘down to’ the POF is difficult (quadratic methods perform better) and where finding the whole POF is difficult (methods that prevent crowding are better). An investigation into the excellent but variable performance of random quadratic thinning is certainly in order, and may well lead to better thinning methods. The question as to whether thinning methods lead to better performance than preservation of the entire non-dominated set has not been answered, but will be in the near future.

Preservation of small tails leads to better performance on three problems, with very little performance degradation on the fourth. Preservation of larger tails does considerably better on two problems, with only slightly worse performance on the other two. Clearly, tail preservation is worthy of further investigation, if only to see if even larger tails perform well on some of the problems. Unfortunately, automatic control of tail region size will probably be difficult.

Comparisons of the performance of QMOO with other algorithms show that it is at worst it performs only slightly more poorly than other algorithms, and at best, it is much faster. Tuning of the algorithm would probably result in even better performance. However, whether QMOO’s performance is a result of the techniques discussed here, or of other, less testable, algorithmic choices such as the high level of elitism, remains to be seen.

Chapter 6

Applications

6.1 Coke Production in Shanxi Province, China

6.1.1 Introduction

The ‘Shanxi Problem’ is an optimisation problem treated in collaboration with Dr. Steven Kraines of the University of Tokyo, and partially funded by the Alliance for Global Sustainability[1]–funded ‘Technology–Energy–Environment–Health Chain’ project. The work is also presented in Leyland et al. [77].

Background

Reduction of inefficient use of resources is a critical concern for improving the sustainability of industry in transition economies. In China, the combination of market liberalisation—resulting in the appearance of town and village enterprises—together with rapidly changing price structures and government regulations, has fuelled tremendous changes in industrial practises. While some of the changes have been for the better—such as reducing the energy intensity of China’s industry [81]—uncertainty in future markets has also resulted in wasteful and short sighted resource use. The focus of this work, the coke-making industry, is pollution and energy intensive. Furthermore, a rapid shift to small scale ‘town and village enterprise’ coke-making plants that generally have less advanced technology may be exacerbating these problems [17].

Coke, a critical material requirement in the iron, steel and chemical fertiliser industries, is made though thermal decomposition of coal in the absence of oxygen. This process is known to be highly polluting, resulting in the emissions of toxic materials such as SO_x, NO_x, and polycyclic aromatic hydrocarbons (PAHs). Technologies exist for capturing the pollutants produced during the coke-making process, but they are not always cost effective for the reduction of emissions to acceptable levels. This has caused several developed countries to close down their domestic coke industries, which in turn has shifted the production of coke to the developing world. As a result, China now produces over 29% of the world coke output [106].

[1] The Alliance for Global Sustainability (AGS) is a collaborative research foundation formed by the Swiss Federal Institutes of Technology, MIT, and the University of Tokyo.

In the developing world, trucks are often used to transport low valued materials over long distances. The trucks are rarely well-maintained, and roads are often in poor condition. Thus, the energy consumption and pollution emissions from these trucks can contribute significantly to the overall energy and pollution intensity of the total industrial supply chain. However, transport pollution is often not considered in industrial planning.

Although measurements of PAHs and other airborne particulates indicate high levels of emissions around the coke-making plants, the highest emission readings are actually observed behind large diesel trucks on the local roads [105]. As diesel trucks transport much of the coal and coke for the many small scale coke-making plants in the Province, the pollution associated with the transport component of the coke-making supply chain may be a significant part of the total pollution load. Smaller plants tend to favour use of trucks because their production rates are neither large enough nor sufficiently predictable to enable them to make the long-term contracts necessary for rail transport.

Furthermore, data gathered on the monetary costs of transport versus plant operation indicate that transport costs can be a significant part of the total production cost of coke [71].

Recent government policy toward the coke-making industry in Shanxi Province is to shut down the large number of small, inefficient, and highly polluting coke-making facilities and replace them with bigger facilities equipped with more advanced technologies both for efficient coke-making and for pollution control [106]. The trend in the coke-making industry of shifting from many small plants to fewer, larger plants has important implications for the costs associated with transport. All else being equal, one would predict that a small number of large plants would tend to have a higher transport cost than many small plants, because in the latter case it is possible to better locate production where it is needed.

The Problem

The new Shanxi province government policy provides the background for ‘the Shanxi Problem’. The problem posed is: if you are going to replace a large number of small coke-making plants with a fewer large plants located on some of those sites, which sites should you choose and what size plants should you build at each site? In this work, the decision is based on construction and transport costs. However, the methods presented here apply equally well to studying trade-offs between costs and pollution, and between various pollutants. The methods developed in this work will be applied to such problems once robust models of pollutant emission from coke-making plants become available.

Resolution of the problem is performed on two levels. The ‘higher’ level is that of the two-objective construction and transport cost problem, viewing the trade-off between the two costs for each Pareto-optimal plant configuration. This level, clearly, is solved by QMOO. The second, ‘lower’ level, is that of finding, given set of plant locations and sizes, the optimal transport cost for transporting coal from coal mines to coke plants and from coke plants to coke consumers. In this work plant locations and sizes and mine–plant–consumer travel distances were taken from a geographic information system (GIS), and the problem was solved with a linear program (LP)—not the most efficient method, but one of the simplest to implement. In future work the GIS software will not only provide the distance information, but also solve the optimisation problem associated with transport, replacing the LP formulation.

6.1.2 Problem Definition

The base data for the problem is a list of locations and capacities of coal mines in the province, the locations and demands of the coke consumers, and the locations of existing coke plants. All this information is available in the Shanxi Energy Resource Atlas [121]. This location data was combined with the GIS database of roads and rails in Shanxi Province described in Kraines et al. [71], and the software package MapInfo, along with the distance calculation algorithm RCalc, was used to calculate road and rail distances between mines, plants, and consumers. This distance data, combined with data on road and track quality, was used to calculate transport costs in MRMB[2] per tonne-kilometre of transport between all points using data for road and rail transport under different road and rail conditions, as described in Akatsuka [1]. The calculations result in transport costs between each mine–plant and plant–consumer pair.

The other information needed is construction costs for plants of different sizes. We use plant cost data reported by Chen [16], for a 300 kT/yr 'non-recovery' coke-making plant, where plant construction cost was given as 13.5 MRMB. In order to account for plant scale effects, we assume that 10% of the plant construction cost is scale independent and that the remaining 90% is a fixed ratio of plant scale. The 300 kT/yr plant reported by Chen consisted of four coke oven batteries each with a production capacity of 75 kT/yr. Therefore, we assume that plants could be constructed in units of 75 kT/yr capacity and set the maximum plant size to be 450 kT/yr—6 coke oven batteries, approximately the size of the largest coke-making plants in operation in the Province in 2000 [129]. This results in construction costs for six different plant sizes, corresponding to 1 to 6 coke-making batteries (a last option being to not build a plant on a particular site). These costs are shown in Table 6.1.

Plant Sizes kT/yr	Construction Cost MRMB
75	4.4
150	7.4
225	10.5
300	13.5
375	16.5
450	19.6

Table 6.1: Plant construction costs for non-recovery type coke-making plants [16].

6.1.3 Problem Resolution

As mentioned earlier, resolution of the problem is performed at two levels. QMOO selects plant sizes at each location (effectively, a vector of integers from 0-6, the number of batteries to be constructed at each location), and requires from the 'objective function' the construction and transport costs for the configuration.

[2]Million RMB. One RMB—the Chinese unit of currency—is worth approximately 13 US cents

Calculation of the construction cost for the objective function is trivial following Table 6.1. Calculation of the transport cost, however, involves solving another optimisation problem. The configuration passed from QMOO to the objective function contains only information about plant sizes, and nothing about from which mine coal will be transported to which plant, and from which plant coke will be transported to which consumer. Clearly, there is a choice of transport routes that will result in a minimal transport cost for a particular plant configuration. This route can be found by solving a linear programming problem, where the variables to be found are the tonnes of coke and coal to be transported between each mine–plant and plant–consumer pair, the costs are the transport costs between the mine–plant and plant–consumer pairs discussed above, and the constraints are on the capacities of the mines and plants (which must not be exceeded), on the consumer demands (which must be exactly met), and also ensure that a plant produces an amount of coke that corresponds to the amount of coal that it uses.

The form of the LP is detailed in Leyland et al. [77], and is not further discussed here. We concentrate instead on the resolution of the multi-objective problem, how constraints are handled, and modifications that were made to QMOO in order to better resolve the problem.

Formulating the 'whole problem' for QMOO required a little work. Here, given a set P of possible sites, we must decide whether to build a plant at each site, and if so what size plant to build. The decision is based on both minimising transport costs—which will probably favour many small plants—and construction costs—which will probably favour fewer large plants. A decision-maker may be interested to see how the optimum configuration of plant sites and sizes changes based on the weight that is placed on transport costs versus construction costs. Conversely, the Pareto-optimal frontier will illuminate the trade-offs that occur between transport costs and construction of the coke-making plants for a range of sites.

QMOO solves the problem by choosing a plant configuration, and calculating its construction and transport cost objectives. There is a requirement that the capacities of the plants chosen by the EA be at least as large as the consumer demand for coke. However, as the problem is posed, there is nothing to stop the EA from generating solutions with insufficient plant capacity. We can indicate to the EA that such solutions are infeasible without causing any immediate problems. Unfortunately, in practise this results in the EA generating few solutions *near* the operating cost minimum because solutions just below the operating cost minimum have insufficient capacity to meet total consumer demand and are discarded.

An alternative approach is to assign a penalty function to solutions that do not meet consumer demand. There are many ways to set such a penalty function—normally, a penalty is added to the objective calculated for the infeasible system. In the Shanxi problem though, an infeasible transport problem cannot even be solved by the LP solver, and, as it is not possible to add a numerical penalty to an error message, another approach must be tried.

In the interest of maintaining a realistic problem framework, we set the penalty function as follows: any coke demand that cannot be met by the plants generated by the EA must be supplied by a fictitious plant outside the Province. So that factories inside the Province will be preferred:

- the construction of the 'out of Province' plant must be paid for at a higher cost than an 'in Province' plant;

- the cost per tonne of coke production for the ‘out of Province’ plant is higher than for ‘in Province’ plants;
- transport from any mine to the ‘out of Province’ plant and from the plant to any consumer is more expensive than any ‘in Province’ route.

With such an approach, the problem can be solved without constraints, and solutions that have a production capacity only slightly larger than the consumer demand are easily found.

Thus, the method for calculating the two objective functions is as follows:

1. The EA supplies a set of plant sizes to the objective function.
2. The total plant capacity is checked to see if it is sufficient; if not, the capacity of the ‘out of Province’ plant is made sufficient to meet the shortfall.
3. The construction cost of the plants is calculated (including the out of province plant if necessary), and the total value is kept as objective 1.
4. The transport LP is run to find the optimal transport cost for *this* set of plants, and this cost is retained as objective 2.

The ‘penalty factor’ on the ‘out of Province’ plant can be quite low—in this work, the ‘out of Province’ plant was set to be only 5% more expensive than the worst ‘in Province’ plant.

6.1.4 Testing

For testing purposes, a ‘reduced problem’ of 10 mines, 10 plant sites and 10 consumers—hereafter referred to as the 10x10x10 problem—was modelled. This problem was used to check and fine-tune the problem formulation, and to test the convergence rate of QMOO against the Struggle GA.

The 10x10x10 problem is sufficiently small that several optimal points can be found by observation. If all plant sizes are set to the maximum (450 kT/yr), and the transport LP is solved, the minimum transport cost, 64.9 MRMB/yr, is found. However, though all the plants are defined to be of maximum size, they are not all running at full capacity. Their production rates are lower, and their actual size can be lowered without increasing transport costs—as shown in Table 6.2. The solution where the size of each plant is reduced so that it has just enough batteries to meet the production rates given by the transport LP gives the minimum construction cost needed to obtain the minimum transport cost—59.4 MRMB. Alternatively, the minimum construction cost that will meet demand can be found. Here, plants with the lowest construction cost per kT/yr (450 kT/yr) are built until there is only a small remaining demand, for which the cheapest possible plant is built. In the 10x10x10 problem the total consumer demand is 864.8 kT/yr, for which two 450 kT/yr plants must be built, with no need for a smaller plant. The corresponding plant cost is 39.2 MRMB. Any of the possible combinations of locations for these two plants will have a minimum construction cost. Since, in this case, there are only $\frac{10!}{8!2!} = 45$ combinations, it is possible to calculate the transport costs of all locations for the two plants to find the minimum transport cost at minimum construction cost—85.5 MRMB/yr. These results, and the corresponding plant capacities are summarised in Table 6.2.

Plant	1	2	3	4	5	6	7	8	9	10	Trans. MRMB/yr	Cons. MRMB
Min. Trans. Production Rate (kT/yr)	88	0	44	33	450	124	0	5	119	0	64.9	-
Min. Trans. Capacity (kT/yr)	150	0	75	75	450	150	75	75	150	0	64.9	59.4
Min. Cons. Capacity (kT/yr)	0	0	0	450	450	0	0	0	0	0	85.5	39.2

Table 6.2: Plant capacities, transport and construction costs.

There is a considerable difference between the minimum transport cost and minimum construction cost solutions—in each case 20 MRMB or 20 MRMB/yr. In other words, a potential coke-making investor could chose to allocate up to 20 MRMB between plant construction and transport costs when deciding where and what size to build the coke-making plants. The actual plant siting configuration and transport flows are shown in Figure 6.1.

Although it appears that there are considerably more transport flows in the map corresponding to the minimum transport cost solution, in fact the addition of the many small plants scattered fairly evenly through the Province reduces the actual quantity of transport in tonne-kilometres and thereby the transport cost.

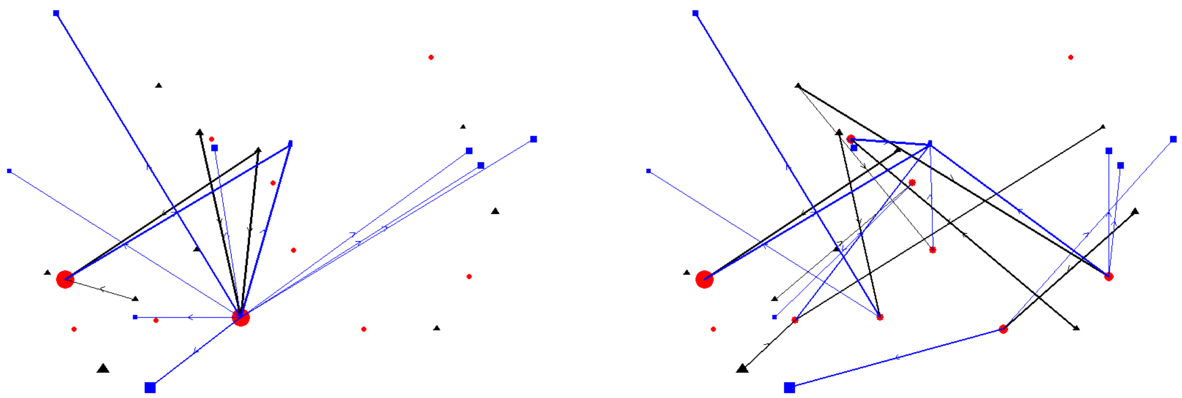


Figure 6.1: Plant sitings for minimum construction (left) and transport (right) cost. Circles represent plants and their sizes (the smallest circles are unused sites), triangles are mines, and squares are consumers. Line thickness indicates the transport volume.

These results, though interesting, were *not* generated by QMOO. However, they can be used to test its convergence. For this purpose, QMOO was run repeatedly, to see how many objective function evaluations were needed to find the following three solutions:

1. the minimum construction cost for minimum transport cost solution;
2. *any* minimum construction cost solution;

3. the minimum transport cost for minimum construction cost solution.

During this process, QMOO was altered in order to improve convergence. Significantly, evolutionary operator choice was developed (§4.4.1). The Shanxi problem is quite different from optimisation problems solved earlier at LENI. All the decision variables are integers, and finding a solution is a combinatorial, rather than continuous, optimisation problem. Early versions of QMOO used only blend crossover, as did the SGA, which is more appropriate for problems with continuous decision variables. Blend crossover can be modified for integer problems, and this approach was tried. Uniform crossover, which handles integer variables simple, was used in separate test runs. A number of mutation operators that should aid convergence were also tested. These were:

- INC : an increment/decrement that operates on a single variable;
- SWAP : an operator to swap two variables: although the construction cost is insensitive to the position of a plant, minimising transport cost depends on having the right size plant in the right place;
- COPY : an operator to copy a variable into another—a 'complement' to the swap operator;
- AGG : an aggregation operator. This operator takes two variables, and, where possible within limits, sets one of the variables to the sum of the two (or the upper limit) and the other to zero (or the remainder). This operator aids the discovery of low construction cost solutions, where several large plants are best. A similar operator was used in Borgarello et al. [9], for a quite different problem, with nonetheless, a similar structure.

To best solve the problem, each of these mutation operators should be used from time to time, and sometimes, no mutation should be used at all. In most EAs probabilities are assigned to the use of each operator by the user. Here, the EOC approach was developed and used.

In order to demonstrate the effectiveness of the approach developed, the test problem was solved a number of times with different operator combinations, to see how many function evaluations were required to find each solution. The operator combinations used were:

- 'integer' blend crossover, with EOC of INC and no mutation;
- uniform crossover with EOC of INC and no mutation;
- EOC of blend, uniform, and no crossover, as well as of INC, SWAP, COPY, and no mutation;
- EOC as above, with the AGG operator.

In the last case, the AGG operator is added separately from the other mutation operators. It has a large effect on convergence performance, and it is interesting to separate this effect.

For each of the operator sets, QMOO was run 48 times, for up to 5000 function evaluations. Only EOC with AGG managed to find all the solutions on every run (which it did comfortably). BC and UC only managed to find the first solution (the lowest transport cost) reliably, but both found a minimum construction cost once out of all the runs. EOC without AGG did a little better, finding a minimum construction cost several times, and even managing to find the lowest transport cost for minimum construction cost once. Table 6.3 summarises the results. For each operator/solution combination, the success rate is shown, then the third best time to find that solution, the mean time to find the solution, and the third worst time.

Operators	Problem	Success	95% low	Mean	95% High
BC, INC	1	100%	730	982	1232
BC, INC	2	2%	-	2131	-
BC, INC	3	0%	-	-	-
UC, INC	1	100%	773	1055	1343
UC, INC	2	2%	-	4460	-
UC, INC	3	0%	-	-	-
EOC, INC, SWAP, COPY	1	100%	822	1134	1571
EOC, INC, SWAP, COPY	2	17%	-	2693	-
EOC, INC, SWAP, COPY	3	2%	-	4101	-
EOC, INC, SWAP, COPY, AGG	1	100%	864	1284	1773
EOC, INC, SWAP, COPY, AGG	2	100%	380	802	1293
EOC, INC, SWAP, COPY, AGG	3	100%	782	1434	2263

Table 6.3: Function evaluations required to find solutions to the 10x10x10 problem.

The results for EOC with AGG are clearly better for solutions 2 and 3 than the other combinations, as was hoped in the design of the AGG operator. This comes at a small cost for the time to find solution 1.

The better performance of EOC and AGG can mostly be attributed to the agglomeration operator, and not to EOC itself. However, on average, the agglomeration operator is used less than 20% of the time[3], and the other mutation operators are also used. Thus, without EOC, the user would have to specify the percentage of the time that the agglomeration operator and the other operators should be used, which could prove a time-consuming process. The positive effect in this case, then, of EOC is to make it easy to incorporate problem specific knowledge into the optimiser. The results make it clear that given that knowledge, EOC will make the most of it.

Parent selection (§4.8) was also developed for the Shanxi problem. It was found that large populations were needed to preserve diversity, but that this slowed convergence. This, QMOO's parent selection method was developed, and used with an exponent of 2—quite strongly preferring better parents. No tests were performed on the value of the exponent.

QMOO was also compared to the Struggle GA. The SGA was run ten times to find the solution for problem 1, and ten times to find the solutions for problems 2 and 3. The SGA found the solution to 1 quite consistently after an average of 1245 evaluations. However, even after 24,000 evaluations it could not find the solutions 2 or 3 on any of the ten runs. The lowest construction cost solution for any transport cost that the Struggle GA found was 40.4 MRMB; it found this solution after an average of 12800 evaluations.

[3]But this varies considerably between runs and throughout a run.

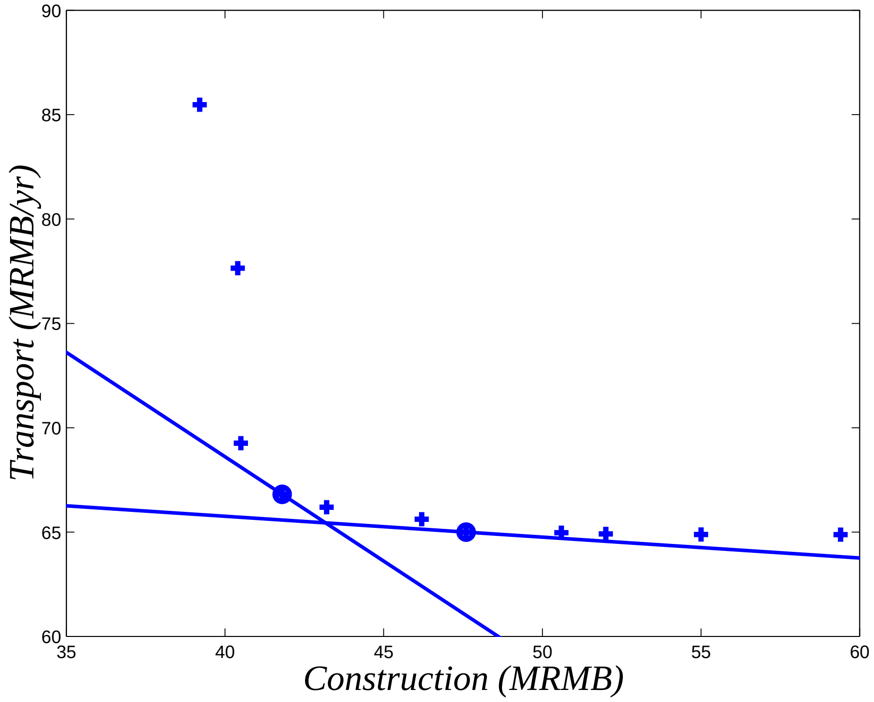


Figure 6.2: Trade-off between construction and transport costs for the reduced problem. Tangents and dots show trade-offs and solutions, respectively, for plant investment times of one and ten years—as described in the text.

The useful result of running the QMOO, of course, is the trade-off curve between construction and transport costs, as shown in Figure 6.2. The trade-off curve is convex, with a sharp 'corner'. This shape means that although a better solution for only transport cost or only construction cost than the solutions that are shown in Table 6.2 cannot be found, there are many solutions that perform considerably better for one cost with only a small compromise in the other cost. For example, the solution at a transport cost of 69.3 MRMB/yr and construction cost of 40.5 MRMB shows that transport costs can be lowered from the lowest construction cost solution by 16.2 MRMB/yr for an increase in construction cost of just 1.3 MRMB. Similarly, the series of solutions on the right side of the graph show greatly decreasing construction costs with little increase in transport cost.

The smallest combined total cost (the sum of the transport cost for one year and the construction cost) is given by the solution with transport cost of 66.8 MRMB/yr and construction cost of 41.8 MRMB. However, the more realistic assumption that the investments in the construction costs of a new coke-making plant are paid over a period of 10 years is made, the solution that minimises our total annual costs of transport and construction investment, is the solution found where the trade-off curve that has a tangent of 1:10. This solution has a transport cost of 65.0 MRMB/yr and construction cost of 47.6 MRMB.

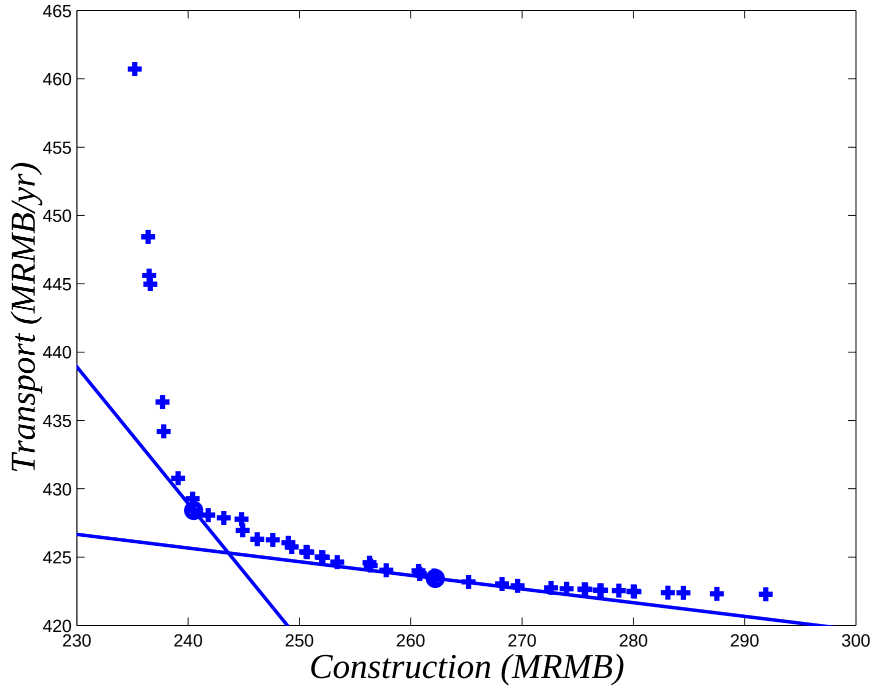


Figure 6.3: Trade-off between construction and transport costs for the whole problem. Tangents and dots show trade-offs and solutions, respectively, for plant investment times of one and ten years as described in the text.

6.1.5 Results

The entire problem is too large for it to be practical to find all the 'interesting' solutions that we described for the reduced problem by inspection. It is still possible, using the methods described above, to find the minimum transport cost (422.3 MRMB/yr), the construction cost for that transport solution (291.9 MRMB), and the absolute minimum construction cost (235.2 MRMB). These points can be used to check that the algorithm is converging on the larger problem. Finding the minimum transport cost for the minimum construction cost by inspection is impractical.

The large problem was run on a network of 6 machines over a weekend—resulting in slightly more than 100,000 function iterations. The results of the algorithm are 46 points approximating the Pareto-optimal frontier, shown in Figure 6.3.

The algorithm found all the points that can be found by inspection mentioned above, and further, the best transport cost at minimum construction cost found is 460.7 MRMB/yr. Therefore, the extremal solutions show a difference of 56.7 MRMB in construction cost and 38.4 MRMB/yr in transport cost.

Figure 6.4 shows the solution for the minimum construction cost. We see that there is a lot of activity in the north-east part of the Province. Several large plants are supplying a number of coke consumers in that area, particularly the large export node that consumes two thirds of the total coke production in the Province. However, for the western and southern parts of the Province, there is only one plant with coal and coke

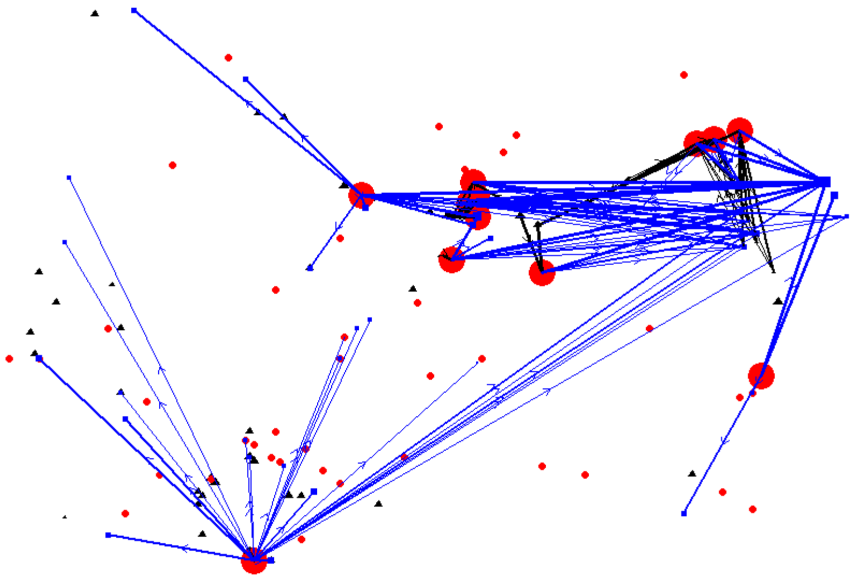


Figure 6.4: Plant Sitings for Minimum Construction Cost

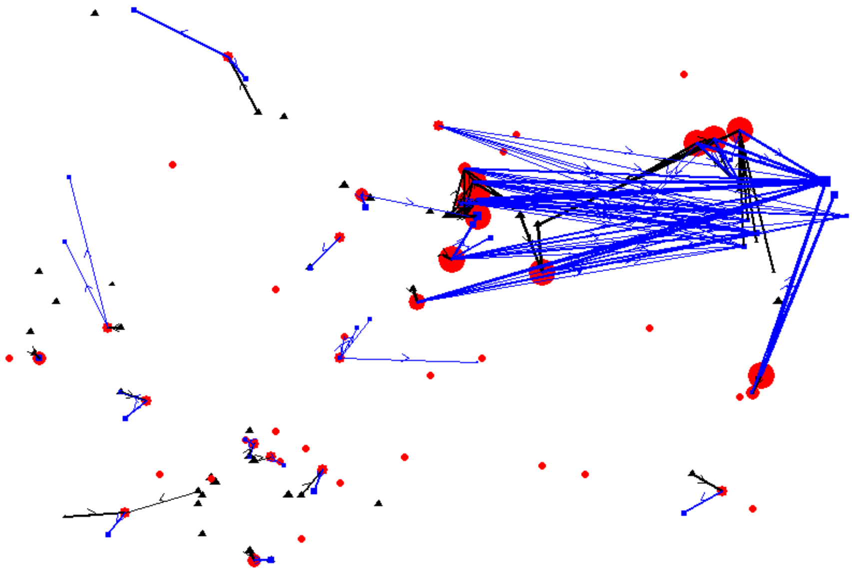


Figure 6.5: Plant Sitings for Minimum Transport Cost

transport vectors radiating from the plant to coal mines and coke consumers around the Province, resulting in the large transport cost for this solution.

In contrast, the map for the minimum transport cost solution, shown in Figure 6.5, illustrates that, while the situation in the north-east has not changed dramatically, there are now ten additional small coke-making plants supplying consumers locally in the western and southern parts of the Province.

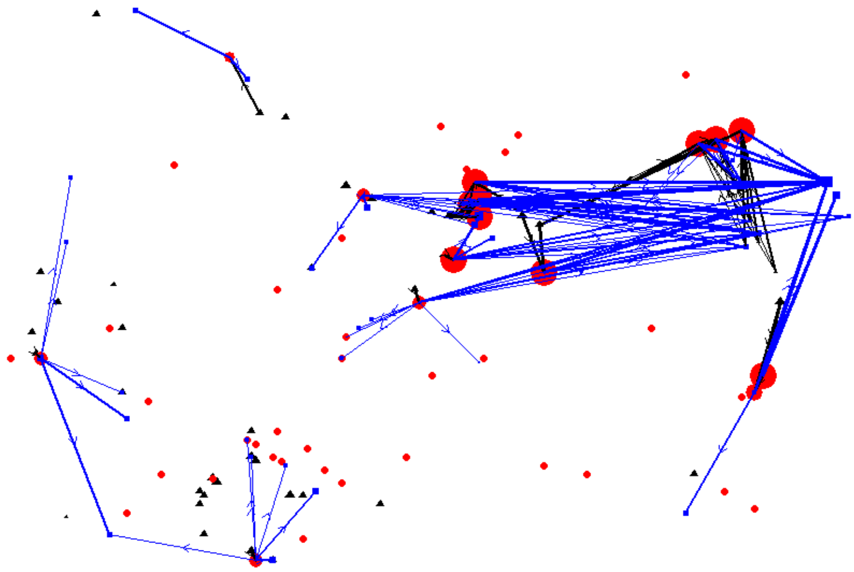


Figure 6.6: Plant Sitings for One Year Construction Payback

Like the reduced problem, the trade-off curve for the whole problem (Figure 6.3) is also sharply convex, indicating that decision-makers may be interested in solutions other than the extremal solutions. If we follow the same reasoning as in the reduced problem, the solution giving the minimum total cost for the 1:1 trade-off of construction and transport costs is the solution at a construction cost of 240.5 MRMB and a transport cost of 428.4 MRMB/yr. If we consider an investment period of 10 years for the plant construction cost, the lowest cost solution is at a construction cost of 262.2 MRMB and a transport cost of 423.7 MRMB/yr.

We show the maps for these two solutions in Figures 6.6 and 6.7. Clearly, there is a considerable difference in the plant siting for these two solutions.

6.2 Configuration of Hybrid Drivetrains

6.2.1 Introduction

The ‘Hybrid Problem’ is an optimisation problem treated in collaboration with Adam Molyneaux of LENI, funded by the AGS ‘Holistic Design’ project. This preliminary work is also presented in Leyland et al. [79] and Wallace et al. [133]. More comprehensive results are presented in Molyneaux [89].

The high computational requirements of the hybrid problem’s objective function (which is a simulation of driving a vehicle through a test cycle), was a driving force in the development of the parallel aspects of QMOO.

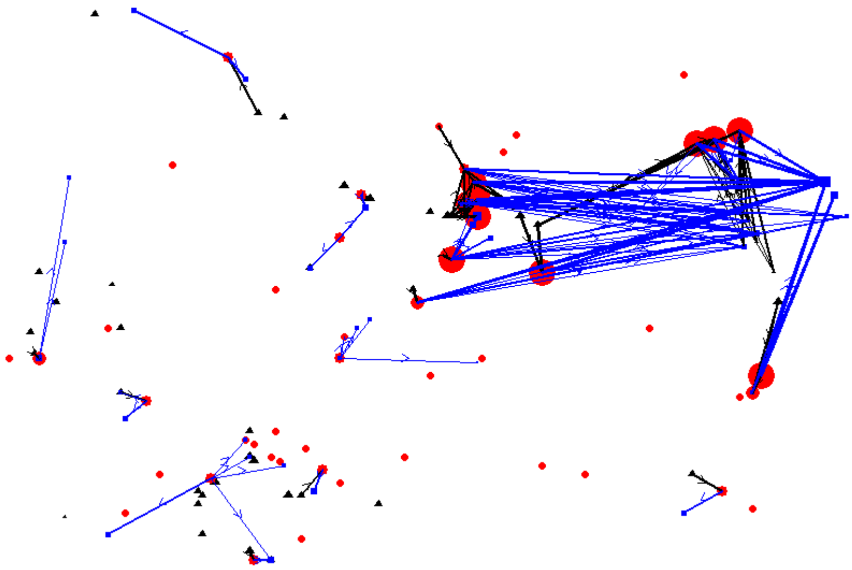


Figure 6.7: Plant Sitings for Ten Year Construction Payback

Background

In a conventional vehicle, the vehicle's final drive (differential and wheels) is driven by the internal combustion engine (ICE) through a gearbox. The gearbox allows the ICE to run near its optimal speed over a wide range of vehicle speeds. However, engine efficiency and emissions are strongly dependent on load, as well as speed.

Consequently, while an ICE's peak efficiency can be over 30%, overall vehicle efficiencies can be as low as 10%. A large portion of these losses are due to the engine not running at its most efficient point, but another important loss is engine running while the vehicle is stopped (more than half of the time in most test cycles). The drivetrain (gearbox and differential), on the other hand, are extremely efficient, and drivetrain losses are generally only on the order of 5%.

Hybrid vehicles attempt to avoid the losses incurred by the engine running when the car is stopped, and to reduce the losses due to not having the ICE running at its maximum efficiency. Furthermore, they can improve efficiency even more by recovering some of the energy dissipated in braking, which otherwise only serves to heat and wear out the braking system.

A hybrid vehicle, in the broadest sense, is a vehicle that contains several power sources, and uses those power sources in order to maximise overall efficiency. This work shows some preliminary analysis of thermal-electric hybrids—with an ICE and an electric motor. These may be configured in a series, parallel or mixed configuration. A hybrid vehicle superconfiguration, showing a series hybrid with bold arrows, is shown in Figure 6.8.

In a series hybrid, the ICE is connected to a generator, which either charges a battery or powers an electric

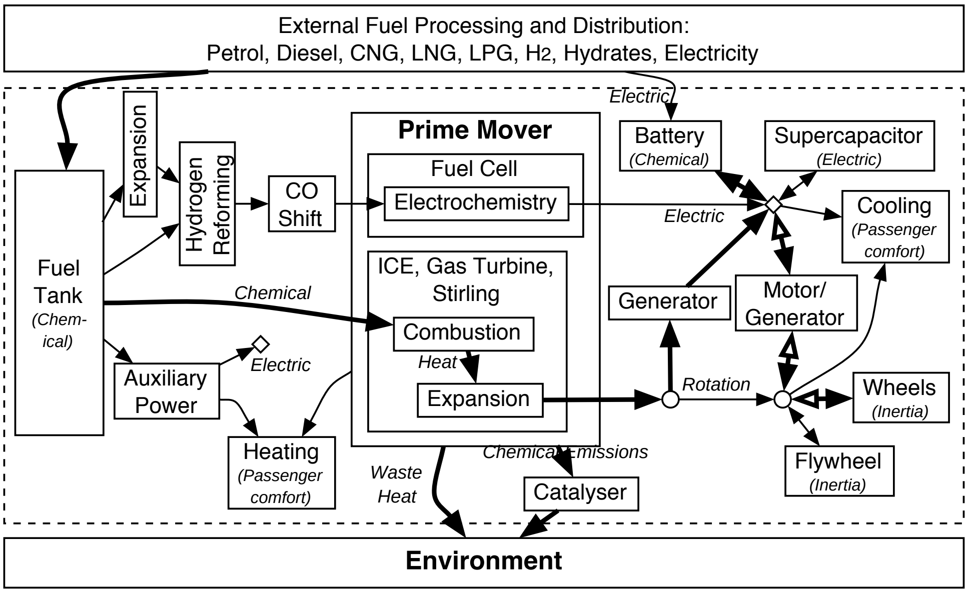


Figure 6.8: Hybrid vehicle superconfiguration. Focus here is on the ICE, electric motor and batteries.

motor. The motor, either powered by the battery or the generator, then powers the wheels. This means that the ICE can, if the control system wishes, always be run at its best operating point, while the electric motor, which hopefully has better characteristics over a wider load/speed range, copes with demand variations. Series hybrids are relatively simple compared to parallel hybrids (described below). However, in situations where the vehicle operating point would allow the ICE to run well, such as at constant speed on an motorway, there is nothing to gain from the hybrid configuration, and the losses of the complex mechanical-electrical-mechanical energy chain are much higher than for a conventional drivetrain.

Parallel hybrids solve this problem by allowing the ICE to either drive the generator or power the vehicle final drive directly. This involves the use of a torque coupling device (for example, the Toyota Prius has an epicyclic gearbox controlled by the generator [128]) between the ICE, electric motor, generator and final drive.

Generally, battery capacity in a hybrid is low (the Prius has 50 kg of batteries while the General Motors EV1 *electric* car has a 520 kg battery pack [54]) and the ICE needs to be run frequently to top up the battery charge, as well as whenever the vehicle has high power demands. In order to reduce charge cycle losses and improve battery life, batteries are usually kept within a narrow range of charge situated around half full. Consequently the batteries are usually larger and heavier than strictly necessary. If, however, the battery capacity is high, the ICE can be turned off completely for short periods, which may be interesting, for example, for crossing a town centre where only non-emitting vehicles are allowed.

The best choice of battery capacity, as with most vehicle design choices, is not evident—the Toyota Prius has twice as many battery modules as the Honda Insight [136].

6.2.2 Methodology and Optimisation

Vehicle Simulation Model

LENI developed simulation models of several advanced vehicle powertrains, some in co-operation with the University of Tokyo [65]. These models allow the engineer to specify the layout of a vehicle drivetrain, specify component sizes and control strategies, and then run the vehicle through any number of test cycles, measuring performance (how well the vehicle managed to follow the cycle), emissions, fuel economy, battery charge, catalytic converter performance and many other variables.

The simulation model is entirely modular, consisting of a series of self-contained components. This approach allows new technologies to be introduced transparently and allows models of different complexity or based on different assumptions to be easily exchanged. Thus, an engineer can construct a vehicle from a set of components, including any control logic that may be necessary.

Development of LENI's own vehicle models nonetheless proved a difficult task, and a number of other vehicle simulation models are freely available. Thus the modelling effort was turned toward ADVISOR [134], a Simulink-based car simulation model, that has a very similar functionality to LENI's original models, but enjoys the advantage that it is well established, open, and has many contributors. This allows us to concentrate on the laboratory's areas of expertise (compressed natural gas engines, catalytic converters and emissions [55], thermodynamic systems modelling), and to make the most of the experience of others in their domains.

The ADVISOR simulation toolbox allows a vehicle to be built up from a series of components, each of which publishes a set of controlling parameters that may be modified from an initial default value. Once specified, a given configuration (a car) can be *driven* through a drive cycle using a specific strategy and any of the calculated values can be monitored or integrated.

Optimisation

The ADVISOR simulink model is computationally expensive—requiring several seconds to run on a modern PC—and to optimise all but the most trivial of problems requires tens of thousands of test runs. For this reason, the parallel version of QMOO was developed.

Although QMOO has been applied to optimisations with many more independent variables than that presented here, the vehicle configuration problem poses particular difficulties. Notably, with the complete change of vehicle configuration between conventional and hybrid drivetrains, some components are added and removed from the configuration, and the optimal dimensions of others differ greatly.

For this study, powertrain configuration (conventional, series hybrid and parallel hybrid), ICE size, final drive ratio, electric motor size, generator size, battery size and battery minimum and maximum charge, were varied. The objectives are fuel economy through two drive cycles. Here, the ability to test the vehicle through several cycles becomes very interesting, as we expect to see a clear advantage for hybrid vehicles in city driving, while on the highway, we do not expect them to show any large advantage over a conventional car.

6.2.3 Results

The objectives chosen were fuel economy over each of the two drive cycles shown in Figure 6.9 (left)—the ECE-EUDC representing typical urban use and the US06-HWY drive cycle representing high speed cruising operation. The choice of configuration was limited to a parallel hybrid, a series hybrid or a conventional small car. The seven other configuration parameters and their limits are shown in Table 6.4. The vehicles defined by these parameters (and many defaults for the vehicle type) were driven through each drive cycle and the fuel economy calculated. The rare vehicles that failed to follow the drive cycle by more than 3.5 km/h were removed from consideration. In order to avoid unfairly preferring hybrid configurations with large battery capacities, the state of charge of the vehicle batteries at the end of the cycle was required to be within 5% of its initial state. To achieve this the cycle had to be to run iteratively, adjusting control parameters until the required final charge state could be achieved. Hybrid vehicles which failed to satisfy this requirement were also removed from consideration.

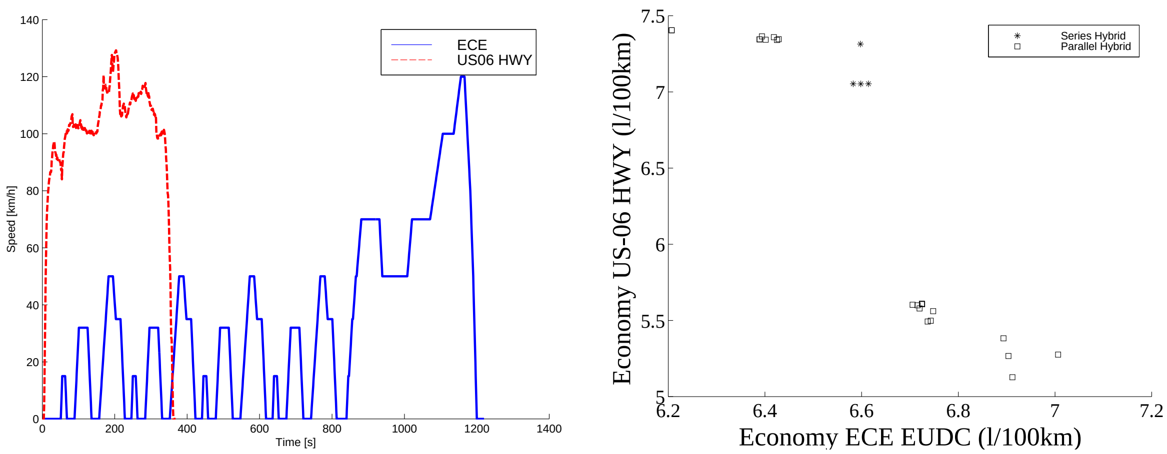


Figure 6.9: Left: The ECE-EUDC and OS06-HWY drive cycles. Right: Two-cycle fuel economy results

Independent Variable	Limits
ICE Size (Nominal 63 kW)	31 to 126 kW
Final Drive Ratio	0.65 to 1.6
State of Charge:	
Upper Limit	0.3 to 0.8
Lower Limit	0.3 to 0.8
Electric Motor Size (Nominal 75 kW)	30 to 112 kW
No. of Battery Modules (12 V, 11 kg modules)	0 to 60
Generator Size (Nominal 32 kW)	13 to 48 kW

Table 6.4: Independent Variables and Limits

The hybrid problem was run across a network of up to 6 computers (generally only about three slaves and the master were working at any one time). Runs of several days (5,000-10,000 objective function evaluations) were required in order to show reasonable progress toward the Pareto-optimal frontier. Figure 6.9 (right) shows the Pareto-frontier with the optimal drivetrain types marked with different symbols.

The results are not quite as expected—notably there are no conventional drivetrains at all in the Pareto-optimal set, and these were expected to perform well in the US-06 HWY cycle. The results show that there is a clear difference between performance on an urban cycle and a highway cycle, and that in general, parallel hybrids are favoured over series hybrids.

The best performing configuration on the ECE-EUDC cycle, at the top left of Figure 6.9, is a parallel hybrid with a relatively small ICE, a middle-sized battery pack, and a high final drive ratio (1.6). The best performers on the US-06 HWY (bottom right), also have small ICEs, but the maximum battery capacity and a lower final drive ratio (1.1). All configurations have the minimum electric motor size, indicating that the limit presented in Table 6.4 could be set lower.

Why are there no conventional drivetrains in the Pareto-optimal set? We speculated earlier that conventional drivetrains should perform well on highway driving, as a hybrid drivetrain, at first view, offers no advantages in this mode. However, while the US-06 HWY cycle is for the most part at a constant high speed, it also contains a hard acceleration at the start of the cycle. Thus, in a conventional vehicle, the ICE *must* be sufficiently large to provide the acceleration and so is not dimensioned for economy. The hybrid configurations, on the other hand, can choose an ICE size appropriate for fuel economy and use the electric motor and the energy stored in the batteries to provide acceleration when necessary.

Following this logic, we can see that the large battery capacities in the well-performing cars on the US-06 HWY are used to provide acceleration, rather than storage—thus it is the battery's *power* density, rather than its *energy* density that is important. The best vehicle for the ECE-EUDC, on the other hand, with, in all cases, the smallest components selected, is effectively a light-weight city car.

Further results and analysis of this problem will be presented in Molyneaux [89].

6.3 Load Scenario Characterisation

6.3.1 Introduction

Simulating or optimising an energy system or industrial process often means simulating the behaviour of the system over time. In some cases, the simulation will need to be fully dynamic—transient behaviour will have a significant effect on overall system comportment. In other cases though, the system can be sufficiently well simulated with a 'discrete time' approach. Here, time is broken up into a number of segments, each with different demands on the system, and each of these demand scenarios is simulated or optimised as a steady state problem. Thus, while transients are not treated, the performance of the system over the full range of operating conditions can be investigated.

Examples of such systems are:

- a combined heat and power generation system with varying demands for power and heat. The demands may vary seasonally, daily or even hourly;
- a chemical process plant that produces several different products;
- variations of production levels as a result of changes in market conditions;
- variations in ambient conditions changing process conditions.

Usually a new plant or process will be simulated using historical data for operating conditions, a simulation for each data point. If there is a large amount of historical data, say, data collected daily for a year, or number of years, the time to execute a full simulation can be extremely long. If the task is to optimise the operating strategy over time, the number of variables needed to specify the strategy becomes enormous.

Often, load cases that are taken at a high time resolution can be grouped into a number of smaller categories. Rather than characterising a system by daily loads, one may find that the variation is mostly seasonal. Alternatively, if one has hourly demand data for each of the processes, one may find that though there is variation throughout the day, the process looks much the same from one day to the next. Thus, only a single day needs to be simulated, rather than the whole year.

Finding typical load cases when there is only one load or demand to meet—one process to be controlled—is quite easy. Unfortunately, in the general case, with many processes to control, finding typical cases is not trivial. As an example, daily production levels from a chemical plant that includes 5 integrated chemical processes are shown in Figure 6.10, which shows no obvious pattern. The goal in this case is to define the optimal co-generation system for providing the power and heat needs for these processes.

Without managing to group the data, the modeller is faced with optimising every day of the year in order to correctly model system behaviour. If a few typical load scenarios could be identified, this would greatly reduce the computation time to run the model.

6.3.2 Problem Description

The modeller has a set of T data points, $\boldsymbol{p}^t$, detailing the demands of M processes in T time segments, and would like to characterise these by N clusters, $\boldsymbol{c}^n$, (N being much less than T) so that some measures of the error caused by approximating each member of the T data points by one of the N clusters is minimised.

Thus, the optimisation algorithm is given the list of data points, and a description of the objective functions to be used, and must find a good set of N clusters. The goal of the exercise is to minimise the errors due to approximating each time segment by the point defining its closest cluster, so the objective function (or functions) are some measure of this error.

In this work, the error in representing a time segment by the nearest cluster was measured as a weighted Euclidean distance from a data point and the cluster which best approximated it. Thus, the error in approxi-

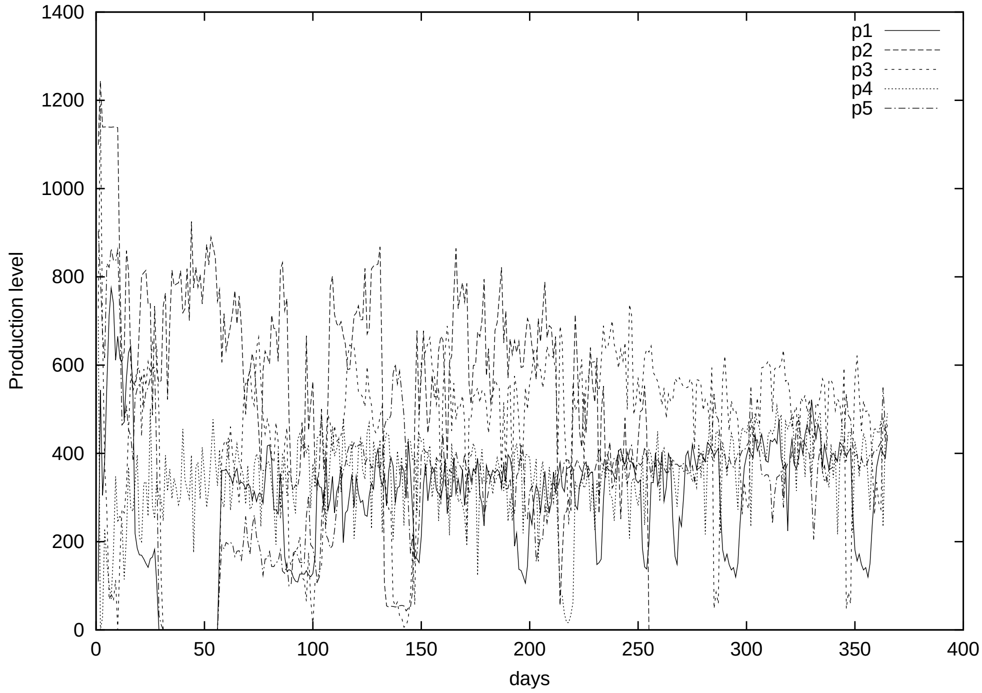


Figure 6.10: Variation of the process levels with time. It is difficult to see any pattern in this data.

mating a point t with a cluster n is:

$$E_{tn} = \sum_{m=1}^{M} \left(w_m(p_m^t - c_m^n) \right)^2 \tag{6.1}$$

where $\boldsymbol{w}$ is a vector of weights for each process.

The error is calculated for every time segment-cluster pair, and the cluster with the lowest error for a time segment is chosen to represent it. A number of measures of the error can be calculated, and possibilities are:

- the total error: $\sum_{t=1}^{T} \frac{\min}{n} E_{tn}$;
- the maximum error for any time segment: $\frac{\max}{t} \frac{\min}{n} E_{tn}$;
- number of time segments that are poorly represented—the number of segments for which $\frac{\min}{n} E_{tn} > eps$, where eps is a limit on the error specified by the user.

Finding any of these errors is a non-linear and discontinuous optimisation problem. Though the formulation is fairly easy, a number of difficulties can be seen:

1. the system is not unimodal—the cluster centres can be permuted without changing the objective function value, and so there exist $N!$ identical solutions;
2. the system is discontinuous because the choice of cluster for each time period is a selection from a list of values made independently: at each time t, we choose from the N clusters the one that best

represents the $\boldsymbol{p}^t$ demand levels. A time period can jump from one cluster to another with slight variations in the values of $\boldsymbol{c^n}$, leading to jumps in the values of the objective function and any computed derivatives required by deterministic optimisation methods.

These observations were confirmed by tests using a non-linear optimisation package employing a Newton-like method to minimise the sum of the errors. This mathematical programming formulation was inefficient and unable to find a solution. The convergence of the method was highly dependent on the initial values, and so an initialisation method was implemented, that searches the set of existing points for starting points that minimised the objective function:

1. Set $j = 1$
2. Choose $\boldsymbol{c}^n = \boldsymbol{p}^t$, choosing t so that $\sum_{t=1}^{T} \left\{ \min_{j=1,\ldots,n_t} \sum_{i=1}^{n_p} \left(\frac{q_{i,j} - q_{i,t}}{p_i} \right)^2 \right\}$
3. Eliminate the points that are represented satisfactorily i.e. $\sum_{i=1}^{n_p} \left(\frac{x_{i,j} - q_{i,t}}{p_i} \right)^2 \leq eps$
4. Set $j = j + 1$
5. Go to step 2

However, the initialisation procedure is time consuming, and the optimisation still failed to find satisfactory solutions. It was also observed that there were local optima which lead to premature convergence of the algorithm. In order to try to smooth the objective function, a second objective function, the number of poorly represented points, was added to the first, but without success.

A number of other approaches to the problem were tried, including fuzzy c-means clustering (§4.6.3), without great success. Using an MOEA was thus considered. This approach offers several advantages:

1. there is no need for a complex problem formulation. In MATLAB, the value of the objective function is computed using simple vector manipulations;
2. there is no need for an initialisation procedure;
3. there is the possibility of optimising several of the objectives defined above;
4. the algorithm can cope with the permutability of the cluster centres;
5. the algorithm should be able to find the global optimum, even if there is more than one local optimum.

6.3.3 Solution

The problem to be solved by QMOO is thus to find $M \times N$ floating point values, while minimising a number of measures of the representation error. The decision variable limits are taken as the minimum and maximum values appearing for each of the M processes. In order to solve the 'permutation' problem—that two solutions with exactly the same clusters in a different order look different—the clusters are sorted in the

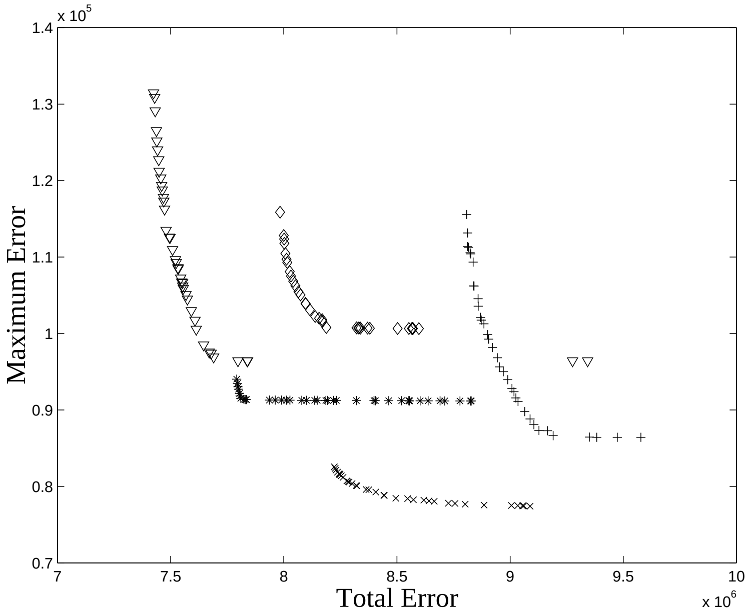


Figure 6.11: The final population of the load scenario problem in objective space. Finding 20 groups for five processes. Several local optima are found though the best group dominates most of the others.

vector of decision variables. After a new individual is assigned decision variable values, the clusters (groups of M variables), are sorted by their value for the first process[4].

Interestingly, the problem proves to be multi-modal—there are several quite different choices of cluster locations that perform reasonably well, and so the clustering problem is best solved with grouping turned *on* in QMOO.

QMOO was used to find 20 clusters to characterise the data shown in Figure 6.10, using the total error and maximum error as objectives (using the number of poorly represented points gave results that were highly dependent on the choice of representation tolerance). The final population, of several groups, is shown in objective space in Figure 6.11, while Figure 6.12 shows the resulting approximation for the minimum total error, and Figure 6.13 that corresponding to the smallest maximum error. Figure 6.11 shows that the two objectives are in clear conflict—the maximum error almost doubles between its own optimum and that of the total error. The total error is not quite as sensitive to the maximum error. In fact, the three clusters making up the global NDS are quite different, but nonetheless, the final population suggests that it may be possible, given enough time, to converge to a single global NDS. Given that the problem is clearly multi-modal (there are several well-formed NDSs) there is no guarantee that the best local optimum has been found.

[4]This is a nice demonstration of the flexibility of an EA. Such an approach was also taken to 'correct' some variables in the district heating problem. It has no incidence on the performance of the EA whatsoever—other than the positive influence of sorting the clusters.

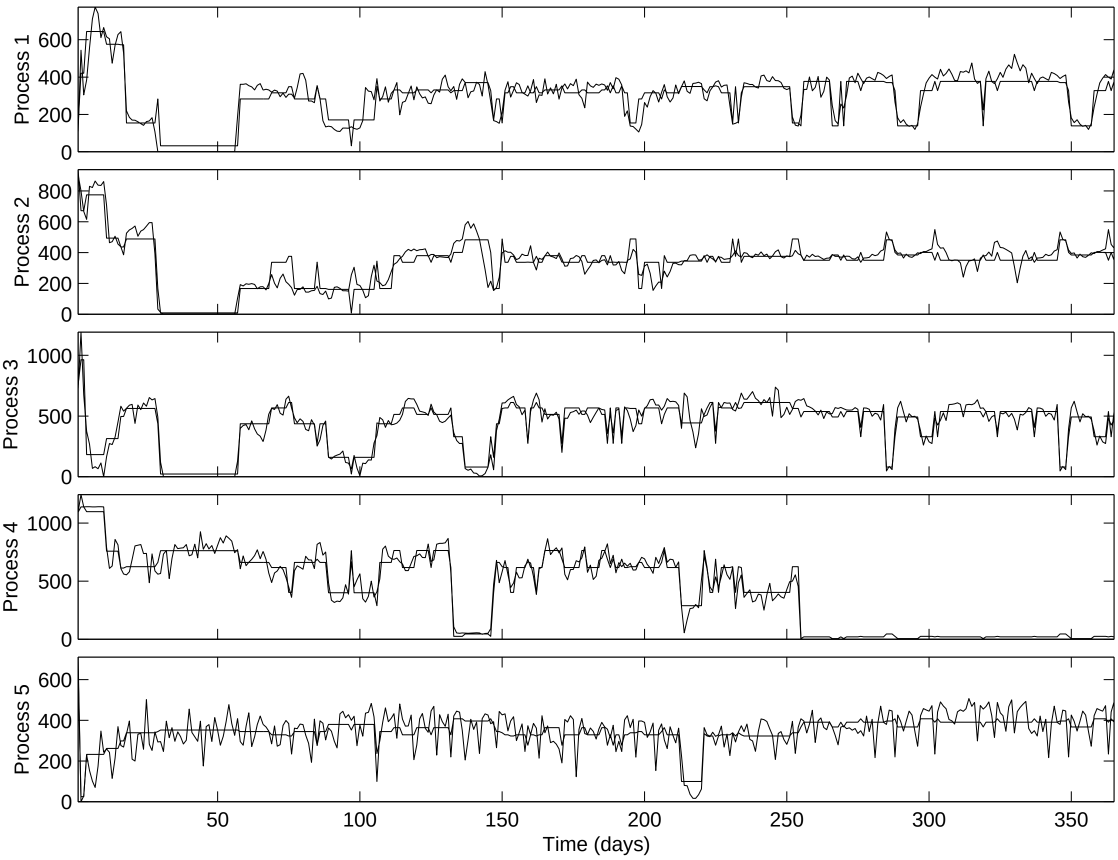


Figure 6.12: Approximations to loads for the minimal total error. (Top left point in Figure 6.11).

6.4 Grand Composite Curve Analysis

6.4.1 Introduction

Most industrial processes have a variety of heat demands and sources. These demands and sources can be characterised by a Grand composite curve which shows the minimum energy requirements of the process as a function of temperature [82, 44]. In order to minimise a process's waste energy, heat can be re-used between sub-processes. For example, the waste heat from a burner can be used in a process requiring lower grade heat, such as a pre-heating or drying stage. Equally, sources of cold can be used for cooling in other parts of the plant.

The transfer of heat around the plant necessarily requires some equipment to bridge between temperature levels. Such equipment is costly and ideally will be placed where it can recover energy most efficiently. Here, only the problem of choosing the temperature levels (or cuts) at which equipment will operate is covered. Kalitventzeff and Marechal [66] treats the subsequent problem—that of choosing the most appropriate technology for the heat and energy requirements from the analysis of the Grand composite curve.

Thus, given a Grand composite curve for a process, identifying the temperature levels at which heat should

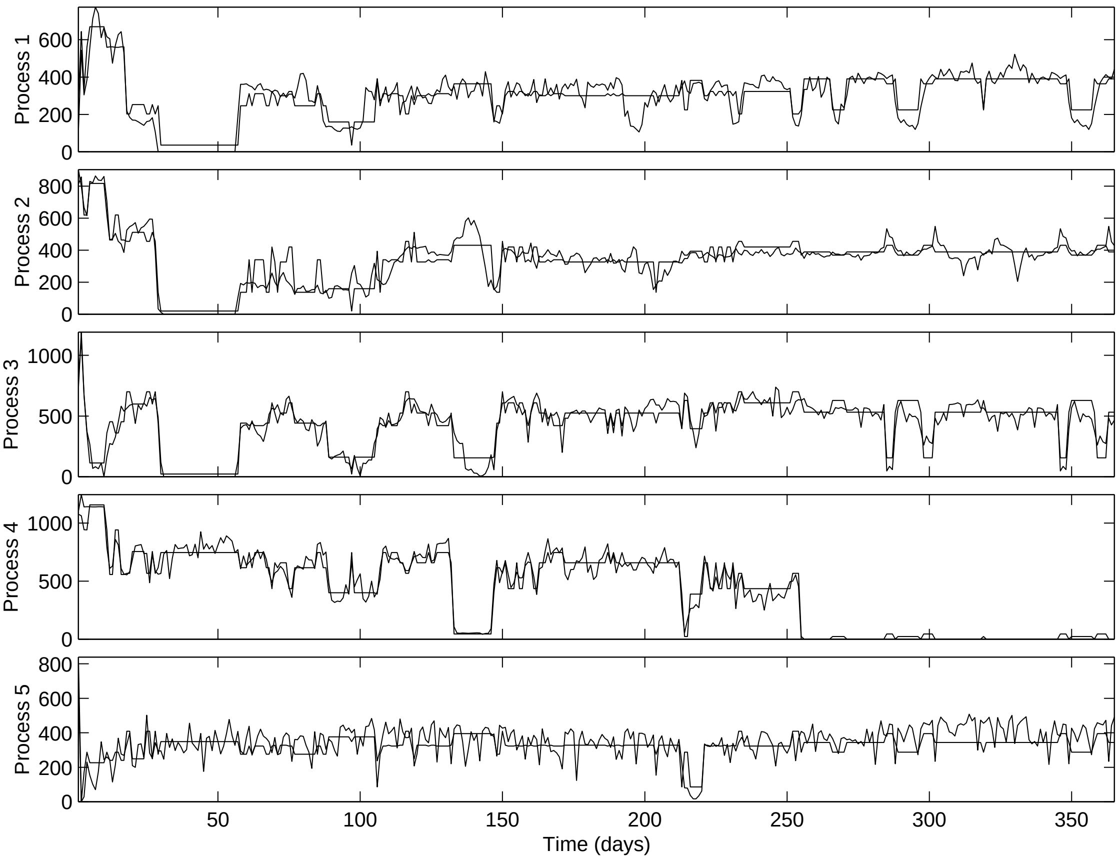


Figure 6.13: Approximations to loads for the smallest maximum error. (Bottom right point in Figure 6.11).

be extracted is an optimisation problem that involves both maximising the heat recovered, and minimising the equipment required to recover that heat.

In cases where there are many cuts compared to the number of segments on the composite curve, and the composite curve is smooth, the problem can be solved by inspection. However when there are many more segments on the Grand composite curve than cuts, as shown in Figure 6.14, the problem becomes considerably more difficult. The composite curve is not continuous, and this makes the problem intractable for many conventional optimisation methods. The problem is described in more detail in Marechal and Kalitventzeff [84; 85], in which a discretisation method is used to formulate the problem as a mixed integer linear programming problem. The solution is obtained by selecting from a list of discretized temperatures those that result in the best objective function. The full range of temperatures is cut into k discrete temperature levels, and the problem is formulated as follows:

$$\underset{R_k,\, y_k,\, q_k}{\text{minimise}} \qquad \sum_{k=1}^{n_k} q_k \left\{ 1 - \frac{T_0}{T_k} \right\} \tag{6.2}$$

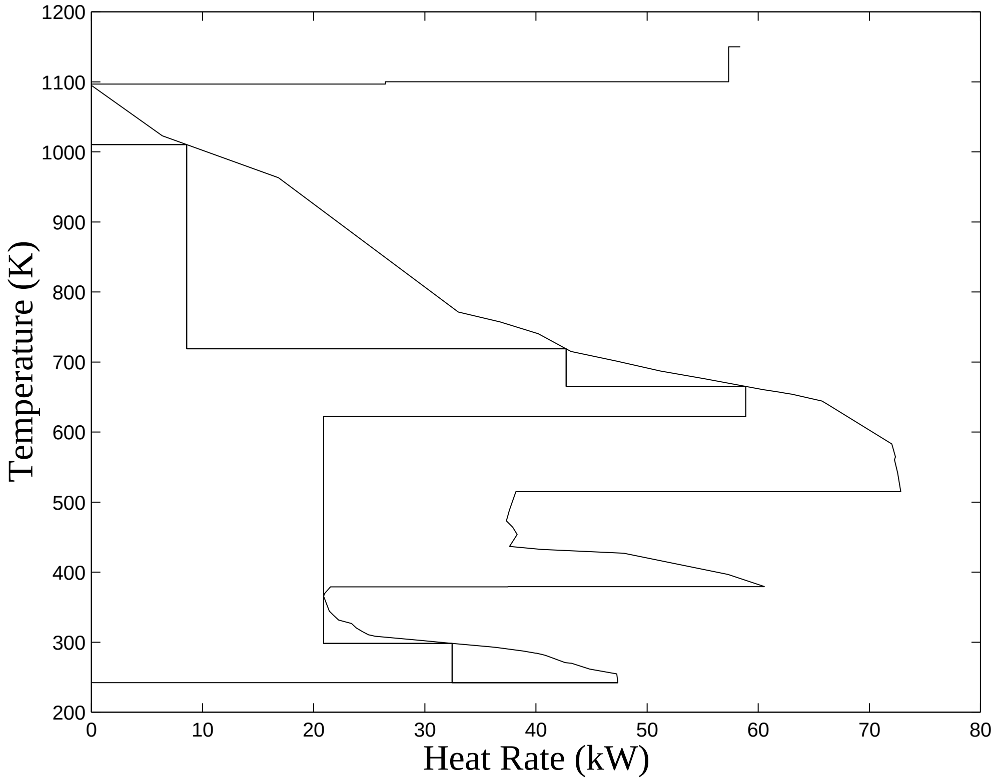


Figure 6.14: A Grand composite curve, and six non-optimal cuts. The top line is the composite curve, the straight line beneath it represents the heat that can be recovered using six (randomly chosen) temperature levels. Optimising means choosing the best possible temperature levels for a limited number of cuts.

Subject to:

$$q_k + \sum_{i=1}^{n} Q_{ik} + R_{k+1} - R_k = 0 \quad \forall k = 1, ..., n_k \tag{6.3}$$

$$q_k^{min} y_k \leq q_k \leq q_k^{max} y_k \quad \forall k = 1, ..., n_k \tag{6.4}$$

$$\sum_{k=1}^{n_k} y_k \leq N_{max} \tag{6.5}$$

$$y_k \in \{0, 1\} \quad \forall k = 1, ..., n_k \tag{6.6}$$

$$R_k \geq 0 \quad \forall k = 1, ..., n_k + 1 \tag{6.7}$$

$$R_1 = 0, \quad R_{n_k+1} = 0 \tag{6.8}$$

with
T_k	the temperature at level k
q_k	the heat rate of the utility system at temperature level k ($q_k \geq 0$ for a hot stream)
Q_ik	the heat rate brought to the system by process stream i in temperature interval k
R_k	The 'error' in the heat rate—the difference between q_k and the composite curve in temperature interval
y_k	the integer variable associated with the use of the heat requirement k
q_k^{min}	the minimum heat rate accepted for a hot utility at temperature level k
q_k^{max}	the maximum heat rate accepted for a hot utility at temperature level k
N_{max}	the maximum number of temperature levels to be considered

Thus, no more than N_{max} temperature levels must be chosen (the y_k). At each of these levels, the heat load q_k must be found, and must not be greater than the available heat (q_k^{max}), or much less than it (q_k^{min}).

Note that the quantity to be minimised is the heat rate times the Carnot factor: $q_k \left\{1 - \frac{T_0}{T_k}\right\}$, the Carnot factor being a non-linear function of temperature.

Some difficulties are noted with this approach:

1. A temperature level is used if: $q_k^{min} y_k \leq q_k \leq q_k^{max} y_k$ thus, q^{min} and q^{max} are used to determine if the level is to be used. Careful definition of q^{min} and q^{max} is needed to ensure the numerical consistency of the equation: it must be possible to discriminate numerically the value of the equation between $y_k = 1$ and $y_k = 0$.

2. The discretisation of the temperature range into discrete temperature levels is arbitrary. When the number of possible levels increases (the discretisation is refined), the number of variables (integer and continuous) will be increased and the size of the problem increases. Further, the difference between one level and its neighbours will be lower, so the choice between the different y_k becomes difficult, inducing problems with convergence and numerical stability.

The discretisation of the temperature range into a set of temperature levels is dictated by the Grand composite curve not being continuous, which precludes the use of classical non linear optimisation algorithms. Some authors [40] suggest using smooth approximations to handle this complication but the methods do not give good results.

An EA approach appears attractive in this situation. The advantages are the following:

1. only two variables are needed per temperature level to be found, rather than two per *discretised temperature*;

2. formulation of the problem is easy and rapid;

3. the resulting method should cope with discontinuities, and provide good numerical precision.

Furthermore, if the number of temperature levels to be found is added as a second objective, an MOEA can be used to show the trade-off between adding a temperature level and heat revalorisation. From an application point of view this gives information on the benefits in terms of exergy losses of adding extra equipment, and what changes this will have on the other temperature levels (adding a temperature level will probably result in most of the others moving).

6.4.2 Solution

At a first glance, the problem seems to be one of selecting both temperature and heat rates at each cut, as Marechal did. This is a difficult problem for an EA, as the heat rate for a cut must not exceed the available heat. Not only must the optimiser choose both the temperature and heat rate for each cut, but many solutions will be infeasible. Many other solutions will clearly be sub-optimal, as they will not use all the available heat.

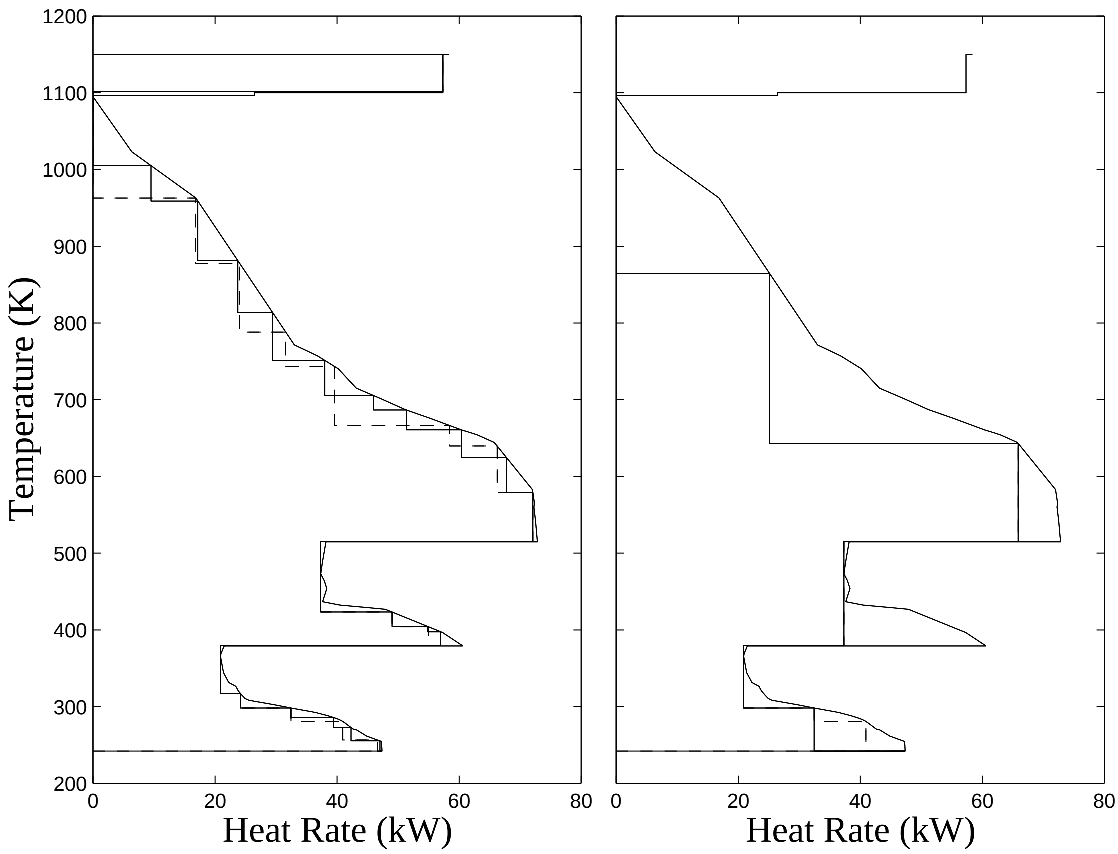


Figure 6.15: Optimal cuts for the Grand composite. Left : 23 and 18 cuts, reducing the number of cuts changes nearly all the cut levels. Right : 7 and 8 cuts. A more practical number of cuts. The new cut is at the bottom of the temperature scale, where the Carnot factor is high.

A better solution is to choose only the temperature levels, and then interpolate the appropriate heat rates off the composite curve. This ensures that all the cuts are feasible, and that they use all the heat available at that temperature level. However, care must be taken when interpolating off the composite, as it is not monotonic, and so it is possible that between two high heat rate cuts, the composite curve has a low heat rate region. Thus, after interpolating the cuts onto the demand curve, any composite curve points inside the cuts must be found, and the cut levels adjusted.

Choosing the number of cuts used requires binary variables to choose which cuts are on or off. Initially, it was thought that only the *number* of cuts to be used needed to be chosen, and that the cuts contributing the least to the coverage could be removed one by one. However, there are easily constructed problems that show that the optimal subset of cuts is not necessarily that generated by this process.

Thus N floating-point decision variables representing temperature levels for a problem considering up to N cuts are required, along with N binary variables choosing whether a cut is used or not. All the floating-point variables choose a temperature level from the entire temperature range covered by the composite curve. If the problem is not posed carefully, this means that individual in a three-cut problem with the cuts $(273.15, 293.15, 323.15)$ and another $(293.15, 323.15, 273.15)$ will be considered different, when in fact they represent exactly the same cuts, just in a different order. If the left this way, the algorithm will waste time optimising different versions of the same cuts. To remedy this problem, a sorting step is added after the creating of new individuals, so that the cuts are ordered in each individual. Sorting the cuts has a considerable influence on the convergence performance for the problem.

The method was tested, and gave useful results for two test cases. A selection of the cuts found are shown in Figure 6.15, where the effect of adding or removing a few cuts can be seen. Power recovery possible vs. number of cuts is shown in Figure 6.16. Above about ten cuts, very little more power can be recovered by adding more cuts.

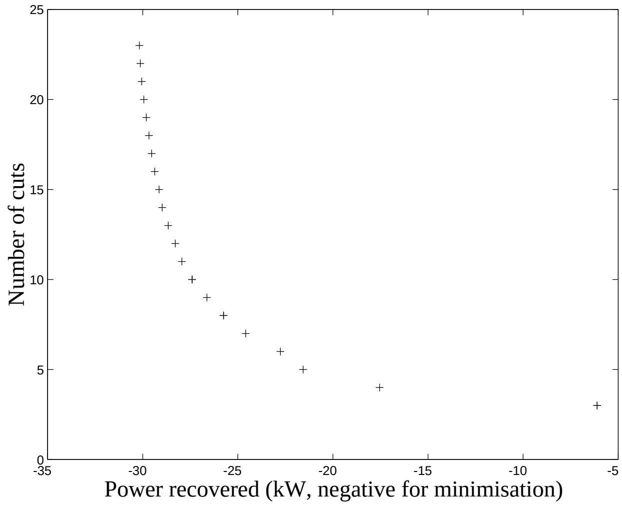


Figure 6.16: The NDS for the composite problem: Power recovered vs. number of cuts.

6.5 Heat Recovery in Batch Processes

6.5.1 Introduction

The ‘Heat Integration’ problem is the subject of Pierre Krummenacher’s thesis [73]. The thesis treats the development of models describing batch processes, and of recovering heat from one part of the process to use in another, or from one time interval to another. Heat recovery in batch processes is more difficult than in continuous processes. In a continuous process hot streams are always present, and so the heat can be used by any other part of the process that requires the heat. In a batch process, streams are only running for a part of the batch cycle time, and so, while there may be an appropriate hot stream to heat another part of the process, the hot stream may not be running at the same time as the heating is required. Krummenacher studies overcoming this problem by using heat storage units—heat can thus be stored until needed. As heat storage units are expensive, they must be carefully chosen to minimise cost and maximise the heat recovery.

The thesis does concern itself with the optimisation of these models (for a single objective), though the optimisation method used is certainly not of prime importance in the thesis. Nonetheless, the optimisation problems proved sufficiently difficult that special methods had to be developed, at some expense, in order to solve the problems. The methods involved optimising in two stages—where the optimisation was started with one objective, and later changed to finish with another—and on two levels—where structural variables were chosen by the ‘top level’ optimiser, and then the ‘process variables’ were optimised by the ‘bottom level’ optimiser. These methods were necessary because without them the optimiser could neither manage to find acceptable solutions, nor perform a ‘full system’ optimisation—optimising both structural and process variables.

The Struggle GA was used and required a distance function in order to find a new individual’s nearest neighbour. This distance can be a Euclidean distance, but in this case it was found that a more complex distance function was required. At first a distance function that only distinguished between stream temperature levels was tried, but with this, structural optimisation was not possible. Then, a distance function discriminating strongly between structures was tried, and this time, the process variables could not be optimised. Thus, the two-level optimisation mentioned above was used, the structural level using one distance function, and the process level, the other.

For further details of the batch processing systems optimisation problem, see Krummenacher’s thesis [73].

In this work, the process models were taken without any changes, and optimised using QMOO. QMOO was not tuned at all for the problem, and no special optimisation methods—the problems were optimised using QMOO with default values for the control parameters. Nor was the distance function developed by Krummenacher used—QMOO’s fuzzy c-means clustering used a Euclidean distance between individuals, and seemed to work well.

6.5.2 Solution

The problem has already been solved for a single objective—total batch costs—by Krummenacher [73]. The Struggle GA was used for the optimisation, but obtaining good convergence proved difficult—the objective function is very flat around the optimum. Furthermore, because the only one objective was being optimised, solutions with very poor heat recovery were found—apparently low-cost solutions with good heat recovery are very hard to find. To remedy this, a two-stage optimisation process was implemented. For the first half (or so) of the optimisation, heat recovery was optimised to ensure that a good value for heat recovery would be found, in the second stage, the optimisation continued, now with total batch costs as an objective. This approach took some time to develop and was somewhat complicated, but provided satisfactory results.

Krummenacher's EP1 test problem, considered here, is an imaginary test case with 22 real 'process' variables, and seven binary 'structural' variables. Krummenacher never solved the full problem, keeping all the structural variables fixed at 1. The process optimisation problem was solved for the single fixed structure using the two-stage approach. A run of 10,000 generations of 75 individuals (thus 750,000 objective function evaluations) found a minimal total batch cost of 28.08 CHF/batch[5], with a heat recovery of 292.16 kWh/batch. The heat recovery is within 2% of the maximum possible for the problem—296.25 kWh/batch.

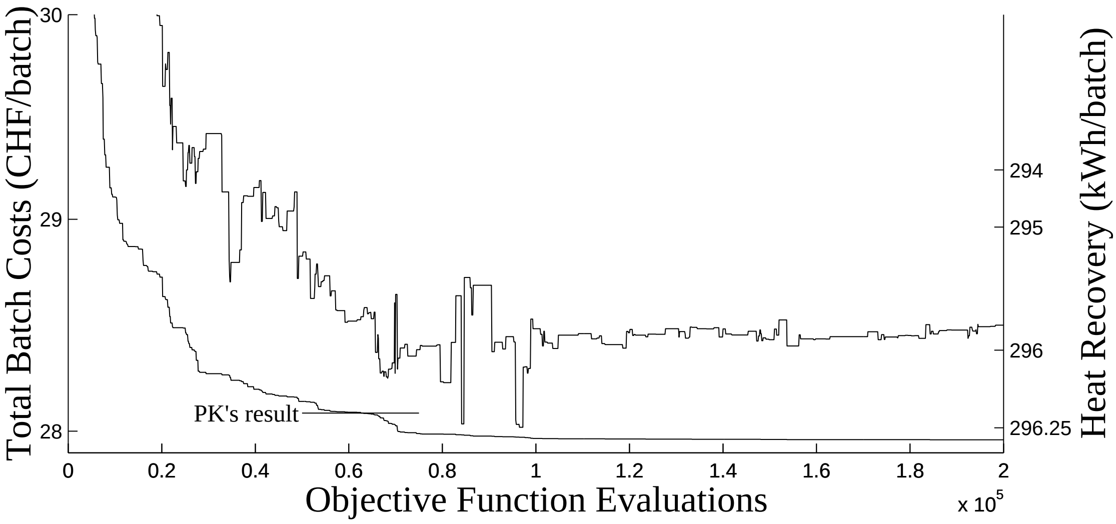


Figure 6.17: Convergence for the first objective on the heat recovery problem EP1. The lower line (left scale) shows convergence of the total batch costs; the lowest cost Krummenacher found after 750,000 is shown, and is passed in less than 75,000 evaluations. The higher line (right scale, note that it is inverted) shows the value of the second objective at which the best first objective occurs.

EP1 was optimised with QMOO in order to see if QMOO offered better performance, and what insight a multi-objective optimisation would give about the relationship between total batch costs and heat recovery.

[5] The figure in Krummenacher's thesis is 27.40 CHF/batch, but this is because the cost in the thesis is adjusted to take account of the re-use of a heat-exchanger that is not considered in the model. Thus, the model returned 28.08 CHF/batch, and the value was adjusted [72].

With very little preparation of the problem (enough to specify the bounds on the variables), QMOO was applied to the problem, with *no* tuning apart from choosing to solve with three groups[6], just in case the problem had several local optima. Less that 45,000 objective function evaluations into the *first ever* run on the problem, QMOO had found cheaper solution, but with a lower heat recovery (290.4 kWh/batch). Exactly when a member of the population dominated Krummenacher's solution was not recorded, but at the end of the run of 75,000 function evaluations ten population members were better in both objectives. Thus QMOO found an entire non-dominated set *ten times faster*[7] than the earlier method had found a single point.

Given this success, QMOO was set on the 'entire' problem—including the seven structural binary variables—that Krummenacher had not attempted to solve. The number of groups was set at seven, as it was imagined that the structural variables would lead to more local optima, and tail preservation was turned on (with very small tail regions that generally only contained one individual), as it was noted that there were problems finding cheap solutions with low heat recovery.

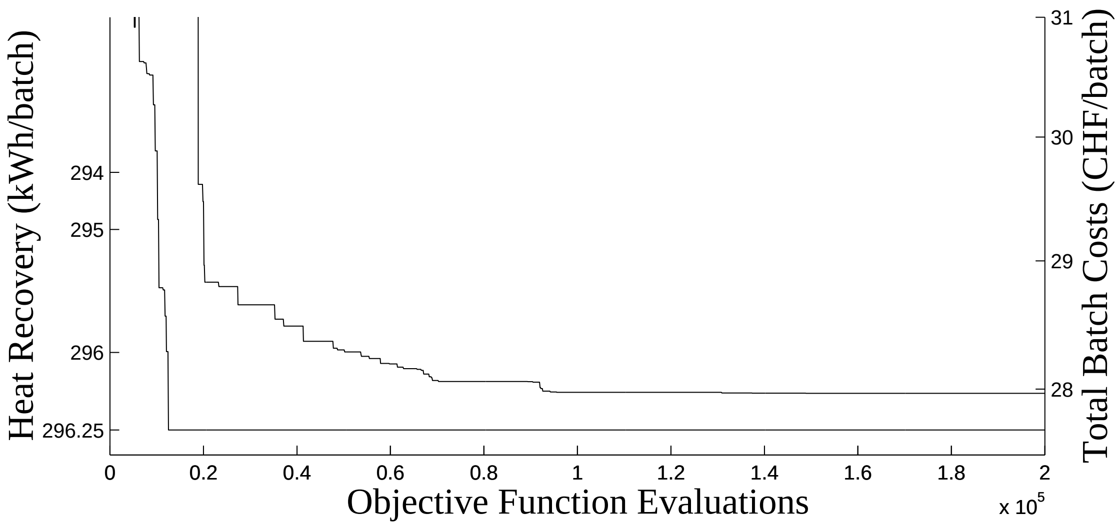


Figure 6.18: Convergence for the second objective on the heat recovery problem EP1. The bottom line (left scale) shows a rapid convergence to the maximum possible heat recovery, once the limit is met, in remains to minimise the cost at with the best heat recovery is found (top line, right scale).

This time, in less that 65,000 function evaluations, QMOO had found a cheaper solution than found earlier, this time with a heat recovery of 295.8 kWh/batch—much closer to the limit, and significantly, meaning that after this point the *entire population* dominated Krummenacher's early solution. QMOO also found a second structure that performed as well as the original, however, though the structure differed in one of the structural variables from that found by Krummenacher, it actually represented the same design configuration.

QMOO was run for 200,000 function evaluations, though convergence was largely finished after 100,000 evaluations. The convergence history for this problem is shown in Figures 6.17 and 6.18, and the final

[6]All other parameters—population size, operators used, etc. were at default values.

[7]Though this is a pretty extraordinary result, not too much attention should be paid to it—it may just have been good luck. A real measure of the speed up would require a number of tests on the same problem.

population, after 200,000 function evaluations, in objective space, in Figure 6.19.

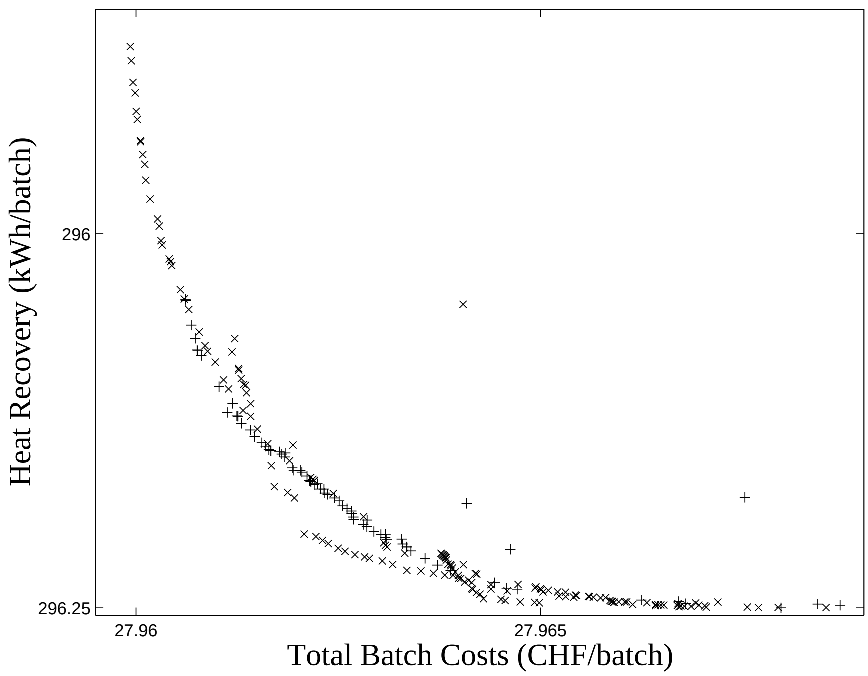


Figure 6.19: Final population in objective space for the heat recovery problem EP1. Two configurations are in the final population. '$+$'s show designs of the same configuration as found by Krummenacher, '$\times$'s show designs of the 'new' configuration. Note the *very* small scale, and that Krummenacher's solution is well above and to the right of the graph.

In the near future, QMOO will be run on other problems presented in Krummenacher's work. There is significant interest in this from LENI, Mr. Krummenacher, the Swiss Federal Energy Office, and several of the industrial partners who provided cases for Mr. Krummenacher's thesis.

6.6 Other Applications of QMOO

QMOO has been applied to a large number of problems with little or no participation by this author. Some of these problems are listed here, along with the benefits of applying QMOO to each problem. For more details on any of the cases, the reader is referred to the original references.

6.6.1 District Heating

The 'District Heating' problem was entirely treated by Adam Molyneaux, and is described in Molyneaux [89]. Molyneaux's work is a continuation of earlier work by Curti [23, 24], which adds detail to the descriptions of the methodology, and compares the results of the earlier work (with single-objective optimisation), to newer results with multi-objective optimisation. The reader is referred to Molyneaux [89] for more information on district heating systems.

The district heating problem had a significant effect of the development of QMOO. It is very multi-modal, and thus lead to the development of the grouping techniques, and has difficult-to-find tail solutions, which lead to the development of tail preservation.

Multi-objective optimisation meant that QMOO was able to find a whole set of the earlier results[8] in a single optimisation run, and resulted in the discovery of a number of new solutions. Grouping proved essential to the convergence of the problem (given that the SGA's method of diversity preservation was being no longer being used), and also meant that QMOO found several new local optima in the solution domain (though these new regions are dominated by the optimal region found by Curti). Tail preservation improved convergence toward the ends of the POF of the 'main front', and meant that extra groups did not have to be added to find these regions. Overall, QMOO found several NDSs with less than a tenth of the objective function evaluations that the earlier method had taken to find a single point.

The district heating problem was the only application of QMOO where a distance other than Euclidean distance was used in the clustering algorithm. In some of the tests Molyneaux grouped individuals using only the difference in network temperature as measure of their similarity.

Notably, as part of this work, Molyneaux implemented a three-objective version of quadratic thinning. Details of its performance are not yet available.

6.6.2 Integrated Energy Systems for Future Cities

QMOO has been used in a number of projects for optimising models of integrated energy systems for modern and future cities. In most of the projects, QMOO is used to optimise the configuration of a electricity and district heating and co-generation plant, looking at integrating modern technologies such as heat pumps and fuel cells into these systems. The objectives are measures of system efficiency (such as specific fuel consumption), pollution ($\mathrm{CO_2}$ emissions) and cost.

Bürer et al. [12] optimises the design and operation of a cogeneration plant for the city of Tokyo. The plant integrates a solid oxide fuel cell-gas turbine combined cycle and heat pumps. The objectives are to minimise $\mathrm{CO_2}$ emissions and total energy cost. The resulting NDS is used to calculated marginal costs of $\mathrm{CO_2}$ emissions, and thus investigate appropriate levels for a carbon tax.

Bürer et al. [11] investigates a similar system for the city of Beijing. Here, fuel cells are not considered, but the whole network is modelled, rather than just the central plant, and heat pumps can either in the plant, or where the heat is used. The results are used in the same manner as before to calculate marginal costs of pollution, and thus estimate tax levels.

It is interesting to note that in both of these studies, which are based on earlier work using environomic optimisation, that the place of the emissions cost has completely changed. Before, finding the correct emissions cost for a system study was a difficult task, and results of the optimisation had a large uncertainty because

[8]Curti performed a parameter study on network temperatures by running the optimisation several times with different temperatures, while Molyneaux used network temperature (a decision variable) as a second objective, to the perform the parameter study in one shot.

of this. Now, the results of the optimisation are used to determine that cost.

Li et al. [80] also analysed a heating system for Beijing, this time using cost and exergy-based specific consumption as objectives. Total energy costs and specific consumption were optimised for a range of payback periods for the plant.

In all the studies QMOO was used without any tuning of control parameters, apart to select a number of groups, and grouping was used without resort to any special distance functions. However, for some cases groups had to be 'forced', by re-running optimisations with some variables fixed. No other optimisation methods or algorithms were tried, but QMOO was found to be rapid and robust, and entirely satisfactory.

6.6.3 Integrated Energy Systems for Isolated Rural Areas

The POLEDURME (pôles énergétiques intégrés pour un développement durable en régions méditerranéennes) project [100, 113] studies energy systems for isolated rural areas, such oases in the deserts of Tunisia. The systems studied are the efficient use of geothermal resources, reduction and recovery of rejected heat from chemical industries, and food industry projects in food production zones. A larger number of criteria are considered than in most of the other studies.

In the most remote communities, drinking water resources are decreasing due to overconsumption and the rising salinity of underground water. Water quality is becoming a major problem in these regions. Other notable problems are the availability of electricity, the risks of pollution of sensitive environments, and noise pollution from generating equipment.

As with the projects above, QMOO was used without any tuning of control parameters, and worked very well. Grouping was used, again without any special preparation, and was found to work. Early in the optimisation phase of the project, GEATbx [104] was also used, but QMOO was found to faster, easier to use and more reliable.

Chapter 7

Conclusions

7.1 Multi-Objective Optimisation

Multi-objective optimisation techniques are applicable to a wide range of optimisation problems. They can be used in cases where there is clearly more than one objective, where they give better results than a variety of methods relying on weighting, constraints or other methods of transforming the problem to a single objective, and can simplify the problem formulation. They can also be used in cases where there are not clearly separate objectives—by making a decision variable also an objective, a parameter study can be performed, and MOO can be used to study the trade-offs between easily addable, but conflicting objectives.

The results of a multi-objective optimisation give the decision-maker a far broader view of the interaction between objectives than a single objective optimisation. In the case of the energy system work presented here, this means that environmental issues can be taken account of very early in the decision process at the same time as the best economic and engineering choses are being investigated. Investigating a wider range of results offers a more complete understanding of the issues surrounding the design of an energy system, such as a heat and power plant, and hopefully means that the system built will be more sustainable.

7.2 QMOO

QMOO is a very rapid multi-objective optimisation algorithm that uses unique methods for preserving the diversity of the population.

Most of the applications show that QMOO's performs well with very little tuning to a particular problem, though the Shanxi problem shows that tuning can improve performance.

QMOO's grouping methods prove their value on a number of applications, where they allow the discovery of several local optima and improve convergence to difficult to find optima. In all cases except the district heating problem, grouping used Euclidean distance as the measure of similarity between individuals. This is in contrast to earlier work with the SGA, where some care had to be taken in defining the distance function used to find an individual's nearest neighbour.

Evolutionary operator choice improves QMOO's performance over a range of test problems, sometimes performing better than any of the single operators, but mostly not quite well as the best possible operator for that problem. However, using EOC is probably a better option than spending time trying to find the best possible operator for a particular problem. The varying performance of the different operators over the set of test problems suggests that an investigation of the operators used for multi-objective problems could well be worth while. A brief study of the mutation operators used here showed that having the right mutation operators can be critical for the performance of certain combination operators.

Methods for removing points from a NDS that attempt preserve both an even distribution of points across the NDS, and those points that are estimated to be closer to the POF, improve performance over the full range of test problems. Methods that preserve only one or the other have much less consistent performance. Testing of the methods gives two 'surprise' results: quadratic thinning outperforms dominated volume thinning, even though performance is measured by dominated volume and the excellent performance of random quadratic thinning on some problems. It remains to be seen how these methods perform compared to a method that preserves the entire non-dominated set.

Tail preservation improves convergence on all problems where it is appropriate, at only a very small cost on problems where it is clearly inappropriate. Seemingly large tail regions (10% of the linear dimensions of the NDS) can make a considerable difference to performance. Tail preservation, like the thinning methods, and unlike the choice of combination operators, has little effect on the algorithm's early convergence rates. By preserving useful diversity, tail preservation means that that continued convergence rates, after initial convergence has bottomed out, are higher than for methods not preserving tails.

In addition to the techniques tested, QMOO's extreme elitism probably has a significant effect on its performance, both in making it faster, and in posing problems with preservation of diversity.

QMOO's queue based structure has been exploited to allow parallelisation of the algorithm, and to make the change between grouping and non-grouping versions of the algorithms transparent. Although a little unconventional, it makes for a very flexible EA that can make the best use of available computing resources.

7.3 Applications

QMOO added value on all the 'real problems' to which it was applied. Some of the problems had been solved earlier by single objective methods, and QMOO proved faster, brought the advantages of multi-objective methods, and found more solutions that the older methods. Other problems had not been solved before, and QMOO enabled these problems to be optimised rapidly, with minimal set up or tuning, and again, found a range of solutions.

In the Shanxi coal problem, multi-objective optimisation illustrates the trade-offs between transport and construction costs, and shows that what are probably the most realistic solutions offer near-optimal performance in both objectives. QMOO's performance, after the addition of a number of problem specific operators proves considerably better than that of another EA.

Results from the hybrid problem provide insight into the utility of hybrid vehicles. It is known hybrid vehicles achieve high overall efficiency by allowing the internal combustion engine (ICE) to run closer to its best operating regime, and by stopping the ICE when the vehicle is stopped. Results from the optimisations show that an optimally dimensioned hybrid has an ICE size appropriate for the vehicle's normal cruising operation, while conventional drivetrains must use ICEs dimensioned for peak power demands, meaning they are less efficient for normal operation.

Treating the load scenario and grand composite problems using an EA means that a simple problem formulation can be used compared to the formulations required to treat the problems with other methods. This can save the decision-maker a lot of time. Multi-objective results for the load scenario problem show that the trade-off between total error and maximum error is quite marked, and that intermediate solutions exist that offer near optimal performance in both objectives. In the grand composite results, multi-objective results mean that the decision-maker can clearly see the advantages of adding cuts (and so equipment) in a plant heat integration problem, and thus see at what point it is no longer worthwhile to add equipment.

Solving the heat integration problem, however, does not result in a clear view of the trade-offs between batch costs and heat recovery per batch—the two objectives are almost non-conflicting. However, the multi-objective approach, along with grouping and tail preservation, mean that QMOO is able to very quickly find solutions better than those found by an algorithm that was carefully tuned to the problem, and which used a problem specific (two-level and two-stage) optimisation algorithm.

Work on a number of other problems have demonstrated the utility of the multi-objective approach, and shown that QMOO is a very rapid, and easy to use optimisation algorithm. The most encouraging result of the work of other researchers is that in all cases, they applied QMOO with very little 'tuning', and in all cases, it worked rapidly from the first optimisation attempt.

7.4 Future Work

7.4.1 Applications

The Shanxi problem was originally posed as problem about trade-offs between transport pollution and production pollution. However, information on the pollutants was not available, and so cost objectives were optimised. More complete pollution data is now available, and so the problem should be re-optimised taking into account the new objectives. It will also be interesting to see the trade-offs between cost and pollution for the problem.

The results for the heat integration problem are very encouraging. However, only one example from Krummenacher's thesis is treated here. Very shortly, other problems from the thesis, some of them larger than EP1, will be tested. The results are of interest to a number of groups.

7.4.2 Algorithm

QMOO is ‘instrumented’ to provide information about the workings of EOC—the operators that created an individual are recorded, and examining this information provides insights into how EOC works. Though this information is available, it has not been investigated here. In the near future the data collected while running the tests for Chapter 5[1], will be studied.

Equally, though the mutation operators were chosen with some care, and it has been shown that they are all of considerable importance, they have not been the subject of detailed investigation. Studying mutation operators, as well as the effects of adding more combination operators (for example simulated binary crossover) could be well worthwhile.

No investigation was made into different methods of managing elitism. It would seem that QMOO’s elitism management (keeping only the best individuals, with very few exceptions) works well, though it clearly causes problems with diversity preservation. Despite QMOO’s flexibility in other aspects of the algorithm, adding other elitism management methods to the algorithm is not trivial. Though the results of such tests will probably be interesting, it seems likely that changes will result in worse performance, and so tests into elitism will not be of the highest priority.

The results of test runs raise some interesting questions about some of the methods tested. Both the results of random quadratic thinning and large tail regions on some of the problems were unexpected. Investigating what makes these methods perform so well, and in what cases, will hopefully lead to the development of methods that offer similar performance, but over a wider range of problems, and with less effect on the variance of the convergence curves.

Using MOO for parametric studies, and the tail preservation methods developed here, hint at another possibility for the use of multi-objective optimisers: constraint satisfaction problems. Here, the goal is not to find the optimum of a system model, but to explore the region in which the model is feasible. By combining the methods for parametric studies with optimisation in several directions (or, if you like preserving tails as large as the main front) a constraint satisfaction algorithm could be constructed. Whether this method would be competitive with existing methods for constraint satisfaction remains to be seen.

The work in this thesis has been limited to two objectives. There are a number of reasons for this. Firstly, with more objectives, a larger population is needed to provide good resolution of the POF, and this places strain on both the ranking and grouping algorithms. Secondly, the thinning methods used here have only been implemented for two objectives. Molyneaux [89] implemented a three-objective version of quadratic thinning, but thinning by dominated volume with more than two objectives is probably impractical[2] Like the thinning methods, tail preservation is currently only implemented for two-objectives.

All these problems can probably be alleviated to some extent by the use of the new rank management techniques made available by Everson et al. [43]. Not only will ranking speed improve, but it will be possible to test whether thinning really offers anything compared to keeping the entire NDS, and tail preservation

[1] Which took some two weeks of computer time to generate, and occupies several gigabytes of disk space.

[2] So it is quite fortunate that it is not the best thinning method.

should become considerably easier. QMOO will use Everson et al.'s non-dominated and dominated trees in the very near future.

Appendix A

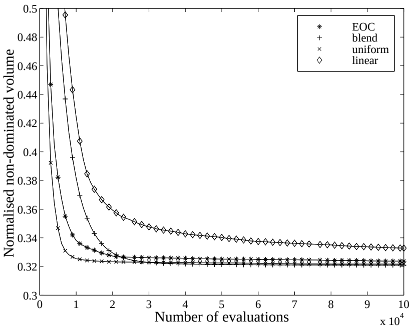
All Graphs from Test Runs

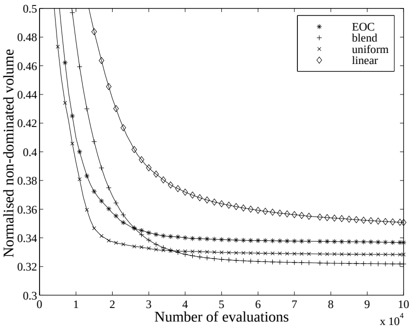
Graphs of test runs for all problems, problem sizes and population sizes are presented here without comment. Each page contains twelve graphs. Across the page they correspond to three problem sizes : 15, 30 and 60 variables (except in the case of ZDTE1, where there are 10, 20 and 40 variables). Down the page the population sizes changes: 48, 64, 80 and 96 individuals.

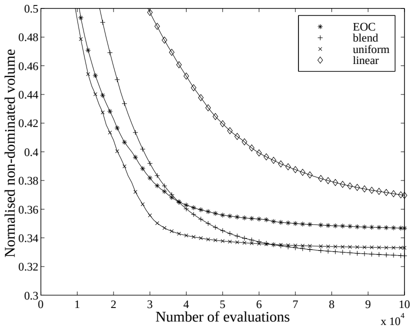
Graph scales are the same for all population sizes for a problem of the same size. The scales for different problem sizes change occasionally in order to give a clearer view of the convergence curves.

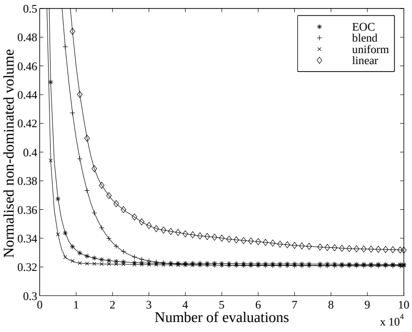
A.1 Evolutionary Operator Choice

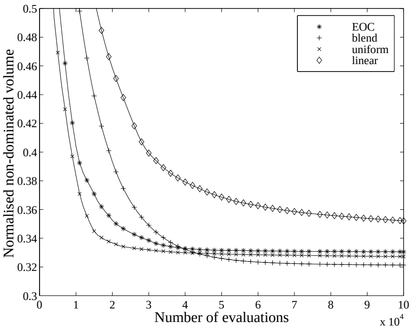
A.1.1 ZDT6

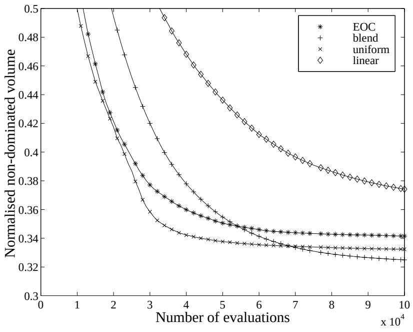


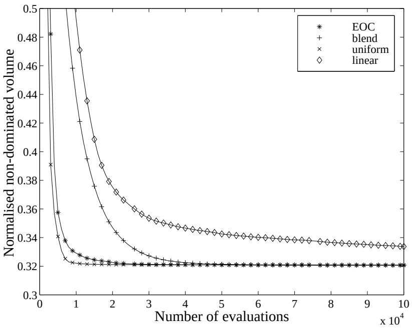


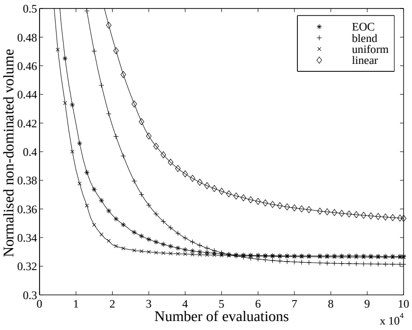


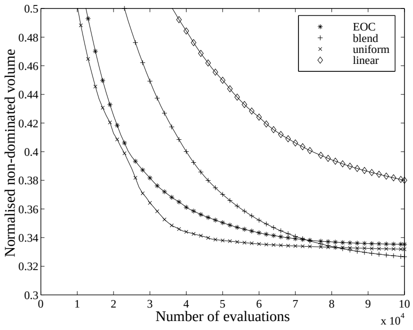


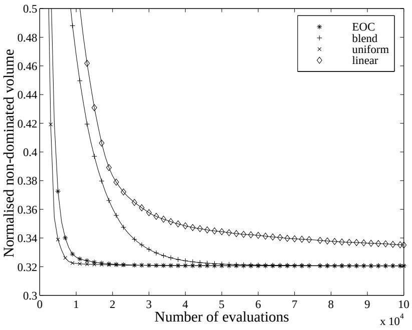


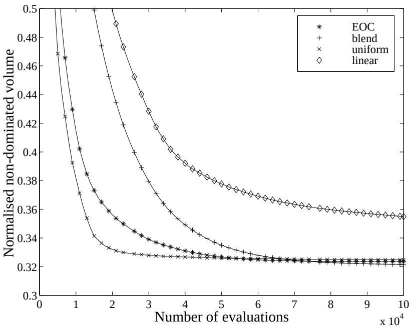


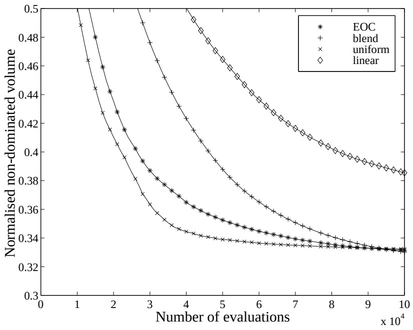




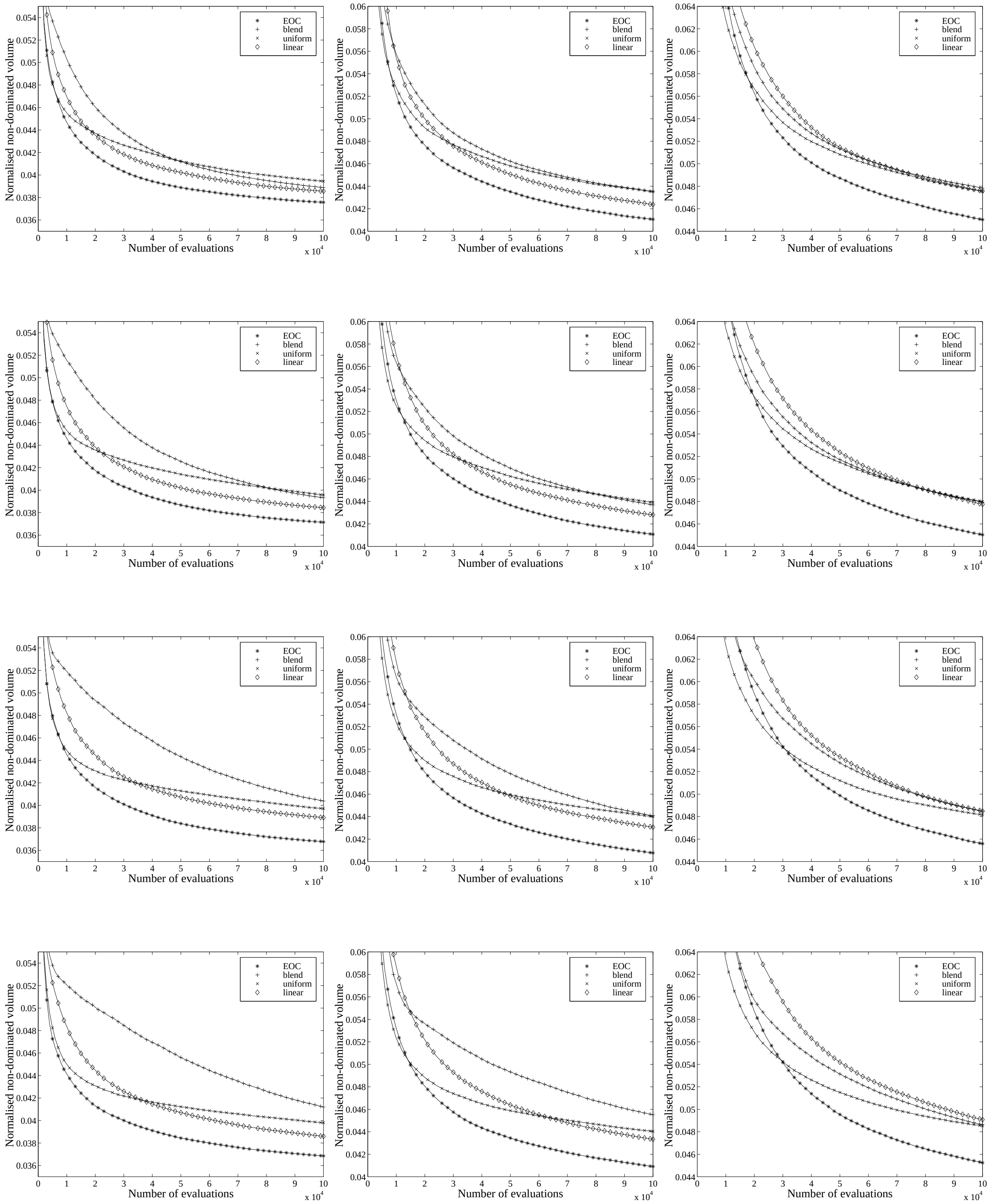




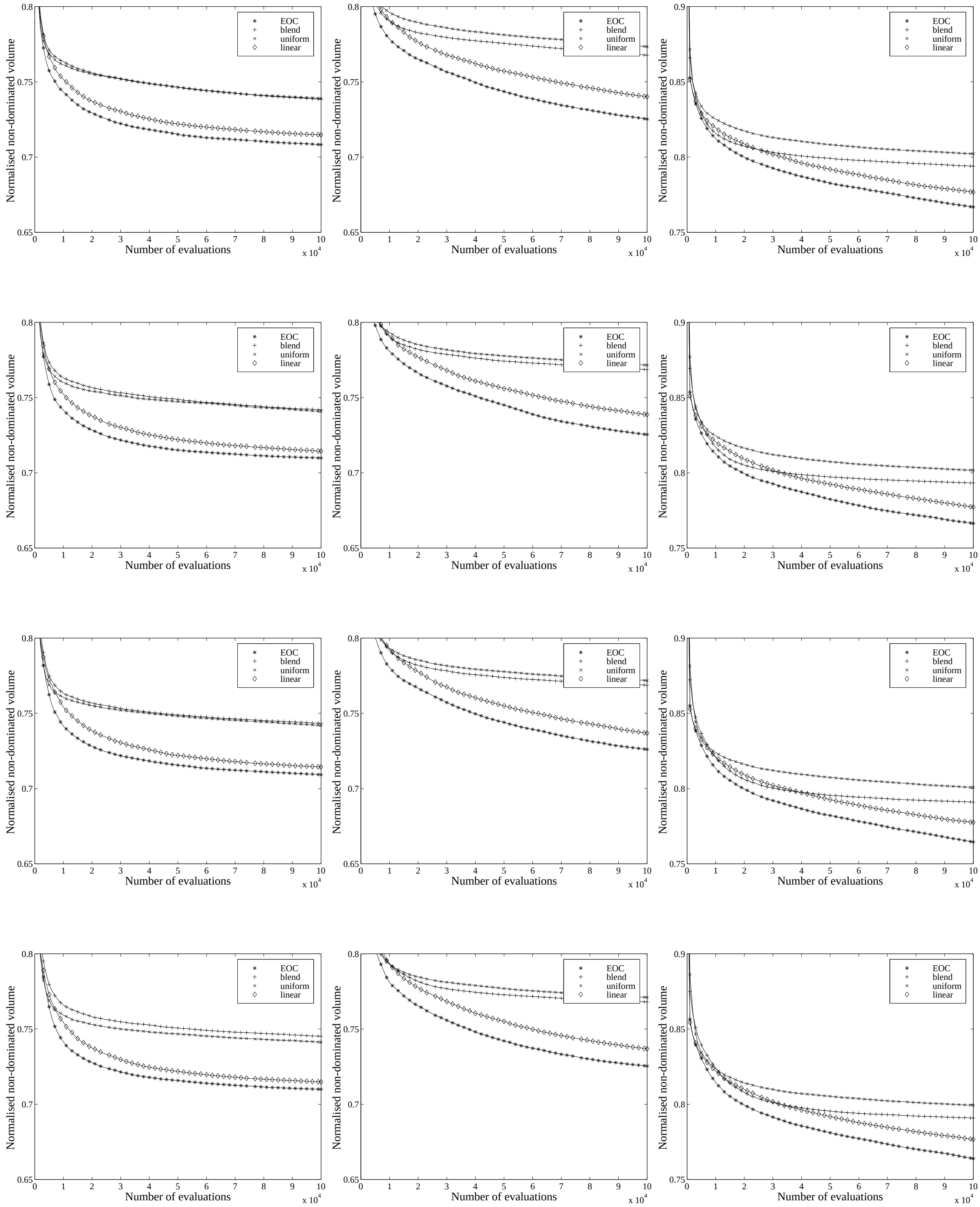




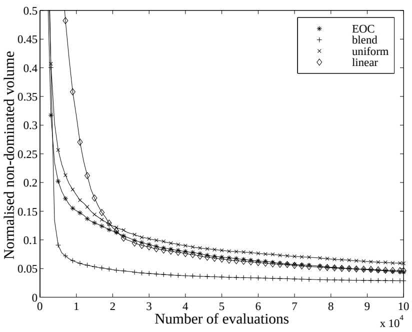
A.1.2 ZDTE1

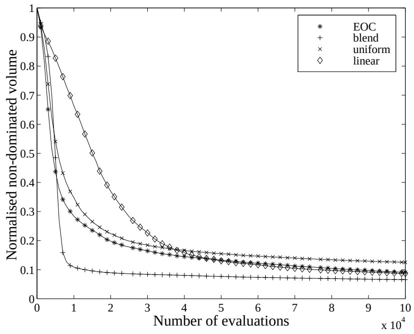

EOC
blend
uniform
linear
Normalised non-dominated volume
Number of evaluations
x 10^4

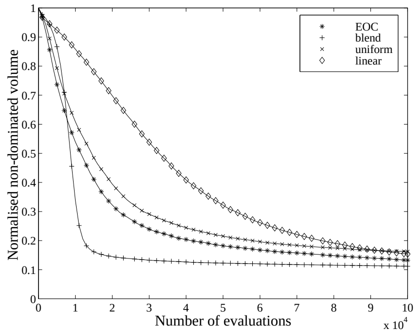
A.1.3 QV

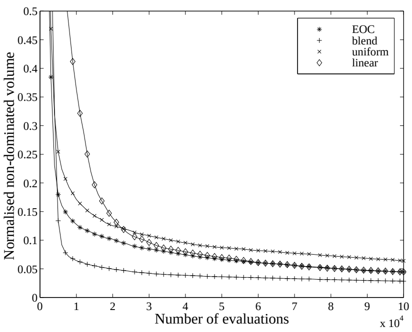

EOC
blend
uniform
linear
Normalised non-dominated volume
Number of evaluations
x 10^4

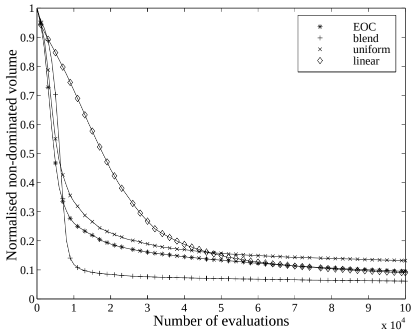
A.1.4 KUR

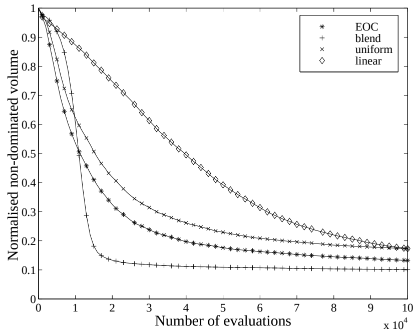

EOC
blend
uniform
linear
Normalised non-dominated volume
Number of evaluations
x 10^4

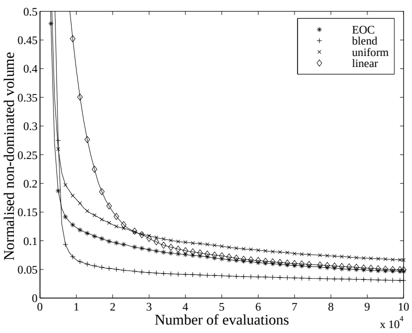

EOC
blend
uniform
linear
Normalised non-dominated volume
Number of evaluations
x 10^4

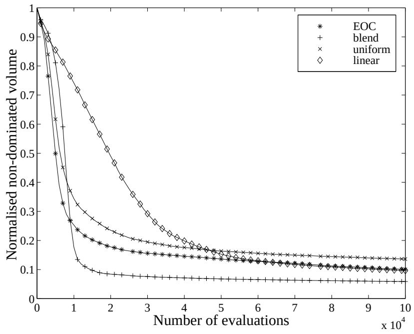

EOC
blend
uniform
linear
Normalised non-dominated volume
Number of evaluations
x 10^4

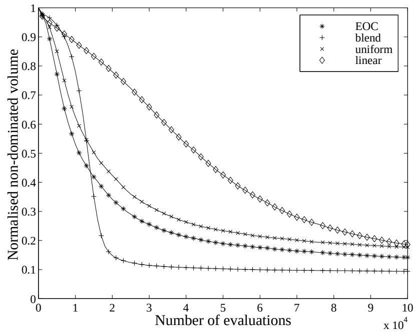

EOC
blend
uniform
linear
Normalised non-dominated volume
Number of evaluations
x 10^4

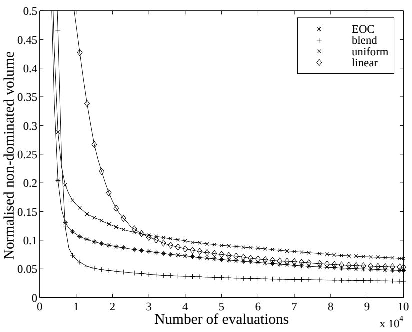

EOC
blend
uniform
linear
Normalised non-dominated volume
Number of evaluations
x 10^4

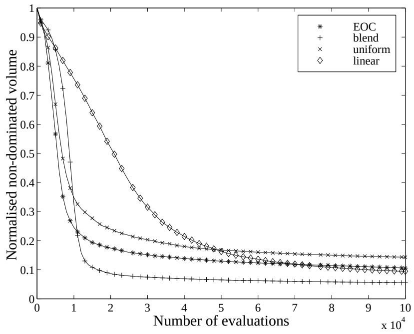

EOC
blend
uniform
linear
Normalised non-dominated volume
Number of evaluations
x 10^4

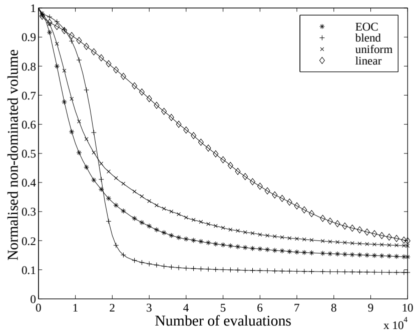

EOC
blend
uniform
linear
Normalised non-dominated volume
Number of evaluations
x 10^4


EOC
blend
uniform
linear
Normalised non-dominated volume
Number of evaluations
x 10^4


EOC
blend
uniform
linear
Normalised non-dominated volume
Number of evaluations
x 10^4

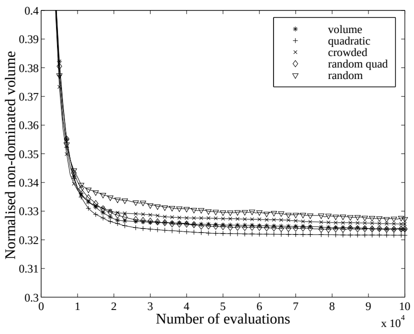

EOC
blend
uniform
linear
Normalised non-dominated volume
Number of evaluations
x 10^4

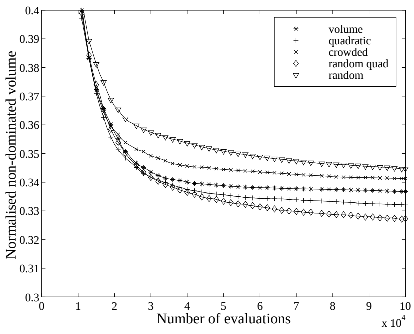

EOC
blend
uniform
linear
Normalised non-dominated volume
Number of evaluations
x 10^4

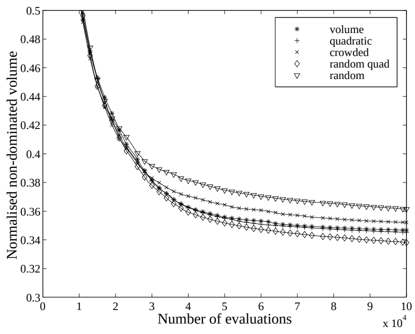

EOC
blend
uniform
linear
Normalised non-dominated volume
Number of evaluations
x 10^4

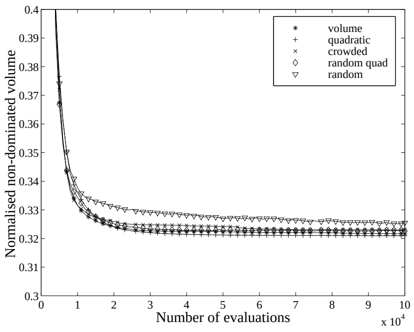
A.2 Thinning

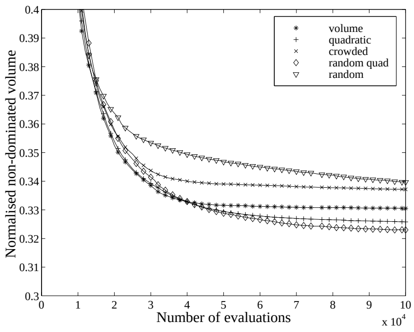
A.2.1 ZDT6

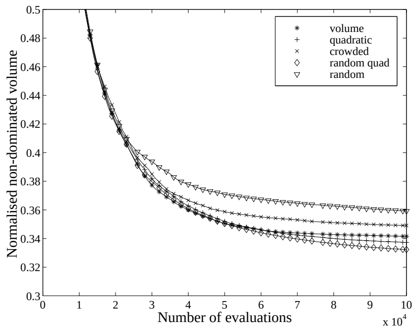


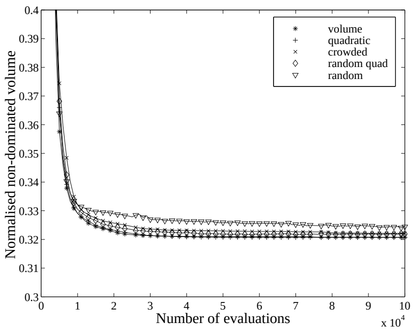


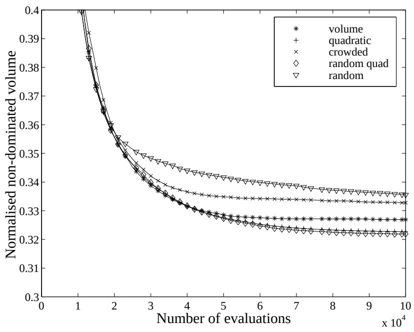


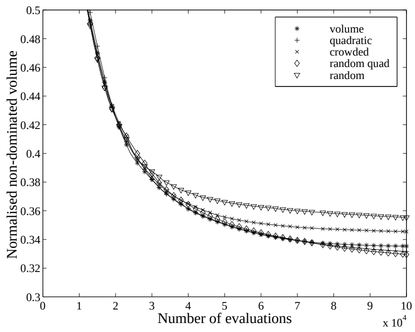


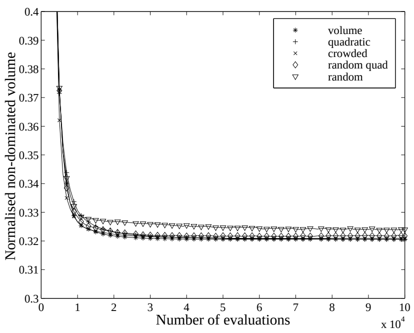


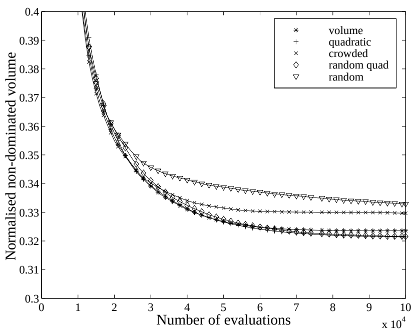


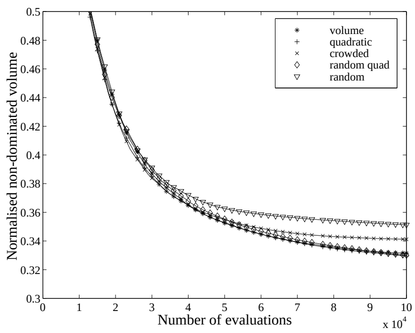




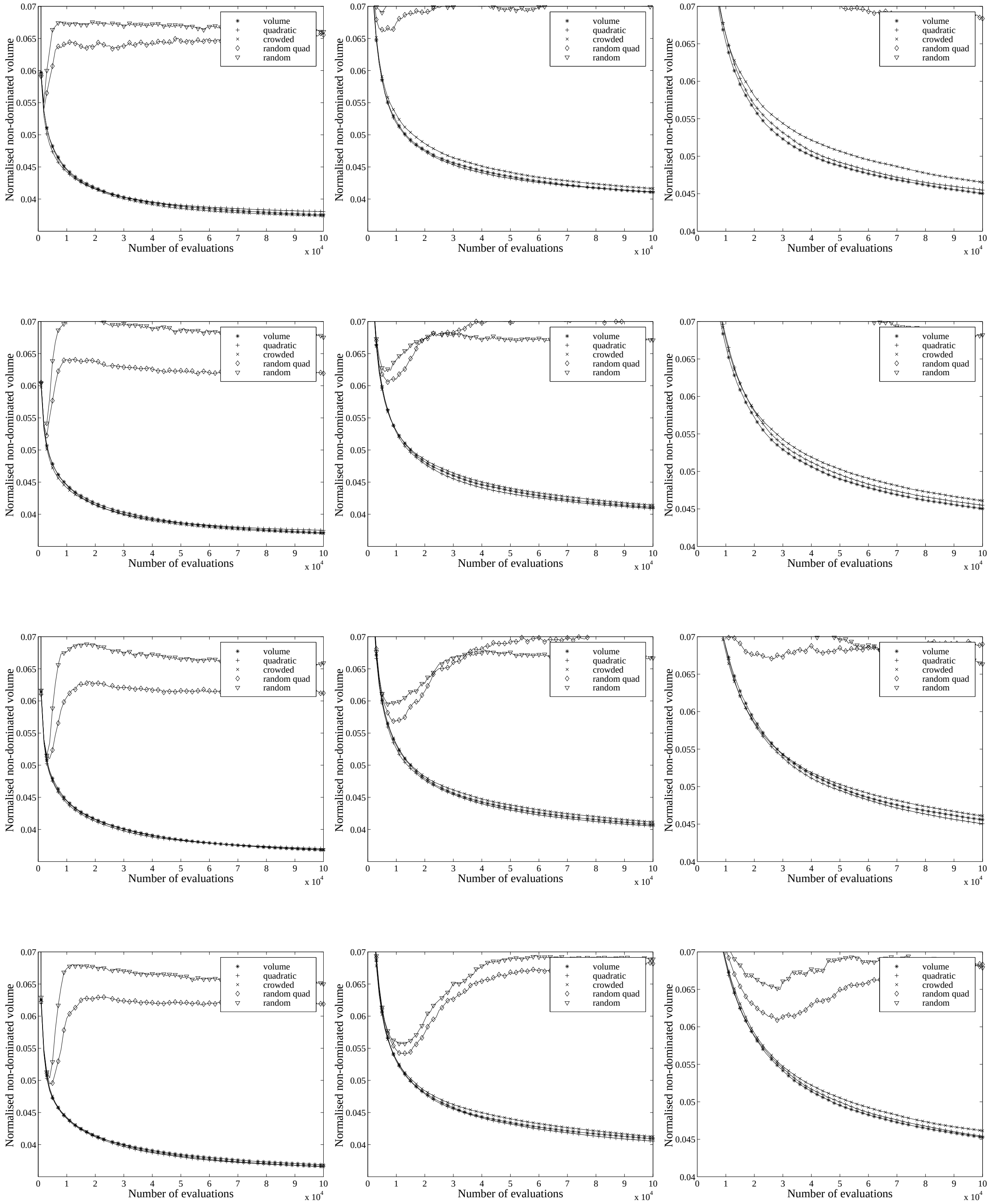




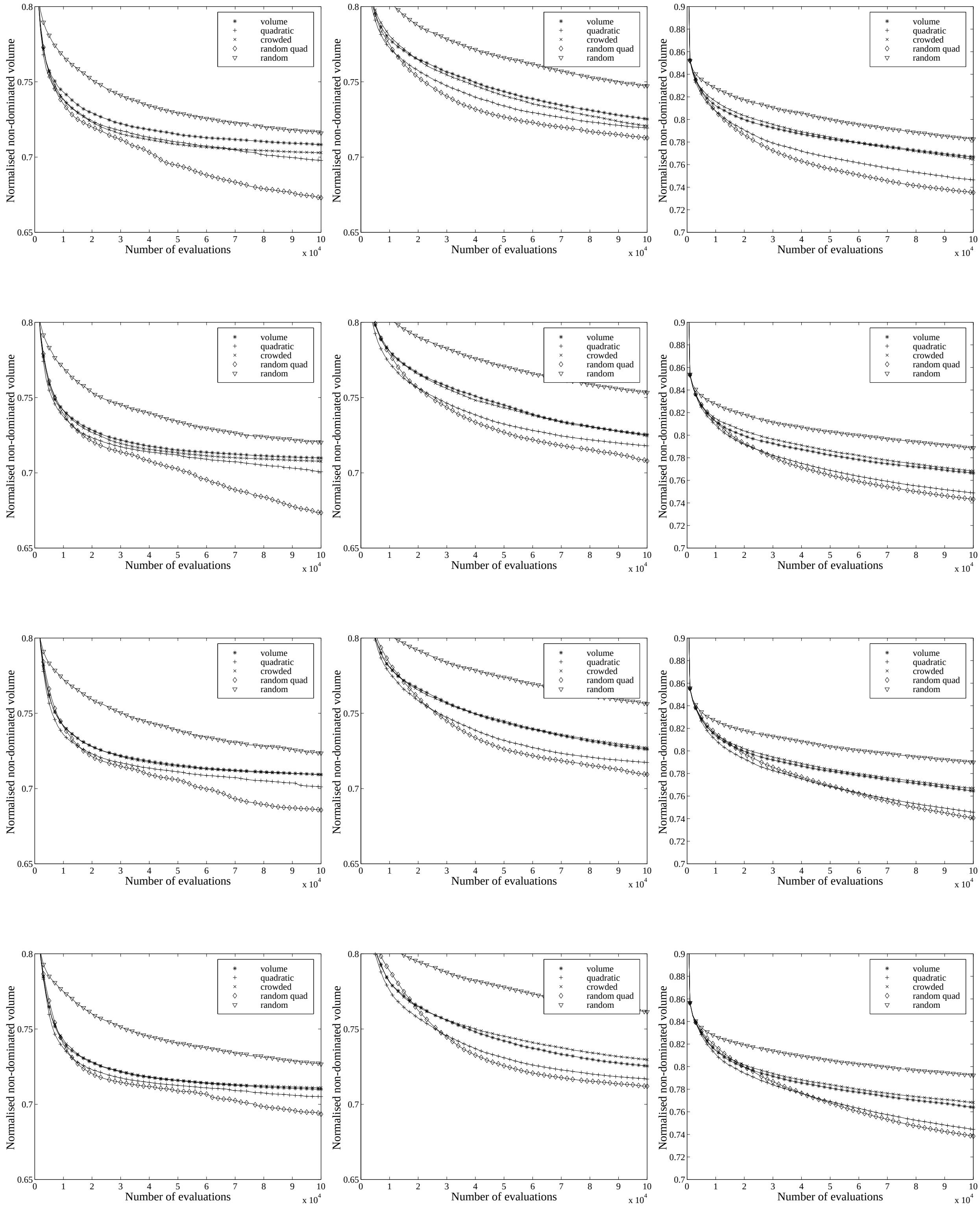




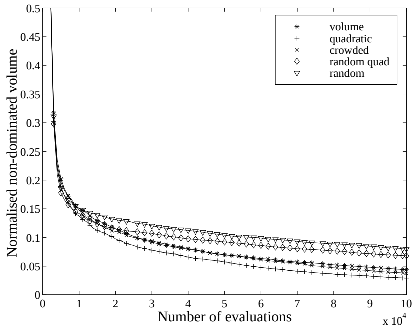
A.2.2 ZDTE1

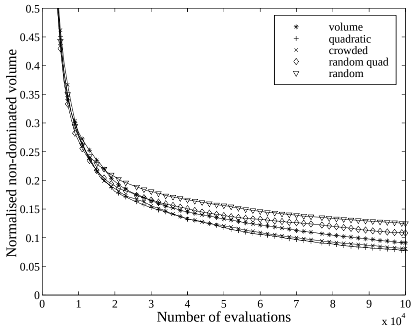

volume
quadratic
crowded
random quad
random
Normalised non-dominated volume
Number of evaluations
x 10^4

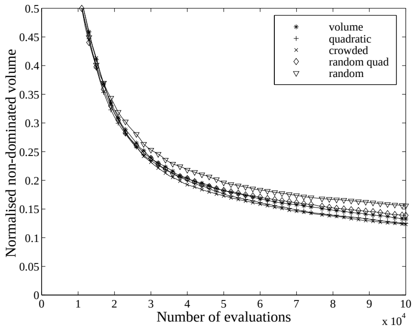
A.2.3 QV

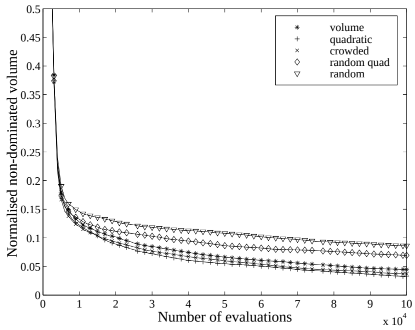

volume
quadratic
crowded
random quad
random
Normalised non-dominated volume
Number of evaluations
x 10^4

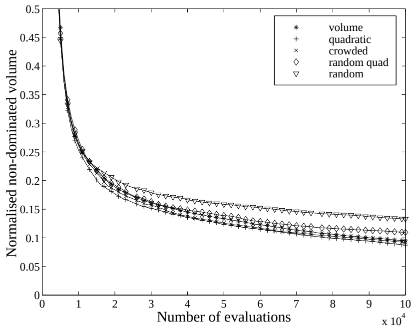
A.2.4 KUR

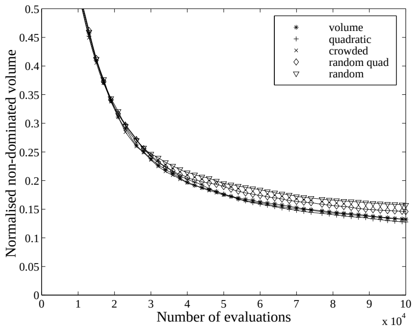

Normalised non-dominated volume
Number of evaluations
x 10^4
volume
quadratic
crowded
random quad
random

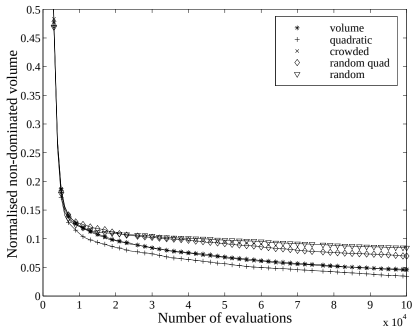

Normalised non-dominated volume
Number of evaluations
x 10^4
volume
quadratic
crowded
random quad
random

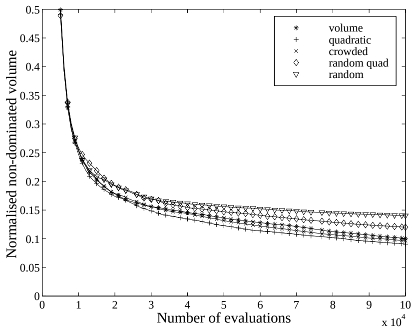

Normalised non-dominated volume
Number of evaluations
x 10^4
volume
quadratic
crowded
random quad
random

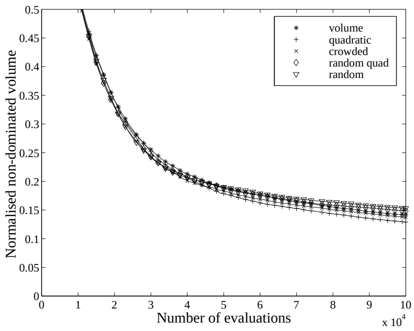

Normalised non-dominated volume
Number of evaluations
x 10^4
volume
quadratic
crowded
random quad
random

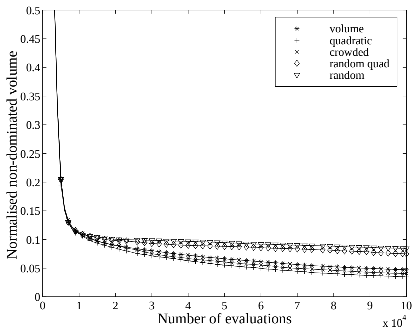

Normalised non-dominated volume
Number of evaluations
x 10^4
volume
quadratic
crowded
random quad
random

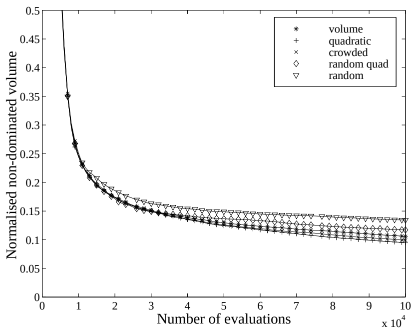

Normalised non-dominated volume
Number of evaluations
x 10^4
volume
quadratic
crowded
random quad
random

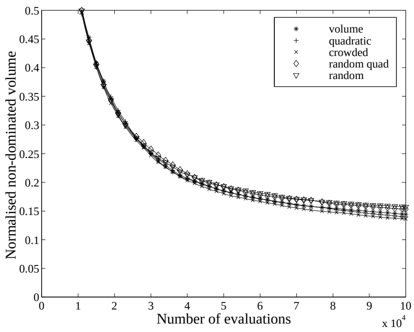

Normalised non-dominated volume
Number of evaluations
x 10^4
volume
quadratic
crowded
random quad
random


Normalised non-dominated volume
Number of evaluations
x 10^4
volume
quadratic
crowded
random quad
random


Normalised non-dominated volume
Number of evaluations
x 10^4
volume
quadratic
crowded
random quad
random

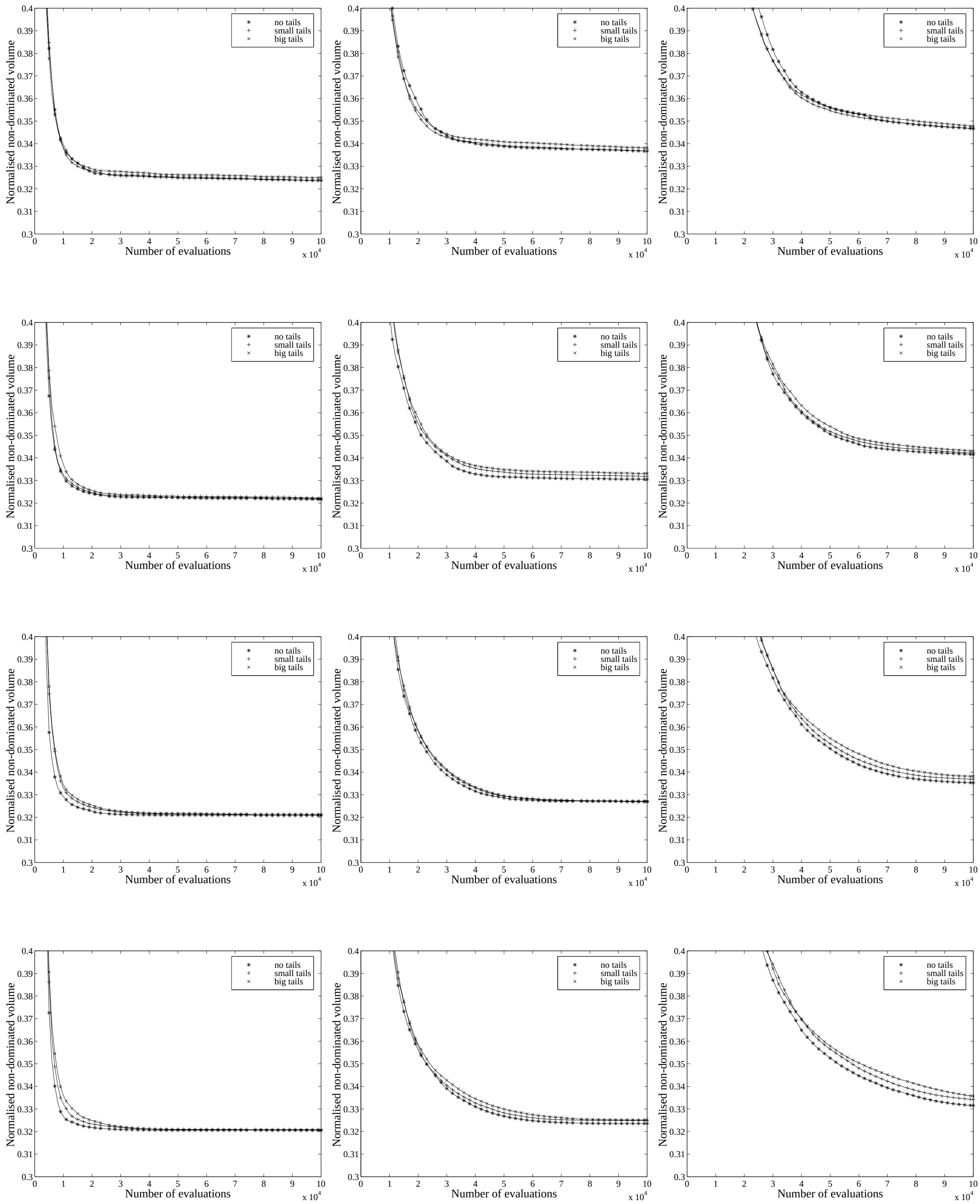

Normalised non-dominated volume
Number of evaluations
x 10^4
volume
quadratic
crowded
random quad
random


Normalised non-dominated volume
Number of evaluations
x 10^4
volume
quadratic
crowded
random quad
random

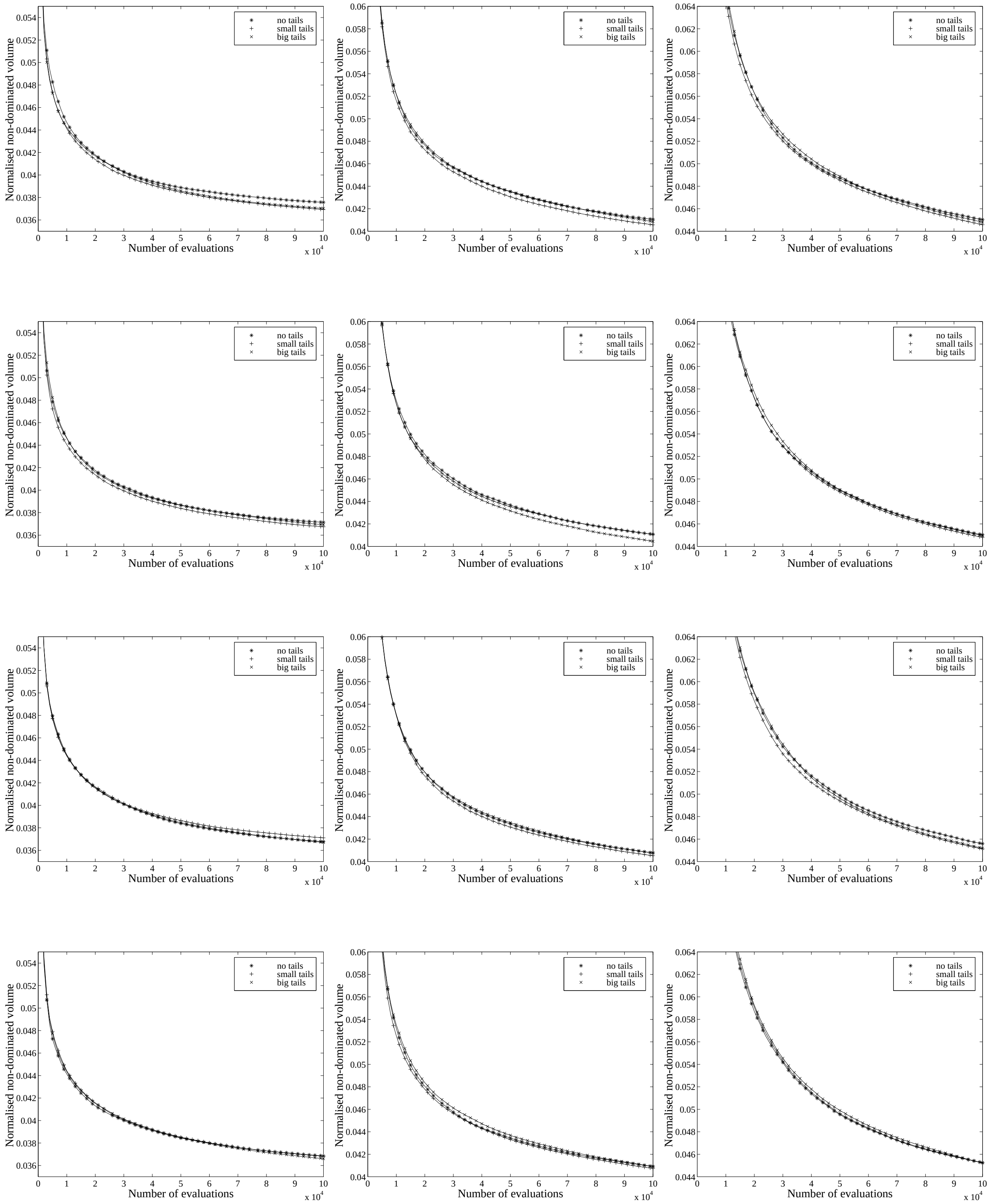

Normalised non-dominated volume
Number of evaluations
x 10^4
volume
quadratic
crowded
random quad
random

A.3 Tail Preservation

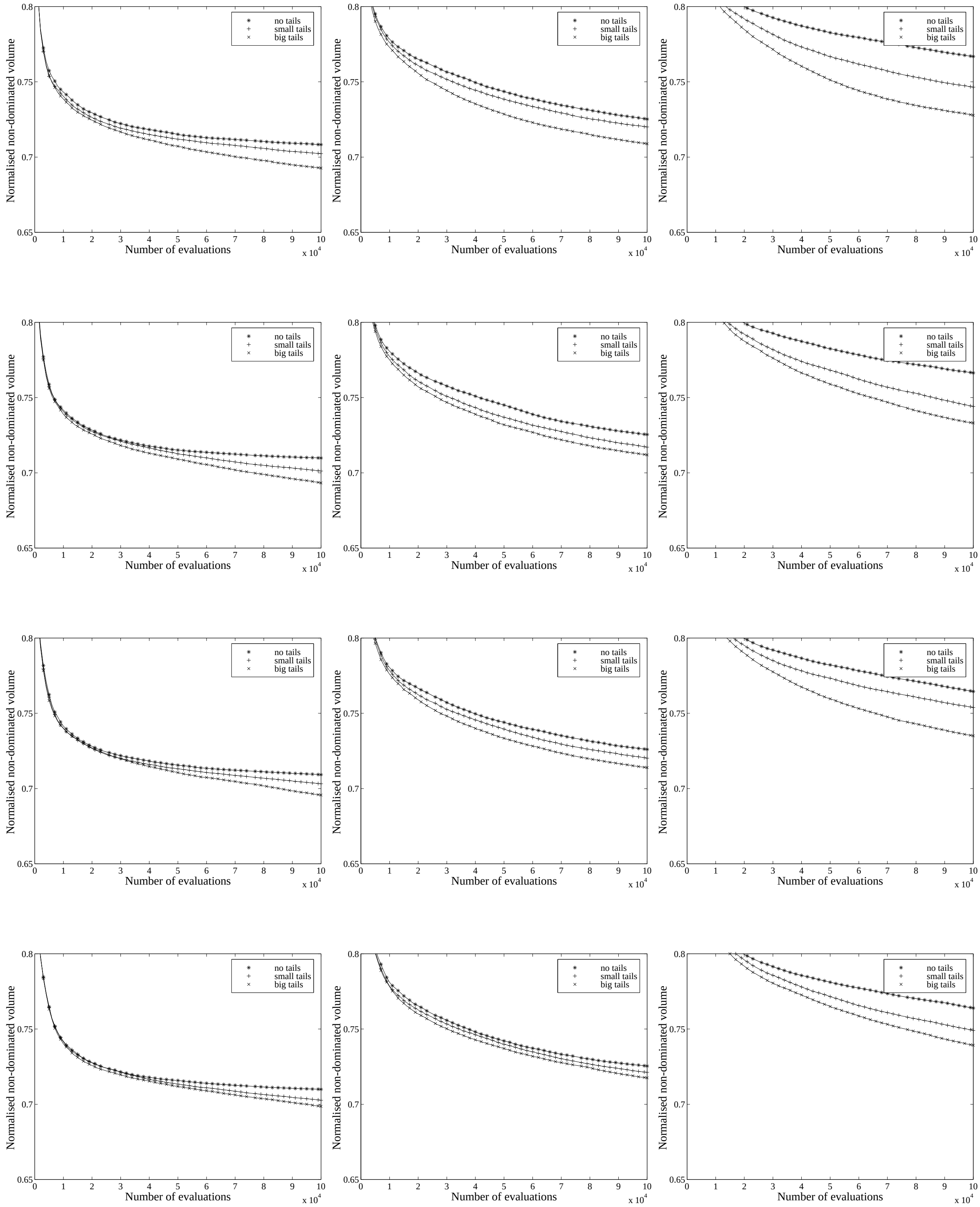
A.3.1 ZDT6



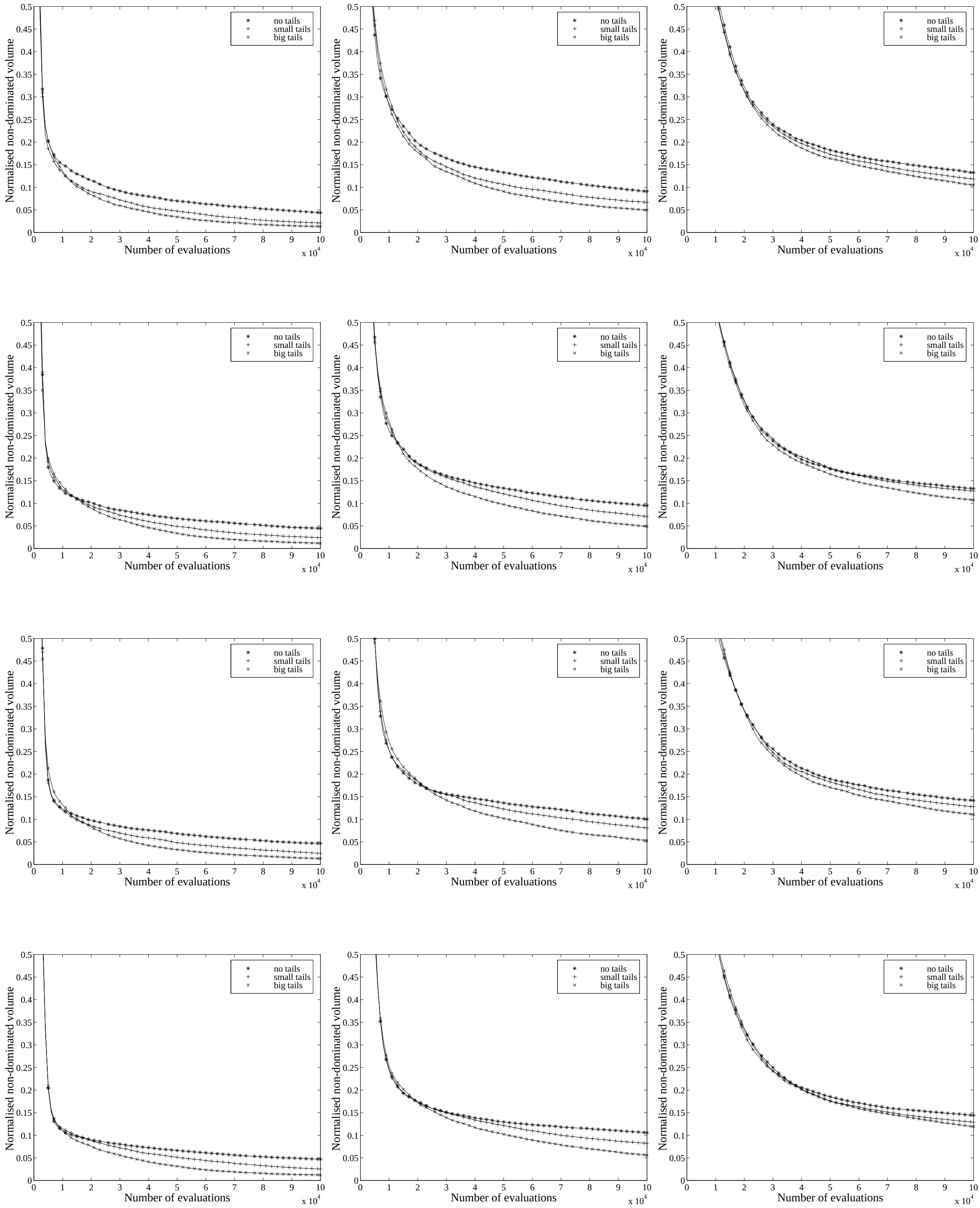
A.3.2 ZDTE1


no tails
small tails
big tails
Normalised non-dominated volume
Number of evaluations
x 10^4

A.3.3 QV


no tails
small tails
big tails
Normalised non-dominated volume
Number of evaluations
x 10^4
0.8
0.75
0.7
0.65
0 1 2 3 4 5 6 7 8 9 10

A.3.4 KUR


no tails
small tails
big tails
Normalised non-dominated volume
Number of evaluations
x 10^4

References

[1] T. Akatsuka. Modeling and evaluation of the transportation sector in the coke-making industry of Shanxi Province, China (in Japanese). Master's thesis, University of Tokyo, 2001.

[2] J. Alander and J. Lampinen. Cam shape optimisation by genetic algorithm. In D. Quagliarella, J. Périaux, C. Polini, and G. Winter, editors, *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, pages 153–174. Wiley, Chichester, 1998.

[3] J. Andersson and D. Wallace. Pareto optimisation using the struggle genetic algorithm. Technical report, MIT CADlab, 2000.

[4] P. Arabie, L. Hubert, and G. De Soete. *Clustering and Classification*. World Scientific, 1996.

[5] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

[6] T. Bäck, T. Beielstein, B. Naujoks, and J. Heistermann. Evolutionary algorithms for the optimization of simulation models using PVM. In *Proceedings of EUROPVM '95*, pages 277–282, Paris, 1995. Hermes.

[7] T. Bäck, G. Rudolph, and H.-P. Schwefel. Evolutionary programming and evolution strategies: Similarities and differences. In D. B. Fogel and W. Atmar, editors, *Proc. 2nd Annual Conf. Evolutionary Programming*, pages 11–22, La Jolla CA, February 25–26, 1993. Evolutionary Programming Society, San Diego CA.

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

[9] L. Borgarello, R. Fontana, and D. Quagliarella. Global sourcing with families of components using genetic algorithms. In Giannakoglou et al. [56].

[10] G. E. Box. Evolutionary operation: A method for increasing industrial productivity. *Applied Statistics*, 6(2):81–101, 1957.

[11] M. Bürer, H. Li, , and D. Favrat. Multi-criteria optimization of a heat pump based district heating system in the city of bejing. In *Heat Pumps—Better by Nature. 7th International Energy Agency Heat Pump Conference*, Bejing, China, May 2002.

[12] M. Bürer, K. Tanaka, D. Favrat, and Y. K. Multi-criteria optimization of a district cogeneration plant integrating a solid oxide fuel cell—gas turbine combined cycle, heat pumps and chillers. in preparation for *Journal of Computers, Environment and Urban Systems*, 2002.

[13] E. Cantù-Paz. Designing efficient master-slave parallel genetic algorithms. IlliGAL Report 97004, Illinois Genetic Algorithms Laboratory, May 1997.

[14] R. A. Caruana and J. D. Schaffer. Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In *Fifth International Conference on Machine Learning*, pages 153–161, Los Altos, CA, June 1988. Morgan Kaufmann.

[15] D. J. Cavicchio. *Adaptive Search using Simulated Evolution*. PhD thesis, University of Michigan, Ann Arbour, MI, 1971.

[16] H. Chen. Technological evaluation and policy analysis for cokemaking: A case study of cokemaking plants in Shanxi Province, China. Master's thesis, Massachusetts Institute of Technology, 2000.

[17] X. Chen, X. Pan, C. Yang, K. R. Polenske, A. Shirvani-Mahdavi, and X. Liu. A sustainable development road for the cokemaking sector. *Alliance for Global Sustainability, January 1999.*, 1999.

[18] C. A. Coello Coello. *An Empirical Study of Evolutionary Techniques for Multiobjective Optimization in Engineering Design*. PhD thesis, Tulane University, New Orleans, Apr. 1996.

[19] C. A. Coello Coello. A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems*, 1(3):269–308, 1999. URL `citeseer.nj.nec.com/coello98comprehensive.html`.

[20] C. A. Coello Coello. An updated survex of evolutionary multiobjective optimization techniques: State of the art and future trends. *Knowledge and Information Systems*, 1(3), 1999.

[21] T. M. Cover. *Elements of Information Theory*. John Wiley & Sons, New York, 2 edition, 1991.

[22] F. Crittin and M. Bierlaire. New algorithmic approaches for the anticipatory route guidance generation problem. In *Swiss Transport Research Conference*, Ascona, Switzerland, 2001.

[23] V. Curti. *Modélisation et Optimisation Environomiques de Systèmes de Chauffage Urbain Alimentés par Pompes à Chaleur*. PhD thesis, École Polytechnique Fédérale de Lausanne, 1998.

[24] V. Curti, M. von Spakovsky, and D. Favrat. An environomic approach for the modeling and optimization of a district heating network based on centralized and decentralized heat pumps, cogeneration and/or gas furnace (Part I: Methodology). *International Journal of Thermal Sciences*, 39:721–730, 2000.

[25] P. Darwen. Black magic: Interdependence prevents principled parameter setting. In *Applied Complexity: From Neural Nets to Managed Landscapes*, pages 227–237, Christchurch, NZ, 2000. URL `citeseer.nj.nec.com/darwen00black.html`.

[26] D. Dasgupta and Z. Michalewicz. Evolutionary algorithms—an overview. In D. Dasgupta and Z. Michalewicz, editors, *Evolutionary Algorithms in Engineering Applications*, pages 3–28. Springer, Berlin, 1997.

[27] L. Davis, K. De Jong, M. Vose, and L. D. Whitley, editors. *Evolutionary Algorithms : Proceedings of Mathematics in High Performance Computing*, New York, 1999. Springer Verlag.

[28] C. M. M. de Fonseca. *Multiobjective Genetic Algorithms with Applications to Control Engineering Problems*. PhD thesis, University of Sheffield, Sheffield, UK, 1995. URL `citeseer.nj.nec.com/article/fonseca95multiobjective.html`.

[29] K. A. De Jong. *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Ann Arbour, MI, 1975.

[30] K. Deb. Multi-objective genetic algorithms: Problem difficulties and construction of test problems. *IEEE Transactions on Evolutionary Computation*, 7(3):205–230, 1999. URL `citeseer.nj.nec.com/deb99multiobjective.html`.

[31] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley, Chichester, 2001.

[32] K. Deb and R. Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9(2):115–148, 1995. URL `citeseer.nj.nec.com/deb95simulated.html`.

[33] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. KanGAL report 200001, Indian Institute of Technology, Kanpur, India, 2000. URL `citeseer.nj.nec.com/deb00fast.html`.

[34] K. Deb and H.-G. Beyer. Self-adaptation in real-parameter genetic algorithms with simulated binary crossover. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 172–179, Orlando, Florida, 1999. Morgan Kaufmann. URL `citeseer.nj.nec.com/deb99selfadaptation.html`.

[35] K. Deb and D. E. Goldberg. An investigation of niche and species formation in genetic function optimization. In J. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 42–50. Morgan Kaufmann, 1989.

[36] K. A. DeJong and J. Sarma. Generation gaps revisited. In *Foundations of Genetic Algorithms 2*, pages 19–28. Morgan Kaufmann, San Mateo, CA, 1993. URL `citeseer.nj.nec.com/dejong92generation.html`.

[37] S. Droste, T. Jansen, and I. Wegener. Perhaps not a free lunch but at least a free appetizer. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 833–839, Orlando, Florida, USA, 13-17 1999. Morgan Kaufmann. ISBN 1-55860-611-4. URL `citeseer.nj.nec.com/droste98perhaps.html`.

[38] S. Droste, T. Jansen, and I. Wegener. Optimization with randomized search heuristics—the (A)NFL theorem, realistic scenarios, and difficult functions. *Theoretical Computer Science*, 2001. URL `citeseer.nj.nec.com/droste97optimization.html`.

[39] R. Dugad and N. Ahuja. Unsupervised multidimensional hierarchical clustering. In *International Conference on Acoustics Speed and Signal Processing*, May 1998.

[40] M. Duran and I. Grossman. A mixed-integer nonlinear programming algorithm for process systems synthesis. *AIChE journal*, 32(4):592–606, 1986.

[41] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, July 1999.

[42] L. J. Eschelman and J. D. Schaffer. Real-coded genetic algorithms and interval schemata. In *Foundations of Genetic Algorithms 2*, pages 187–202. Morgan Kaufmann, San Francisco, CA, 1993.

[43] R. M. Everson, J. E. Fieldsend, and S. Singh. Full elite sets for multi-objective optimisation. In *5th International Conference on Adaptive Computing in Design and Manufacture (ADCM 2002)*, Exeter, UK, Apr. 2002.

[44] D. Favrat and F. Staine. Exergy and environmental considerations in process integration. In *PI'99 International Conference on Process Integration*, Copenhagen, Mar. 1999.

[45] D. B. Fogel. An overview of evolutionary programming. In Davis et al. [27], pages 89–109.

[46] D. B. Fogel. *Evolutionary Computation*. IEEE Press, New York, second edition, 2000.

[47] D. B. Fogel and A. Ghoziel. A note on representations and variation operators. *IEEE Transactions on Evolutionary Computation*, 1(2):159–161, 1997.

[48] L. J. Fogel. Autonomous automata. *Industrial Research*, 4:14–19, 1962.

[49] L. J. Fogel. *Intelligence Through Simulated Evolution*. John Wiley & Sons, New York, 1999.

[50] C. M. Fonseca and P. J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In S. Forrest, editor, *Genetic Algorithms: Proceedings of the Fifth International Conference*, pages 416–423, San Mateo, CA, USA, July 1993. Morgan Kaufmann. URL `citeseer.nj.nec.com/fonseca93genetic.html`.

[51] C. M. Fonseca and P. J. Fleming. An overview of evolutionary algorithms in multiobjective optimisation. *Evolutionary Computation*, 3(1):1–16, 1995. URL `http://www.lania.mx/~ccoello/EMOO/fonseca94.ps.gz`.

[52] A. Fraser. Simulation of genetic systems by automatic digital computers. *Australian Journal of Biological Sciences*, 10:484–491, 1957.

[53] B. Galvan, G. Winter, P. Cuesta, and S. Alonso-Lorenzo. Flexible evolution. In Giannakoglou et al. [56].

[54] General Motors. EV1 electric vehicle website. http://gmev.com, 2001.

[55] S. Germano, M. Nicollerat, and D. Favrat. A new pulsed multipoint injection system to reduce natural gas vehicle emissions. In *Natural Gas Vehicles*, Köln, Germany, May 1998.

[56] K. Giannakoglou, D. Tsahalis, J. Periaux, K. Papailou, and T. Fogarty, editors. *EUROGEN—Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*, Athens, Sept. 2001.

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

[58] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Genetic Algorithms and Their Applications: Proc. of the 2nd international Conference on GAs*, pages 41–49, Cambridge MA, July 1987. Lawrence Erlbaum.

[59] V. S. Gordon and D. Whitley. Serial and parallel genetic algorithms as function optimizers. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 177–183, San Mateo, CA, 1993. Morgan Kaufmann. URL `citeseer.nj.nec.com/gordon93serial.html`.

[60] T. Grüninger and D. Wallace. Multimodal optimisation using genetic algorithms. Technical Report 96.02, MIT CADlab, Boston, MA, 1996.

[61] G. R. Harik. Finding multimodal solutions using restricted tournament selection. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 24–31. Morgan Kaufmann, 1995. URL `citeseer.nj.nec.com/harik95finding.html`.

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

[63] J. Horn. Multicriteria decision making and evolutionary computation. In T. Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Institute of Physics Publishing, London, 1997.

[64] J. Horn, N. Nafpliotis, and D. E. Goldberg. A Niched Pareto Genetic Algorithm for Multiobjective Optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, volume 1, pages 82–87, Piscataway, New Jersey, 1994. IEEE Service Center. URL `citeseer.nj.nec.com/horn94niched.html`.

[65] H. Ishitani, Y. Baba, A. Molyneaux, and D. Favrat. An optimization tool of hybrid type EV systems. In *The 15th International Electric Vehicle Symposium and Exhibition (EVS-15)*, Brussels, Sept. 1998.

[66] B. Kalitventzeff and F. Marechal. Optimal insertion of energy saving technologies in industrial processes : a web based tool helps in developments and coordination of a european r&d project. *Applied Thermal Engineering*, 20(15–16):1347–1364, 2000.

[67] A. Kanacharos, D. Koulocheris, and H. Vrazopoulos. Acceleration of evolution strategy methods using deterministic algorithms. In Giannakoglou et al. [56].

[68] J. B. Kim and D. Wallace. A goal-oriented design evaluation model. In *Proceedings of the ASME Design Engineering Technical Conference*, Sacramento, California, Sept. 1997.

[69] J. Knowles and D. Corne. The Pareto archived evolution strategy: A new baseline algorithm for Pareto multiobjective optimisation. In *1999 Congress on Evolutionary Computation*, pages 98–105, Piscataway, NJ, 1999. IEEE Service Center. URL `citeseer.nj.nec.com/knowles99pareto.html`.

[70] J. D. Knowles and D. W. Corne. Local Search, Multiobjective Optimization and the Pareto Archived Evolution Strategy. In B. et al. McKay, editor, *Proceedings of Third Australia-Japan Joint Workshop on Intelligent and Evolutionary Systems*, pages 209–216, Ashikaga, Japan, 1999. Ashikaga Institute of Technology. URL `citeseer.nj.nec.com/knowles99local.html`.

[71] S. B. Kraines, T. Akatsuka, L. W. Crissman, K. R. Polenske, and H. Komiyama. Modeling pollution and cost of the cokemaking supply chain in Shanxi Province, China. *in preparation for submission to Journal of Industrial Ecology*, 2001.

[72] P. Krummenacher. personal communication, 2002.

[73] P. Krummenacher. *Contribution to the Heat Integration of Batch Processes (with or without Heat Storage)*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2002.

[74] F. Kursawe. A variant of evolution strategies for vector optimisation. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, pages 193–197, Berlin, 1991. Springer Verlag.

[75] M. Laumanns. personal communication, 2001.

[76] D. Levine, P. Hallstrom, D. Noelle, and B. Walenz. Experiences with the PGAPack parallel genetic algorithm library. In Davis et al. [27], pages 139–149.

[77] G. Leyland, S. Kraines, T. Akatsuka, A. K. Molyneaux, D. Favrat, and H. Komiyama. Minimising transport and investment costs in the coke industry in Shanxi province, China. in preparation for *Journal of Computers, Environment and Urban Systems*, 2002.

[78] G. Leyland, A. K. Molyneaux, and D. Favrat. A fast multi-objective evolutionary algorithm applied to industrial problems. In Giannakoglou et al. [56].

[79] G. Leyland, A. K. Molyneaux, and D. Favrat. A new multi-objective optimisation technique applied to a vehicle drive train simulation. In *ECOS—Efficiency, Costs, Optimization, Simulations and Environmental Impact of Energy Systems*, Istanbul, Turkey, July 2001.

[80] H. Li, M. Bürer, S. Z. Ping, and D. Favrat. Green energy system of heating: Potential evaluation and multi-criteria optimization of integrated energy system for more sustainable urban area. In *ECOS—Efficiency, Costs, Optimization, Simulations and Environmental Impact of Energy Systems*, Berlin, Germany, July 2002.

[81] X. Lin and K. R. Polenske. Input-output anatomy of China's energy-demand change, 1981-1987. *Economic Systems Research*, 7(1):67–84, 1995.

[82] B. Linnhoff, D. Townsend, D. Boland, G. Hewitt, B. Thomas, A. Guy, and M. R.H. *A User Guide on Process Integration for the Efficient Use of Energy*. Institution of Chemicals Engineers, England, 1982.

[83] S. W. Mafhoud. *Niching Methods for Genetic Algorithms*. PhD thesis, University of Illinois, Urbana, IL, 1995.

[84] F. Marechal and B. Kalitventzeff. A methodology for the optimal insertion of organic rankine cycles in the industrial processes. In *2nd International Symposium on Process Integration*, Halifax, Oct. 2001. URL `http://www.chemeng.ca/halifax2001/Program/00000087.htm`.

[85] F. Marechal and B. Kalitventzeff. A tool for optimal synthesis of industrial refrigeration systems. In *Computer-aided chemical engineering: Proceedings of ESCAPE-11*, volume 9, pages 457–463, 2001.

[86] J. Miller, T. P., and F. T. Designing electronic circuits using evolutionary algorithms: A case study. In D. Quagliarella, J. Périaux, C. Polini, and G. Winter, editors, *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, pages 105–131. Wiley, Chichester, 1998.

[87] A. K. Molyneaux. personal communication, 2001.

[88] A. K. Molyneaux. A simple parallel MATLAB implementation using PVM. Technical Report 0101i, École Polytechnique Fédérale de Lausanne, Jan. 2001.

[89] A. K. Molyneaux. *A Methodology for Optimising Complex Systems with Time Dependent Demand Variations Applied to Real World Test Cases*. PhD thesis, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 2002.

[90] A. K. Molyneaux, G. Leyland, and D. Favrat. A new, clustering evolutionary multi-objective optimisation technique. In *International Symposium on Adaptative Systems*, pages 41–47, Havana, Cuba, Mar. 2001.

[91] H. Muhlenbein. personal communication, 2001.

[92] H. Muhlenbein and T. Mahnig. Convergence theory and applications of the factorized distribution algorithm. *JCIT: Journal of Computing and Information Technology*, 7, 1999. URL `citeseer.nj.nec.com/muhlenbein99convergence.html`.

[93] H. Muhlenbein and T. Mahnig. Evolutionary algorithms: From recombination to search distributions. In *Theoretical Aspects of Evolutionary Computing*, pages 137–176. Springer, 2000.

[94] J. Ocanasek and J. Schwarz. The distributed bayesian optimisation algorithm for combinatorial optimisation. In Giannakoglou et al. [56].

[95] B. Olsommer. *Méthode d'Optimisation Thermoéconomique Appliqué aux Centrales d'Incinération d'Ordures à Cogénération avec Appoint Énérgétique*. PhD thesis, École Polytechnique Fédérale de Lausanne, 1998.

[96] B. Olsommer, D. Favrat, and D. Comte. Time-dependent thermoeconomic optimization of the future waste incineration power plant in Posieux, Switzerland. In *5th World Congress on Integrated Resources Management*, Toronto, Canada, June 2000.

[97] B. Olsommer, D. Favrat, and M. von Spakovsky. An approach for the time-dependent thermoeconomic modeling and optimization of energy system synthesis, design and operation (Part I: Methodology and results). *International Journal of Applied Thermodynamics*, 2(3):97–114, Sept. 1999.

[98] B. Olsommer, D. Favrat, and M. von Spakovsky. An approach for the time-dependent thermoeconomic modeling and optimization of energy system synthesis, design and operation (Part II : Reliability and availability). *International Journal of Applied Thermodynamics*, 2(4):177–186, Dec. 1999.

[99] V. Pareto. *Cours D'Economie Politique*, volume I and II. F. Rouge, Lausanne, 1896.

[100] X. Pelet, A. Repetti, R. Prélaz-Droux, M. Piguet, C. Pécoud, A. Musy, M. Fadhli, D. Cretegny, Y. Allani, and D. Favrat. POLEDURME, pôles énergétiques intégrés pour un développement durable en régions méditerranéennes. Rapport final, École Polytechnique Fédérale de Lausanne, Sept. 2000.

[101] S. Pelster. *Environomic Modelling and Optimization of Advanced Combined Cycle Cogeneration Plants Including* CO_2 *Separation Options*. PhD thesis, École Polytechnique Fédérale de Lausanne, 1998.

[102] S. Pelster, D. Favrat, and M. von Spakovsky. Optimisation thermo-économique et environomique de nouvelles centrales de production d'électricité à cycles combinés. *GWA Gas Wasser und Abwasser*, Aug. 2000.

[103] M. Peschel and C. Reidel. Use of vector optimization in multiobjective decision making. In D. Bell, R. Keeney, and H. Raiffa, editors, *Conflicting Objectives in Decisions*, pages 97–121, Chichester, 1977. Wiley.

[104] H. Pohlheim. GEATbx: Genetic and evolutionary algorithm toolbox for use with MATLAB. http://www.geatbx.com/docu/index.html, 1998.

[105] K. R. Polenske. personal communication, 2000.

[106] K. R. Polenske and F. C. McMichael. How to grow the chinese coking industry cleanly. *accepted for publication in Energy Policy*, 2001.

[107] D. Quagliarella and A. Vicini. Coupling genetic algorithms and gradient based optimization techniques. In D. Quagliarella, J. Périaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science–Recent advances and industrial applications*, pages 289–309. Wiley, Chichester, 1997.

[108] N. J. Radcliffe and P. D. Surry. Fundamental limitations on search algorithms: Evolutionary computing in perspective. *Lecture Notes in Computer Science*, 1000:275–??, 1995. URL `citeseer.nj.nec.com/radcliffe95fundamental.html`.

[109] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973.

[110] A. Rogers and A. Prügel-Bennett. Modelling the dynamics of a steady-state genetic algorithm. In W. Banzhaf and C. Reeves, editors, *Foundations of Genetic Algorithms 5*, pages 57–68. Morgan Kaufmann, San Francisco, CA, 1999. URL `citeseer.nj.nec.com/rogers99modelling.html`.

[111] S. Ronald. Robust encodings in genetic algorithms. In D. Dasgupta and Z. Michalewicz, editors, *Evolutionary Algorithms in Engineering Applications*, pages 29–44. Springer, Berlin, 1997.

[112] G. Rudolph. Evolutionary search for minimal elements in partially ordered finite sets. In V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, editors, *Evolutionary Programming VII*, pages 345–353, Berlin, 1998. Springer. URL `citeseer.nj.nec.com/rudolph98evolutionary.html`.

[113] C. Sarreau. Simulation thermoéconomique d'un site isolé avec plusiers appoints énergetiques en focalisant sur les aspects environomiques. Travaille de diplome, École Polytechnique Fédérale de Lausanne, June 2000.

[114] D. Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 93–100, 1985.

[115] N. N. Schraudolph and R. K. Belew. Dynamic parameter encoding for genetic algorithms. *Machine Learning*, 9:9–21, 1992. URL `citeseer.nj.nec.com/schraudolph92dynamic.html`.

[116] H.-P. Schwefel. Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik. Diplomarbeit, Technische Universität Berlin, 1965.

[117] H.-P. Schwefel. *Numerical Optimisation of Computer Models*. John Wiley, Chichester, UK, 1981.

[118] H.-P. Schwefel. *Evolution and Optimum Seeking*. John Wiley & Sons, New York, 1995.

[119] H.-P. Schwefel and T. Bäck. Artificial evolution: How and why? In D. Quagliarella, J. Périaux, C. Polini, and G. Winter, editors, *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, pages 1–19. Wiley, Chichester, 1998.

[120] N. Senin, D. Wallace, and N. Borland. Object-based design modeling and optimization with genetic algorithms. In *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, Orlando, FL, USA, July 1999.

[121] Shanxi committee of atlas compilation. *Energy resources atlas of Shanxi Province (in Chinese)*. Shanxi: Science Press, 1994.

[122] O. Sharpe. Beyond NFL: A few tentative steps. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 593–600, University of Wisconsin, Madison, Wisconsin, USA, 1998. Morgan Kaufmann. URL `citeseer.nj.nec.com/296977.html`.

[123] W. M. Spears. Simple subpopulation schemes. In *Proceedings of the 1994 Evolutionary Programming Conference*, pages 296–307. World Scientific, 1994.

[124] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1995. URL `citeseer.nj.nec.com/srinivas94multiobjective.html`.

[125] V. S. Sunderam. PVM: A framework for parallel distributed computing. *Concurrency: Practice and Experience*, 2(4):315–339, Dec. 1990.

[126] The Mathworks. *MATLAB Reference Manual*. The Mathworks, 1998.

[127] D. Thierens and P. Bosman. Multi-objective optimization with iterated density estimation evolutionary algorithms using mixture models. In *International Symposium on Adaptative Systems*, pages 129–135, Havana, Cuba, Mar. 2001.

[128] Toyota. Prius hybrid system. CD-ROM, Jan. 2001.

[129] Unpublished. Unpublished survey data, 2001.

[130] D. A. Van Veldhuizen and G. B. Lamont. Evolutionary computation and convergence to a Pareto front. In J. R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1998 Conference*, University of Wisconsin, Madison, Wisconsin, USA, 22-25 1998. Stanford University Bookstore. URL `citeseer.nj.nec.com/vanveldhuizen98evolutionary.html`.

[131] D. A. Van Veldhuizen and G. B. Lamont. Multiobjective Evolutionary Algorithm Test Suites. In J. Carroll, H. Haddad, D. Oppenheim, B. Bryant, and G. B. Lamont, editors, *Proceedings of the 1999 ACM Symposium on Applied Computing*, pages 351–357, San Antonio, Texas, 1999. ACM. URL `citeseer.nj.nec.com/david99multiobjective.html`.

[132] M. D. Vose. What are genetic algorithms? A mathematical perspective. In Davis et al. [27], pages 251–293.

[133] D. Wallace, D. Favrat, T. Tomiyama, and H. Ishitani. A framework for holistic life-cycle design: the integration of performance, economic, manufacturing and environmental measures. Technical report, Massachusetts Institute of Technology, 1999.

[134] K. Wipke, M. Cuddy, and S. Burch. ADVISOR 2.1: A user-friendly advanced powertrain simulation using a combined backward/forward approach. *IEEE Transactions on Vehicular Technology*, 1999.

[135] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997. URL `citeseer.nj.nec.com/wolpert96no.html`.

[136] J. Yamaguchi. Global viewpoints: More on insight. *Automotive Engineering International*, Jan. 2000.

[137] E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimisation: Methods and Applications*. PhD thesis, Eidgenöissiche Technische Hochschule Zürich, Nov. 1999.

[138] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. TIK Report 103, Eidgenöissiche Technische Hochschule Zürich, May 2001.

[139] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, Nov. 1999.

Curriculum Vitae

Geoff Leyland
born on the 4^{th} of March 1972, in Auckland, New Zealand.
Citizen of New Zealand

Education

1997	Master of Engineering in Mechanical Engineering from Auckland University, Auckland, New Zealand.
1992	Bachelor of Engineering in Engineering Science with First Class Honours from Auckland University, Auckland, New Zealand.
1989	University Scholarship and University Bursary from Avondale College and Green Bay High School, Auckland, New Zealand.

Professional Experience

1997-2002	Research assistant at the Laboratoire d'Energétique Industrielle (LENI), EPFL. Instrumenting an co-generation natural gas engine test stand, modelling hybrid vehicle performance and development of high-performance multi-objective evolutionary algorithms.
1994	Engineer at Mott Ewbank Preece, Brighton, England. Modelling economics of proposed power plants. Preparation of operating procedure manuals for a desalination plant in Saudi Arabia.
1993	Project engineer at Toussaint and Richardson, Perth, West Australia. Projects in the alumina industry in West Australia. Modelling pressure drop in fluid-bed calciners.