Multi-Objective Evolutionary Optimization in Time-Changing Environments

by

Iason Hatzakis

Master of Science in Ocean Engineering
Massachusetts Institute of Technology, 2004

Diploma in Naval Architecture and Marine Engineering
National Technical University of Athens, 2000

SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY IN MECHANICAL ENGINEERING

AT THE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUNE 2007

Author:....……………………………………………………………………………………
Department of Mechanical Engineering
June 6, 2007

Certified by:…………………………………………………………………………......
David R. Wallace
Professor of Mechanical Engineering
Thesis Supervisor

Accepted by:…………………………………………………………….……................
Lallit Anand
Professor of Mechanical Engineering
Chairman, Departmental Committee on Graduate Studies

Multi-Objective Evolutionary Optimization in Time-Changing Environments

by

Iason Hatzakis

Submitted to the Department of Mechanical Engineering on June 6, 2007 in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Mechanical Engineering

Abstract

This research is focused on the creation of evolutionary optimization techniques for the solution of time-changing multi-objective problems. Many optimization problems, ranging from the design of controllers for time-variant systems to the optimal asset allocation in financial portfolios, need to satisfy multiple conflicting objectives that change in time. Since most practical problems involve costly numerical simulations, the goal was to create algorithmic architectures that increase computational efficiency while being robust and widely applicable.

A combination of two elements lies at the core of the proposed algorithm. First, there is an anticipatory population that helps the algorithm discover the new optimum when the objective landscape moves in time. Second, a preservation of the balance between convergence and diversity in the population which provides an exploration ability to the algorithm. If there is an amount of predictability in the landscape's temporal change pattern the anticipatory population increases performance by discovering each timestep's optimal solution using fewer function evaluations. It does so by estimating the optimal solution's motion with a forecasting model and then placing anticipatory individuals at the estimated future location. In parallel, the preservation of diversity ensures that the optimum will be discovered even if the objective's motion is unpredictable. Together these two elements aim to create a well-performing and robust algorithmic architecture. Experiments show that the overall concept functions well and that the anticipatory population increases algorithm performance by up to 30%.

Constraint handling methods for evolutionary algorithms are also implemented, since most of the problems treated in this work are constrained. In its final form the constraint handling method applied is a hybrid variant of the Superiority of Feasible Points, which works in a staged manner.

Three different real-world applications are explored. Initially a radar telescope array is optimized for cost and performance as a practical example of a static multi-objective constrained problem. Subsequently, two time-changing problems are studied: the design of an industrial controller and the optimal asset allocation for a financial portfolio. These problems serve as examples of applications for time-changing multi-objective evolutionary algorithms and inspire the improvement of the methods proposed in this work.

Thesis Supervisor: David Wallace
Title: Professor of Mechanical Engineering
Committee Members: Olivier de Weck, Professor of Aeronautics and Astronautics & Engineering Systems
Daniel Frey, Professor of Mechanical Engineering

Acknowledgements

I owe a lot to David Wallace. For his help, his continuous presence in CADLab, and for the restless source of inspiration he has been to me over my years in MIT, from the classroom and the workshops of 2.744 to the conclusion of this work.

Geoff Leyland deserves many thanks. First, for creating QMOO. Second, for spending so much time and patience with me while I was learning how to work with it, and practically manning an evolutionary algorithm hotline from sixteen time zones away.

Many thanks are due to Olivier de Weck and Daniel Frey, for devoting the time to be on my doctoral committee and providing their insightful feedback. Also, Prof. de Weck's class on Multidisciplinary Design Optimization, which he taught together with Karen Willcox, was another source of inspiration for this work.

A large number of people have helped me over the time, and without them this work would not have been possible. Frank Schlottmann deserves many thanks, his work having inspired the portfolio optimization problem and himself having collaborated with me on it. Gergana Bounova deserves many thanks for her help with the telescope array optimization problem. Ann Annaswamy, Paul Barton, David Ceglio, Carlos Coello Coello, Lawrence 'David' Davis, Kalyanmoy Deb, Marco Farina, Sevan Ficici, Kyriakos Giannakoglou, Kenneth de Jong, Chris Magee, Kamal Malek, Alexandros Mitsos, Paul Sclavounos, George Tzabiras, Michael Yukish and several others are owed many thanks for the helpful and inspiring conversations we had in conferences, meetings and through correspondence.

I owe a lot to Elina, for standing next to me from so far, and for being my muse.

Although Cambridge became my home, it is very far from my homeland. I am deeply grateful to my mother and to the rest of my family and friends in Greece who made me feel I was closer than I actually was.

And, I would not have made it through without my friends here. Konstantinos, Emanuela, Yiannis, Johnna, Dooyon, Justin, Dee, Panayiotis, Ansgar, Anna, Elena, Ali, Tufool, Alex, Steve, Jay, and many others; the CADLab crew (Sittha, Monica, Jessica, Alan, Sangmok, Wei, Mika, Tom, Barry, James, Jacob, Amy, and many others); my friends from the Charles river dock of MIT Sailing (Franny, Sean, Alvar, Dwight, and many others); my sailboat racing friends from the starting lines of Marblehead (John, Lisa, Kay, Julie, Dan, Russell, Justin, and many others). Thank you.

Contents

List of Figures

List of Tables

Abbreviations and Symbols

Abbreviations

AR	Autoregressive Model
AP	Anticipatory Population
CTI	Closest-to-Ideal
D-QMOO	Dynamic Queuing Multi-Objective Optimizer
DOME	Distributed Object-based Modeling Environment
EA	Evolutionary Algorithm
EOC	Evolutionary Operator Choice
EP	Evolutionary Programming
ES	Evolution Strategy
FPS	Feed-forward Prediction Strategy
GA	Genetic Algorithm
LH	Latin Hypercube
NDS	Non-Dominated Set
PM	Penalty Methods
POF	Pareto Optimal Front
POS	Pareto Optimal Set
QMOO	Queuing Multi-Objective Optimizer
SFP	Superiority of Feasible Points
SFP-IFE	Superiority of Feasible Points with Initial Feasibility Exploration
VAR	Vector Autoregressive Model
VaR	Value-at-Risk
WSFP	Weighted Superiority of Feasible Points

Symbols

$\mathbf{x}$	design vector

n	design vector dimension
$\mathbf{f}$	objective vector
m	objective vector dimension
g	inequality constraint function
q	number of inequality constraints
h	equality constraint function
l	total number of constraints
l_i	lower bound for design variable i
u_i	upper bound for design variable i
t	time
$\mathbf{x}^*{}_t$	optimal solution at time t
$\tilde{\mathbf{x}}_t^*$	forecast for the optimal solution at time t
$\boldsymbol{\varepsilon}_t$	forecast error at time t
X	search space
r	penalty weight
f_j	constraint violation function for constraint j
θ_k	superiority function for objective k
$eval_k$	penalized objective function for objective k
a	univariate autoregressive coefficient
$\mathbf{A}$	multivariate autoregressive coefficient matrix
w	univariate autoregressive model intercept term
$\mathbf{w}$	multivariate autoregressive model intercept term

1 Introduction

This work is focused on the creation of evolutionary optimization techniques for the solution of multi-objective problems that change in time.

Our world is in a constant state of change, both in nature and society. Natural environment changes, from daily temperature fluctuations to long-term climate variation, requiring various species to change correspondingly in order to survive. Consumer expectations and manufacturing techniques change, requiring product design to be altered in order to provide adequate, low-cost solutions. Market conditions change, requiring investments to change accordingly in order to provide the desired return and protection against risk.

In parallel, when we make decisions we usually find ourselves having to balance a set of conflicting trade-offs. A common set of trade-offs occurs in engineering design and product development where the decision maker needs to position a product in terms of performance and cost, which are usually conflicting. The notion of conflicting objectives is encountered very often, for example in everyday life where one needs to compromise between the amount of rent they are willing to pay and the location they want to live in, or in corporate management where a decision for the allocation of resources among various business units has to be made.

The goal of this work is to propose and examine techniques for the discovery of efficient solutions for problems that change in time and need to satisfy conflicting trade-offs.

1.1 Overview and contribution

A brief overview of this thesis noting its contributions is given here.

In chapter 2 a discussion on evolutionary computation as an optimization tool is provided. The general application area of integrated product development environments is used as a ground for the employment of evolutionary algorithms, helping identify some of their most important and desirable attributes such as robustness and applicability. The Queuing Multi-Objective Optimizer (QMOO) is also described in this chapter. QMOO provided the base algorithm which is developed into the Dynamic-QMOO (D-QMOO) algorithm during the course of this work.

In chapter 3, the solution of constrained optimization problems with evolutionary algorithms is treated. Constraint handling methods, hybrid variants of the method of Superiority of Feasible Points, are created for D-QMOO. Enabling D-QMOO to handle constrained problems is the first contribution of this work, and an important step since most of the benchmark and real-world problems solved in this thesis are subject to constraints.

Chapters 4 and 5 contain the core contribution of this work. The proposed algorithmic architecture for the solution of time-changing multi-objective problems is presented there. This architecture is based on the presence of two elements. On one hand, an anticipatory population which helps the algorithm discover the new optimum when the objective changes in time. As a result the algorithm uses fewer function evaluations and its performance is increased. On the other hand, a balance between population convergence and diversity in the design space which ensures the retention of an exploratory group of individuals. This balance assists in the discovery of the new optimum, even if the objective moves in an unpredictable way and an anticipatory population cannot be created successfully. Together, these two elements aim to create an algorithmic architecture that is both well-performing and robust.

In chapter 4 the basic time-changing optimization concept is described and used on test problems, where it becomes apparent how the use of the anticipatory population increases performance. In chapter 5 the topology of the anticipatory population is further explored. Various techniques are proposed for the coverage of the non-dominated set by the anticipatory individuals, and for dealing with the cases in which there is a large amount of forecast error or the objective's motion is unpredictable.

In chapter 6, two practical applications are treated: an industrial controller design and a financial portfolio optimization problem. These applications serve as examples of areas where multi-objective time-changing evolutionary optimization can be applied, and also inspire some additional development of the techniques proposed in the previous two chapters.

1.2 Earlier work on which this thesis is based

This thesis is based on and inspired by a large amount of prior work, for which the author is grateful. In general, prior work is pointed out in the form of bibliographical references throughout the text. However, there are two main legs of earlier work on which this thesis is based:

From the algorithmic standpoint, the author built on Geoff Leyland's work, using the QMOO algorithm as a base code for which constraint-handling and time-changing capabilities were developed.

From the theoretical standpoint, this thesis is based on a large amount of past work on dynamic optimization and constraint handling with evolutionary algorithms. The overall number of authors is large, but especially the work of Jürgen Branke, Claus Wilke and Marco Farina provided a lot of inspiration in the area of time-changing evolutionary optimization, the work of Carlos Coello Coello in the area of constraint handling and the work of Kalyanmoy Deb in the area of multi-objective optimization.

2 Evolutionary Optimization

2.1 Evolutionary algorithms in the present day

A population of a biological species is usually composed of individuals that are different to each other. Each individual is characterized by its fitness that allows them to adapt to the environment and survive. Different individuals might have similar or different levels of fitness. As the species evolves, some individuals thrive and procreate while others do not survive. The level of fitness as a function of the individuals' characteristics can be visualized as a fitness landscape, and the evolution as an exploration process that seeks the peaks of this landscape.

Evolutionary computation has been defined as *the use of evolutionary systems as computational processes for solving complex problems* (De Jong 2006). Picturing evolution as the exploration of a fitness landscape leads to the idea of an evolutionary system as an optimization process, with the peaks of high fitness that the population seeks being the optimal solutions.

This idea historically[1] led to the development of a number of evolutionary optimization methods. Initially these methods were classified into distinct groups, the three most prominent ones being *genetic algorithms (GA)*, *evolution strategies (ES)* and *evolutionary programming (EP)*. During the past fifteen years however, there has been an increasing interest in evolutionary optimization both from the algorithmic development and from the practical application aspect. This interest has led to the emergence of a host of new algorithms and methods, several of which cannot be strictly classified into one of these groups. The algorithm used and developed in this work, QMOO, is such an example. Interaction and cross-breeding of ideas among the various communities such as GA, ES and EP (De Jong, Spears 1993) has gradually led to a more abstract and fundamental view of these methods, under the name *evolutionary algorithms*.

Evolutionary algorithms are heuristic optimization tools which emulate the natural evolution and adaptation process. Their main principle of operation is to work with a set of solutions (designs) to the optimization problem. Each design is called an *individual* and the set is called the *population*. In some cases such as a (1+1) evolution strategy, the population can be composed of a singe design. During each iteration of the algorithm, new solutions are derived from the existing population by applying a set of operators. These new individuals are often called *children*, in liberal reference to the offspring of a generation in a biological system. The operators applied to derive the children have appeared in various forms among the different algorithms, but

[1] Starting mainly during the 1960s.

in general they are either of the *crossover* or the *mutation* type. A crossover operator combines the design vectors of two or more members of the population in order to produce a new design, while a mutation operator alters part of the design vector of an individual in a random way. Each individual is characterized by its *fitness* measure which is derived from the objective function(s) of the optimization problem. The framework is completed by the existence of a set of *selection criteria* which help choose individuals from the population in order to perform crossover or in order to eliminate them. These criteria simulate the survival of the fittest process that we encounter in nature.

Convergence to the optimal solution is effected through some mechanism of preference for better designs over worse ones. This mechanism usually derives from the selection criteria. For example this preference may be expressed through the way parent individuals are selected, through the way individuals to be eliminated are selected, or through the way the best-so-far individuals are preserved. Algorithms that never allow an inferior individual to replace a fitter one show the strongest form of preference and are called *elitist*.

At the same time however, most EAs have a way of preserving *diversity* in their population of designs during at least part of the optimization process. Roughly speaking, a diverse population of solutions is one that is spread over the design space. Diversity preservation is basically accomplished through the partial or full stochasticity of the crossover and mutation operators, through the existence of separate, independent groups in the population, and through the fact that in an EA a population of individuals rather than a single design is being processed at any moment. Some algorithms use additional explicit techniques for the preservation of diversity.

Some of the most attractive characteristics of EAs as compared to other optimization methods are owed to their stochastic and population-based nature and to the diversity preservation they encourage. EAs are global optimization tools in the sense that they explore the whole design space and have the potential to discover a global optimum instead of being trapped in local optima – even if one part of the population is converging to a local optimum, individuals might be exploring other areas of the design space. This population-based nature also provides the decision maker with a more comprehensive overview of the design space and allows the discovery of several local optima along with the problem's global solution, which might prove to be useful information. Another vital attribute of EAs is that they are not restricted in terms of the nature of the objective function – for example it may be discontinuous, multimodal or non-differentiable (Branke 2002) in contrast to other approaches such as gradient-based methods. Essentially all that is required is a 'black box' process which returns some quantitative measure of merit given a design vector (Reeves, Rowe 2003).

These attributes make EAs able to handle a wide variety of optimization problems – among several applications, EAs have been used in large-scale industrial energy problems (Leyland 2002), water distribution and resource management (Deb 2004), ship design (Day, Doctors 1997) and aerodynamic design (Obayashi, Sasaki 2004). Another indicator of the level of acceptance of EAs as indispensable optimization algorithms is their increasing inclusion in industrial optimization packages such as iSight (Tong, Powell 2003) and MATLAB (Chipperfield, Fleming 1995).

In parallel EAs suffer from a range of disadvantages. One of their most prominent drawbacks is their computational expense. Indeed, many optimization problems can be solved in a fraction of the processing time using other problem-specific optimization methods – an example are linear problems. EAs also often lack a rigorous proof of convergence to the global optimum in finite time, in contrast with methods such as branch-and-bound[2].

Evolutionary algorithms as multi-objective problem solvers

As the design process becomes holistic and multi-disciplinary, the optimization problems that need to be solved are characterized by an increasing number of conflicting criteria leading to necessary trade-offs. Indeed, even a relatively contained problem such as designing the driver's seat of a sedan can end up having as many as 20 to 30 different objectives.

EAs are among the few algorithms that naturally lend themselves to the solution of multi-objective problems. Given their population-based structure, they do not require any kind of approximation or aggregation of multiple objectives into a single metric; on the contrary they accept the notion of the best solution being a set of several designs. Indeed, EAs have the ability to process several solutions in parallel, evolving a population towards the approximation of a Pareto optimal set in a single run (the concept of Pareto optimality for multi-objective problems will be discussed in section 2.3). The unique advantages of EAs as multi-objective problem solvers have been discussed by several authors, and a number of well-performing multi-objective EAs have been developed during the 1990s and 2000s. A concise overview of evolutionary techniques for multi-objective optimization can be found in the books by Kalyanmoy Deb (Deb 2001) and Carlos Coello Coello and Gary Lamont (Coello Coello, Lamont 2004).

2.2 Evolutionary algorithms as a search tool for distributed simulation and product development environments

Distributed simulation and product development environments are frameworks which address today's complex and multidisciplinary product design process. These environments aim to make the design process more manageable in the face of big design teams and large amounts of data that need to be processed by the designers and decision makers. Distributed design frameworks perform two core functions:

- They provide a means of publishing design and simulation services on a publicly available network such as the world wide web, and a means of subscribing to and using these services.
- They provide a means of integration, where a designer can use a set of distributed design and simulation services from a host of different disciplines in order to synthesize a product, in an easy and flexible way and without requiring discipline-specific knowledge.

Such environments are a relatively new tool in product development, with an active research and commercial activity during the past decade. Examples are the Distributed Object-based Modeling Environment (DOME – Senin, Wallace & Borland 2003), Engineous FIPER (Bailey, VerDuin 2000) and Phoenix ModelCenter (Woyak, Malone 1999).

A core function in these environments is the selection of parameter values for design modules and systems. Numerical tools in the form of search and optimization algorithms are valuable by performing this function and helping designers select the most desirable configurations among a plethora of alternatives.

As distributed simulation and design frameworks evolve and become part of industry practice, the design process becomes increasingly modular and multidisciplinary. Hence the optimization problems addressed can be very diverse; they can vary in terms of nature, type, and difficulty. A

[2] However, researchers such as Suzuki (Suzuki 1995) and Rudolph (see for example Rudolph 1998) study the convergence properties of EAs and in some cases provide finite-time convergence criteria applicable to a range of algorithms.

need exists for optimization tools that aid in the selection of dominant solutions in such problems. This work addresses the need for such tools.

We can define problem diversity (as it relates to optimization) in two dimensions:

- *type*, and
- *difficulty* of the optimization problem

By *type* we mean the category that the problem in hand falls into. Type relates to differentiation between static and time-changing, constrained and unconstrained, single- and multi-objective problems. By *difficulty* we denote an abstract attribute which expresses the probability that a given optimization algorithm will be able to solve the problem. Linearity or non-linearity, convexity, continuity, differentiability, and size and shape of the feasible region relate to the problem's difficulty. A non-convex function whose global minimum can be hard to discover is shown in the sketch of Figure 1.

Given the positive attributes of EAs as discussed in section 2.1, it becomes apparent that they are an apt choice as optimization tools for distributed simulation and design environments. These environments demand robust optimization tools with wide applicability. Methods with larger specificity such as Quadratic Programming would be more efficient for the solution of some problems, but are also bound to face problems to which they are not applicable. EAs on the other hand exhibit the robustness and wide scope required by distributed environments.

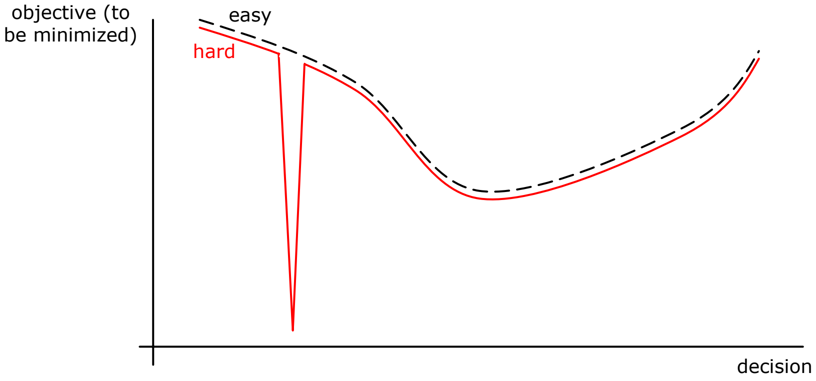


Figure 1. Problem difficulty (Leyland 2002)**.**

2.3 General problem statement

The mathematical statement of an optimization problem which may be time-changing, multi-objective and constrained is provided in equation (2.1):

$$\begin{aligned}
&\text{Find } \mathbf{x}^*_t \in \mathrm{X} \subseteq R^n \quad \text{which minimizes} && \mathbf{f}(\mathbf{x},t) = \left[f_1(\mathbf{x},t),\dots,f_m(\mathbf{x},t) \right] \\
&\text{subject to} && l_i \le x_i \le u_i, \quad i=1,\dots,n \\
& && g_j(\mathbf{x},t) \le 0, \quad j=1,\dots,q \\
& && h_j(\mathbf{x},t) = 0, \quad j=q+1,\dots,l
\end{aligned} \qquad (2.1)$$

where **x** is an n-dimensional design vector defined on a search space X and **f** is an m-dimensional objective function. The g and h functions express a total of l inequality and equality constraints. Parameter t represents a temporal dimension that advances in a continuous or discrete manner – it may represent actual time or simply different stages of a problem. In the discrete case, time advances through a series of timesteps $\{\ldots, t\text{-}2, t\text{-}1, t, t+1, \ldots\}$.

Definition (2.1) encompasses a wide variety of problems. If for example the time variable is held constant, it is a static optimization problem, and if the l parameter is zero it is an unconstrained problem. The problems explored in this work are instances of definition (2.1).

Pareto optimality and dominance

In multi-objective problems, the concept of *Pareto optimality* is used in order to define the optimal solution:

A solution $\mathbf{x} \in$ X is Pareto optimal in X if there is no other solution $\mathbf{y} \in$ X such that:

$$f_k(\mathbf{y}) \le f_k(\mathbf{x}) \text{ for every } k \in \{1,...,m\} \tag{2.2}$$
$$f_k(\mathbf{y}) < f_k(\mathbf{x}) \text{ for at least one } k \in \{1,...,m\}$$

The set of all Pareto optimal solutions in the search space is called *Pareto optimal set (POS)* (Coello Coello, Lamont 2004). Essentially, a solution belongs to the Pareto optimal set if there is no other solution which is not worse in all objectives and better in at least one. The Pareto optimal set maps onto the *Pareto optimal front (POF)* in the objective space.

The concept of *dominance* is used in order to compare solutions, since some individuals are better than others in the Pareto sense:

A solution $\mathbf{x} \in$ X dominates another solution $\mathbf{y} \in$ X if:

$$f_k(\mathbf{x}) \le f_k(\mathbf{y}) \text{ for every } k \in \{1,...,m\} \tag{2.3}$$
$$f_k(\mathbf{x}) < f_k(\mathbf{y}) \text{ for at least one } k \in \{1,...,m\}$$

If a solution is not dominated by any other solution in the population, it belongs to the *non-dominated set (NDS)*. The non-dominated set can be thought of as the algorithm's approximation of the Pareto optimal set (Leyland 2002). In the sketch of Figure 2, individual **a** dominates individuals **b**, **c** and **d**. Individual **a** itself belongs to the non-dominated set.

Certain members of the NDS provide the best values for each of the objectives. These individuals lie at the extremities of the NDS and are called *anchor points*. In Figure 2 the anchor points for the f_1 and f_2 objectives are shown.

Returning to the problem statement in (2.1), in the multi-objective case (m > 1) the optimal solution $\mathbf{x}^*_t$ at time t belongs to the Pareto optimal set.

Ranking

In order to assign a fitness value to an individual (recall the discussion on Evolutionary Algorithms in section 2.1), a ranking scheme can be used. In a single-objective problem individuals can be ranked according to their objective value – the best solution (with the lowest objective) has a rank of one, the second best has a rank of two and so on.

Several ranking schemes have been proposed for multi-objective problems (see for example Goldberg 1989, Fonseca, Fleming 1993, and Zitzler, Thiele 1999). A variant of the scheme proposed by Goldberg (Goldberg 1989) is used as one of QMOO's ranking schemes (described in section 2.4). According to this scheme, the non-dominated set of the population is found, and these individuals are given the best rank (one) and removed from the rank search. The non-dominated set of the remaining population is then found, the individuals are given a rank of two and removed from the search, and so on until a maximum number of ranks is found or all individuals have been ranked. This way a series of successive fronts is discovered. An algorithm similar to the non-dominated sorting proposed by Deb et al. (Deb et al. 2000) is used in order to perform this process quickly.

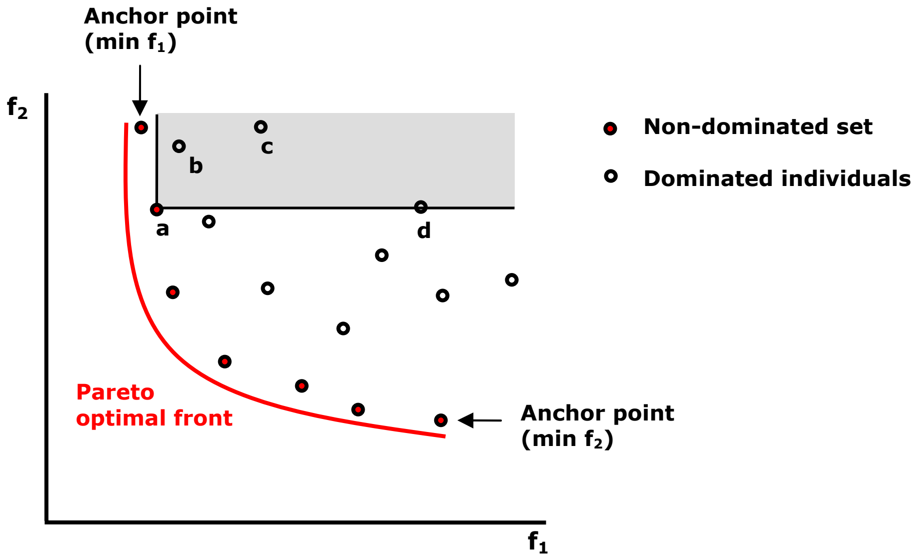


Figure 2. A two-objective problem. The non-dominated set is an approximation of the Pareto optimal front.

As a method of assigning fitness to individuals, ranking has a comparative nature. It does not take in account the absolute objective value of a solution, nor the distance between solutions in the objective space – it only expresses the relative merits of solutions to each other. Leyland (Leyland 2002) provides an interesting discussion on the nature of rank. Here we will only repeat that despite rank's comparative character, Evolutionary Algorithms are capable of optimizing using it as a fitness measure.

Problems that change in discrete time

Let us recall the problem definition in (2.1). In the discrete time-changing case, the objective function remains constant during each timestep and changes when the time parameter advances to its next value. In the simplified illustration of Figure 3, the sequence of three instances of the POS in a time-changing two-objective problem is shown. An optimization algorithm ideally discovers the new optimal solution or POS (in this case the POS is the black line segment connecting the two minima in each time instance) in the computational time available between each change in the objective landscape.

Given the computationally intensive simulations that characterize most practical problems, the main source of computational cost for an optimization process is the objective function evaluation. For this reason the objective change frequency (which translates to the amount of computational time available between changes in the objective landscape) is measured by the number of available objective function evaluations between timesteps. The importance of the objective change frequency will be discussed in more detail in section 4.1, but it can noted here that a fundamental measure of merit for an algorithm is the solution accuracy it achieves given a specific objective change frequency.

The problems treated in this work change in discrete time. In many cases problems that change in continuous time can be discretized to a desired level of accuracy and solved by a discrete-time algorithm such as the one developed here. However, there may be cases when a discretized continuous time problem changes so fast that there are not enough objective function evaluations available per timestep for a discrete-time algorithm to work as designed[3]. In these cases different algorithmic architectures are required[4].

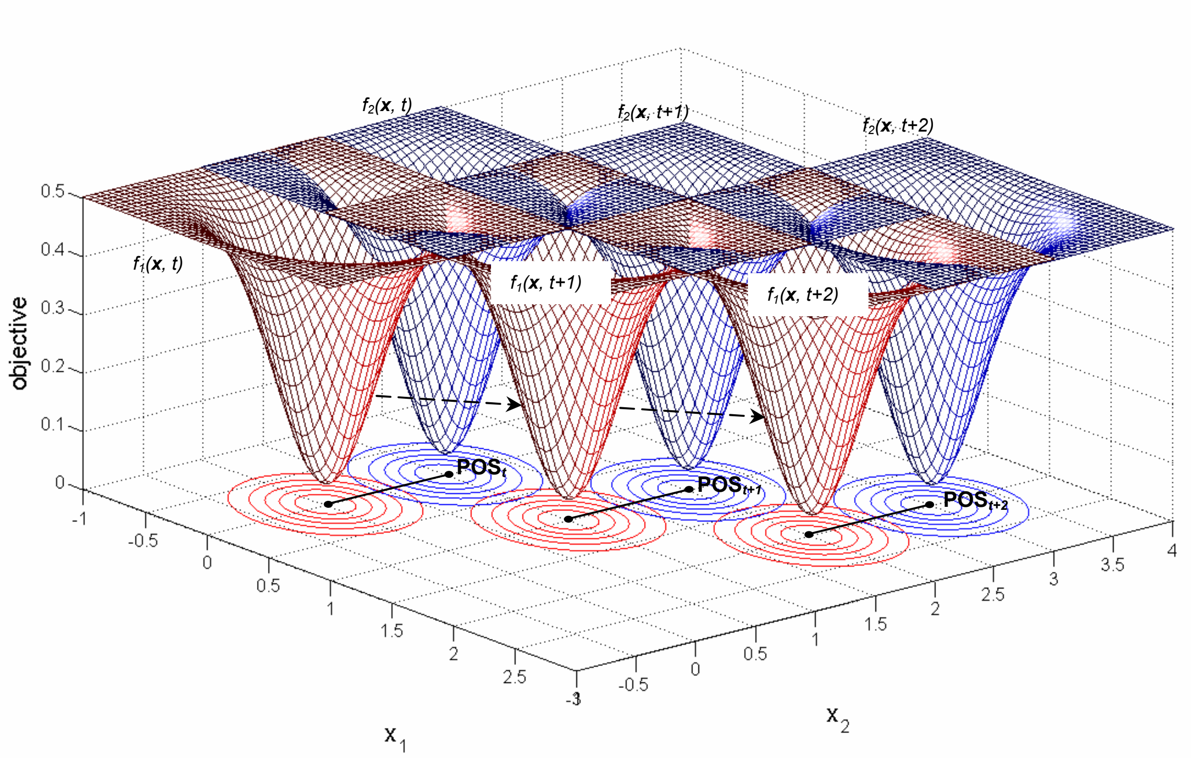


Figure 3. Time-changing two-objective problem. The minimization of f_1 (left, in red) and f_2 (right, in blue) is sought. This is a simplified sketch that is not representative of most benchmark or real-world problems. For example, the POS rarely retains the same form and travels along a straight line as it does here. However this sketch will become useful for the illustration of certain concepts later in this work.

[3] One such case can occur when a generational algorithm is used, and the objective change frequency is so high that the landscape moves one or more times during a single generation. Note, however, that the D-QMOO algorithm developed in this work is not generational and different criteria would be required in order to distinguish cases where the objective change frequency is too high.

[4] For example, Arnold and Beyer study the optimum tracking of linearly moving targets in continuous time with Evolution Strategies (Arnold, Beyer 2006), using a (μ/μ, λ) ES. In this context, 'continuous time' means that the objective is moving once with each generation of the ES, resulting in an objective change frequency of λ evaluations per timestep.

The main scope of this thesis is the development of algorithmic structures that can solve problems of such a discrete form and possess a wide scope of applicability. Emphasis is given to time changing optimization for multi-objective problems, which is an area that has seen relatively little development.

2.4 The Queuing Multi-Objective Optimizer

In this section a brief description of the Queuing Multi-Objective Optimizer (QMOO) is given. QMOO provided the base algorithm, which is further developed into a time changing and constraint handling form in this thesis.

QMOO was developed by Geoff Leyland (Leyland 2002) and other researchers at the Laboratory for Industrial Energy Systems (EPFL-LENI) in Lausanne, as an evolutionary algorithm initially focused on the solution of computationally intensive industrial energy problems. However QMOO has proven to be a robust and well-performing optimization algorithm for a wide range of applications, and as a result it has been implemented as a search tool in the Distributed Object-based Modeling Environment (Wronski 2005). QMOO's robustness and wide applicability has been further verified in the course of this work. Indeed, the author never encountered a problem (either practical or benchmark) that QMOO could not solve. In many cases QMOO discovered better-performing solutions than other algorithms, even when it was used to solve problems to which these algorithms were custom designed[5].

General

A key characteristic of QMOO is that it is a steady-state algorithm; it is based on a population of designs to which new solutions are incrementally added and existing solutions incrementally eliminated, while the population undergoes a continuous process of ranking and re-distribution. QMOO's core iteration is based on the concept of *queues*. A queue is a number of new solutions (children) created in each iteration, evaluated, and inserted into the population. In practice a queue is composed of around ten solutions. Each basic process (creation, assignment, evaluation, ranking) has its own queue. The algorithm's steady-state character makes it easier to implement the basic processes as queues. This leads to a flexible algorithmic architecture, and also offers the option to parallelize the solution process with several computers performing objective function evaluations for individuals taking as much time as they need, while a master computer holds the population and advances the solution (Leyland 2002).

If we attempt to place QMOO under one of the existing EA classes, we will see that is has several of the characteristics of an Evolution Strategy and a Genetic Algorithm. It is neither of the two though. An overview of how the algorithm works can be found in the flowchart of its basic iteration in Figure 4. Next, we list and discuss some core characteristics of QMOO's architecture.

Ranking and population

Two different ranking schemes are used, dictating how the population is divided with regard to the individuals' fitness (see Figure 5). According to the first scheme there exist two ranks: the first rank (rank one, to which all non-dominated individuals belong) and the worst rank (to which all other individuals belong). The definition of dominance in (2.3) is applied. The non-dominated group is called the *front*, and the worst rank group is called the *cruft.* If an individual is dominated it will be placed in the cruft, no matter whether it is barely dominated by only one other design (like solution **a** in Figure 5) or if it is one of the worst solutions in the population (like solution **b** in Figure 5). The front, being the population's non-dominated set, is an approximation of the Pareto optimal front.

[5] See for example the telescope array design problem in section 3.5.

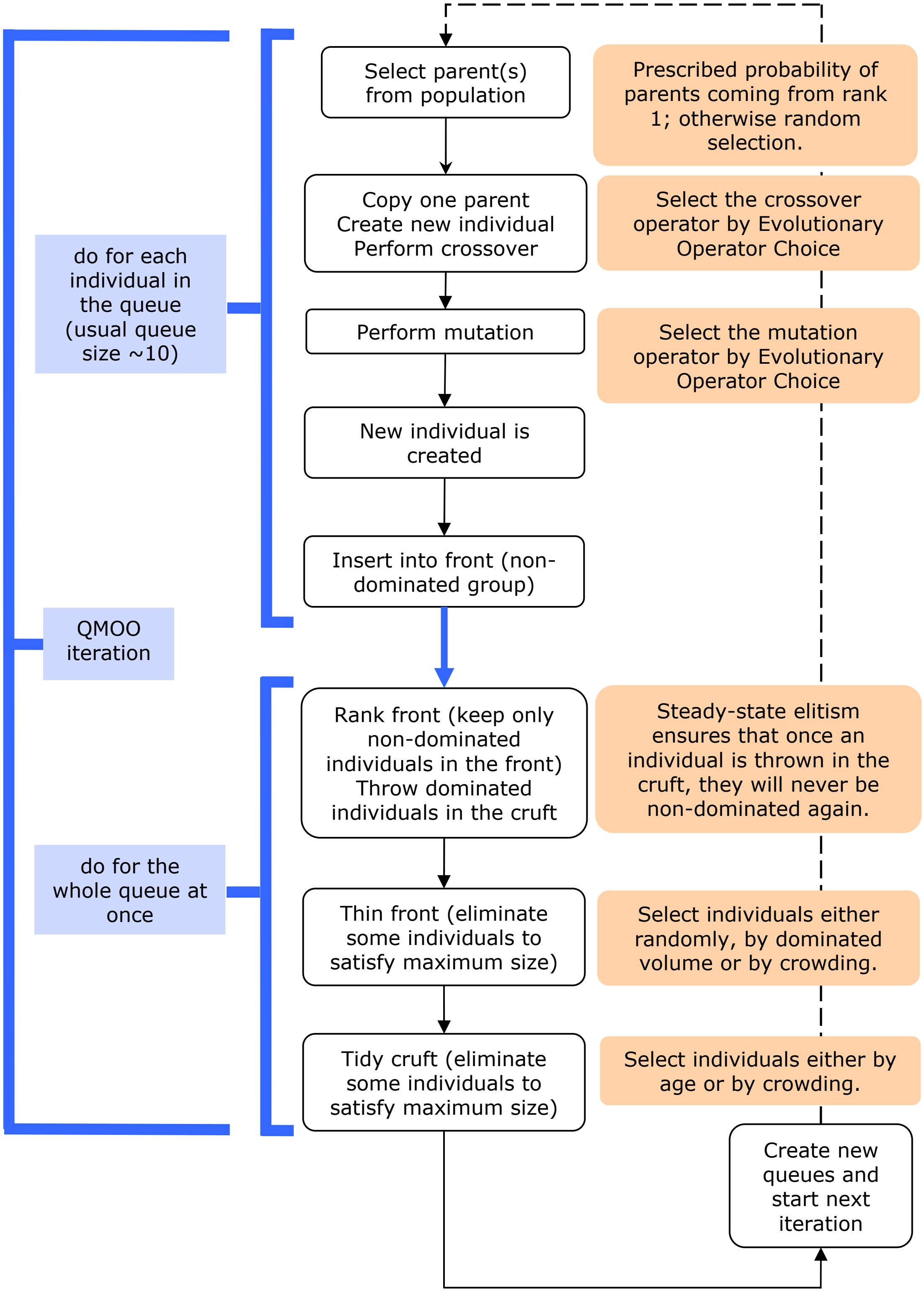

Select parent(s) from population
Prescribed probability of parents coming from rank 1; otherwise random selection.
Copy one parent Create new individual Perform crossover
Select the crossover operator by Evolutionary Operator Choice
do for each individual in the queue (usual queue size ~10)
Perform mutation
Select the mutation operator by Evolutionary Operator Choice
New individual is created
Insert into front (non-dominated group)
QMOO iteration
Rank front (keep only non-dominated individuals in the front) Throw dominated individuals in the cruft
Steady-state elitism ensures that once an individual is thrown in the cruft, they will never be non-dominated again.
do for the whole queue at once
Thin front (eliminate some individuals to satisfy maximum size)
Select individuals either randomly, by dominated volume or by crowding.
Tidy cruft (eliminate some individuals to satisfy maximum size)
Select individuals either by age or by crowding.
Create new queues and start next iteration

Figure 4. How QMOO works – the basic iteration.

The cruft has an exploratory nature and ensures the preservation of diversity in the population. For this reason, it completely disregards fitness. The cruft controls its state throughout the solution process through the criteria it uses for the selection of individuals to be eliminated. These criteria encourage the cruft's exploratory nature. Originally the selection criterion used was age, according to which older individuals are eliminated since they have already contributed to the solution. In this work, a crowding criterion is added according to which individuals in more crowded areas are eliminated, and a control process is developed in order to select between the two criteria[6].

The front is an elitist, steady state group which stores the best solutions discovered up to each instant. Like the cruft, it has a maximum size. When this size is exceeded, individuals in the front are selected for elimination. These individuals can either be selected randomly, by crowding (individuals in more crowded areas of the front are eliminated) or by a non-dominated volume metric (individuals which have the least effect on the reduction of the volume not dominated by the front are eliminated – this metric will be described in section 6.1.3). These elimination criteria for the front individuals are called *front thinning methods*, and they are an important factor which affects the solution performance. For example, it affects the way extreme areas of the front (the *tails*) which define the best solutions for each objective (anchor points) are discovered. In the course of this work, it is found that the crowded and the non-dominated volume thinning methods perform best, the choice between the two depending on the problem.

Usually the relative size of front and cruft ranges between 30% front - 70% cruft and 70% front - 30% cruft, in percentages of the total population size. The optimal value of this ratio depends on the problem. However the author's experience during the course of this work dictates that a ratio of around two thirds front – one third cruft provides satisfactory all-round performance in two-objective problems. A larger percentage of front individuals would probably be required in problems with many objectives.

This two-rank scheme performs well in multi-objective problems because it provides a simple means of separation between the elitist and the exploratory part of the population and makes it easier to control the balance between exploration and exploitation. Its great advantage in multi-objective problems is that even though there is only one 'good' rank, the amount and the spread of individuals along the non-dominated front is enough to ensure an *elitist diversity* – a large number of solutions which are potentially diverse in the design space, while sharing the characteristic of belonging to the first rank. This diversity can drive the solution towards the global optimum. The telescope array problem in chapter 3 is an example of the elitist diversity's positive effect.

According to the second ranking scheme a larger number of the top ranks, usually between ten and forty, are kept as separate groups in the population (see Figure 5). A variant of non-dominated sorting described in section 2.3 is used, with rank one being the best rank, two the second best and so on. This multi-rank scheme performs better than the two-rank scheme in single-objective problems. Indeed, if the two-rank scheme is used on a single objective problem, the front is composed of only a single individual: the best solution so far. This individual dominates all other solutions in the population, which are thrown in the cruft. As a result convergence towards the global optimum is very slow or unattainable since only one individual is elitist and all other are exploratory. On the contrary, a multi-rank scheme allows the population to retain several good solutions and helps it advance towards the global optimum.

[6] This process will be described in chapter 4.

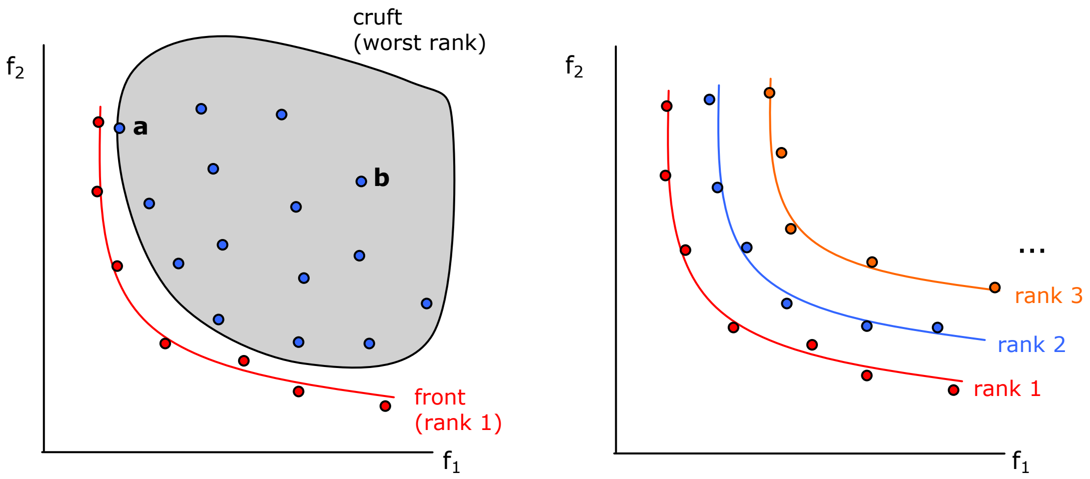


Figure 5. The two ranking schemes.

Encoding

Direct real-number encoding is used. The design vector is stored and processed as is by the algorithm. Hence in QMOO the genotype and phenotype of an individual are the same; an individual's chromosome is the design vector itself. This encoding requires the use of real-number assignment operators, which are described next.

Assignment

There are two kinds of continuous real-number assignment operators in QMOO: crossover (combination) and mutation. There is a number of available options for each kind of operator:

- **Crossover.** The crossover operators used in QMOO have been proposed in past literature.

  Blend crossover (BLX-α). The offspring is placed in a hypercube defined by the location of the two parents (see for example Eschelman, Schaffer 1993).

  Uniform crossover. This combination operator is mostly found in Evolution Strategies (Bäck, Schwefel 1993). The offspring takes each of its variable values from the corresponding value of one of the parents.

  Simulated binary crossover (SBX). This operator emulates the result of binary single-point crossover, for a real-valued chromosome (Deb, Agarwal 1995).

  Linear crossover. The offspring is placed somewhere on the linear segment connecting the two parents. This operator is referred to as *generalized intermediate combination* by Bäck (Bäck 1996).

  No crossover. The offspring is an exact copy of one of its parents.

- **Mutation.** The mutation operators used in QMOO were designed for this algorithm (Leyland 2002).

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

Global mutation. This operator changes all of the individual's variables by selecting them from a normal distribution that is centered on the individual, and with a span that is one twentieth of the search space for each variable.

Normal mutation. This 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.

No mutation. The individual is not altered.

A more detailed description of each of the operators can be found in Geoff Leyland's thesis (Leyland 2002).

When an individual is about to be created, a crossover and a mutation operator must be selected. This is done through the process of Evolutionary Operator Choice (EOC). According to EOC, operator selection follows an evolutionary process along with the solution – essentially the algorithm evolves and adapts itself to the problem while solving it[7]. In the beginning of the solution process, operators are selected randomly for each individual with an equal probability. Subsequently, each new individual has a positive probability of inheriting the operator used to create one of its parents. This probability is determined by the user, and it is usually set high (around 90%). The intention behind EOC is that if an operator produces successful children in a specific problem, it will be used increasingly (Leyland 2002). Although there are opportunities for improvement of the EOC process (for example defining 'success' more precisely than an individual's mere existence in the population), it makes the algorithm flexible and allows it to adapt to different problems. It can be argued that a significant part of QMOO's applicability and robustness are owed to the EOC.

An interesting note regarding the EOC and the real-valued mutation operators listed before is that in practice there is a very high probability that an offspring will be subjected to mutation. If for example we examine the first batch of offspring produced, when each operator still has an equal probability of being used, it is obvious that there is a 75% probability (three out of four) of some kind of mutation happening. In practice, as the solution advances, one of the mutation operators becomes dominant, and the probability that each offspring will be in some way mutated increases to around 85-95%. This behavior is in contrast with a binary genetic algorithm, where the probability of mutation is one or two orders of magnitude smaller (often less than 1%). On the other hand, the real-valued mutation operators described earlier usually move the individual by a relatively small amount from its initial location in the design space, while a binary bit-flip mutation can have a dramatic effect on the location of an individual. The end result is that mutation in QMOO works in a more homogeneous and incremental manner than in a GA.

Parent selection in the two-rank case is controlled by prescribing a specific probability that each of the parents will come from either the front or the cruft. Apart from this, parent selection is random. Very often this probability is prescribed in such a way that it reflects the relative size of the front and the cruft, in which case parent selection ends up being a uniform random pick from the whole population. For example if the front size is 70 individuals and the cruft size is 30 the probability that a parent will come from the front can be set at 0.70. Then the selection of each parent becomes a uniform random pick from the population. In general no selective bias towards parents of better rank needs to be applied since QMOO is a very elitist algorithm and if an

[7] An early form of a process similar to EOC for crossover adaptation was presented by Spears (Spears 1995), while Michalewicz and Fogel give a good overview of self-tuning for heuristic algorithms (Fogel, Michalewicz 2002).

individual simply exists in the population, it is considered to be good (Leyland 2002).

Grouping

QMOO performs grouping[8] in the design space in order to allow different regions to evolve independently in the population. The advantages of separating the population into groups which are independent to some extent have been studied by several researchers; an example is the *crowding* or *niching* methods evolved during the 1980s and 1990s (see for example Gruninger, Wallace 1996). With grouping, several local optima can be tracked simultaneously if the problem is multimodal since the global optimum does not necessarily render the less fit local optima extinct. The design space is explored more fully and diversity is better preserved, especially when cross-breeding between groups is allowed.

The grouping performed by QMOO is a separation of the population into different neighborhoods in the design space using a fuzzy c-means clustering algorithm (Leyland 2002). Grouping does not affect the individuals' fitness (in contrast to niching methods). The groups evolve somewhat independently: each group has its own front and cruft, hence there is no competition for dominance among groups. Cross-breeding is usually allowed.

A disadvantage of design space grouping is that is does not work well in problems with many design variables (in practice, more than 20 or so); objective space grouping can potentially be used in such cases, but this was not tried in the course of this work.

Elitism and exploration

QMOO is an intensely elitist algorithm. If an individual is good (i.e. non-dominated) it will survive indefinitely, unless it becomes dominated by a better individual or it is thinned out of the front by other non-dominated individuals. This provides the advantages of elitism, and makes it unnecessary to use any rank bias in parent selection (recall that parent selection can be essentially random).

This strong elitism is balanced by the existence of the cruft which ensures that a explorative ability remains with the population. This explicit separation between the functions of exploitation and exploration (in the form of the front and cruft groups) is one of QMOO's most positive characteristics. It makes it straightforward to understand and control the basic dynamics of the algorithm, and it will be further developed and used for the solution of time-changing problems (chapter 4) where the balance between exploration and exploitation becomes even more crucial.

QMOO is the algorithmic start point of this work. During the course of this thesis developments in the areas of constraint handling and time-changing optimization were implemented, leading it to evolve into its present form, D-QMOO.

2.5 Aspects of evolutionary optimization explored in this work

Evolutionary algorithms today still warrant a lot of development in order to be able to respond to the problems they are called to solve. In this work, we focus our attention into the solution of multi-objective time-changing problems. First, however, we engage in providing QMOO with a constraint handling ability.

2.5.1 Constrained problems

EAs are not naturally suited to handle constrained problems, and over the years a multitude of different constraint handling approaches has appeared. These approaches are very varied in terms

[8] Often referred to as 'clustering' in literature.

of their conceptual nature and their range of applicability.

However practically all design problems are constrained in one way or another and it is imperative for an optimization tool to have a constraint handling ability. In chapter 3, constraint handling with EAs is discussed along with a brief survey of modern evolutionary constraint handling methods. The formulation of a constraint handling method for QMOO focused on robustness and wide applicability is presented, tested on benchmark problems, compared with other methods, and applied on a telescope array design problem.

2.5.2 Time-changing problems

As was briefly discussed in the introductory chapter, a large number of real-world problems are of a nonstationary nature – either because externally specified parameters change, such as consumer preferences, or because they are defined on nonstationary systems that change in time, such as the pricing of airfares or the scheduling of a fleet of trucks. Evolutionary algorithms are conceptually well suited to handle time-changing problems since the natural evolution process they emulate is a problem of adaptation to a continuously changing fitness terrain, the external environment.

However an EA designed to solve static problems will likely not be able to solve a time-changing problem, or its performance in doing so will suffer. For example, a static algorithm might converge on the first timestep's global solution, but when the objective terrain changes the population will not have the diversity needed to explore the design space once more for the new solution and it will remain centered around the previous peak. Time-changing problems require tailored heuristics which provide a solver with the means to tackle a moving objective landscape. The core of this work (from chapter 4 onwards) is dedicated to the development of algorithmic architectures for the solution of such problems. Some of the methods developed are metamorphoses and evolutions of elements found in static optimization algorithms, while others are new concepts. Specific focus is given to multi-objective problems, since many real-world applications are characterized by conflicting trade-offs while at the same time there has been minimal research activity in the area of time-changing multi-objective optimization. At the same time, real world problems are continuously increasing in computational complexity – for example an airline with global operations has a multitude of different scheduling and routing options, and a portfolio manager has thousands of assets to select from. For this reason significant attention is given to the creation of heuristics which increase performance and allow the discovery of solutions using less computational time.

3 Solving Constrained Problems

Most evolutionary algorithms are designed to handle unconstrained optimization problems. Although EAs have seen extensive research and application during the past years, there still isn't an established, universal way of dealing with constraints. Constraint handling is a challenging problem, and this is reflected in the amount of work by various researchers in this area (see for example Hernandez, Aguire et al. 2003, Mezura-Montes, Coello Coello 2002, and Runarsson, Yao 2000). In this chapter constraint handling methods are developed for the D-QMOO evolutionary algorithm. The methods implemented here are primarily based on two techniques: first, *penalty methods* (see for example Fogel, Michalewicz 2002 for a general description), and second, the method of the *superiority of feasible points* (Powell, Skolnik 1993) and its variant, the *weighted superiority of feasible points*.

The first constraint handling method implemented is discussed in section 3.1. After a brief discussion on existing work in constraint handling with EAs, a description of penalty methods is given since these form the building block of the constraint handling techniques used in D-QMOO. Then, the superiority of feasible points (SFP) and its weighted version (WSFP) are discussed, with a variant of the WSFP method implemented in D-QMOO. Results from benchmark problems follow where the constraint handling version of D-QMOO is encouragingly compared with other algorithms.

A basic disadvantage of many constraint handling methods is that they rely on some kind of problem-specific parameter tuning. This often requires the user's intervention each time a new problem is solved. In order to achieve independence from parameter tuning, a second constraint handling method for D-QMOO is developed in section 3.4 (*superiority of feasible points with initial feasibility exploration*). This method has the advantage that it requires minimal problem-specific regulation. The chapter concludes with a practical application on a two-objective problem for the design of a telescope array, where D-QMOO is used to discover previously unknown non-dominated solutions.

3.1 Handling constraints with evolutionary algorithms

During the past two-and-a-half decades there has been a plethora of different approaches to the solution of constrained optimization problems with evolutionary algorithms. A thorough survey has been conducted by Coello Coello (Coello Coello 2002). In this survey most of the methods developed to date are classified into five categories:

Penalty Methods. If a solution is infeasible, the amount by which it violates the constraints is used to define a *penalized objective*. The penalized objective is used, instead of the objective, to set the individual's fitness (see for example Homaifar, Qi & Lai 1994).

Special Representations and Operators. These methods are problem-specific and are used when it is extremely difficult to locate even a single feasible solution. Custom genotype representations and genetic operators are designed, with the goal of allowing only feasible solutions to be represented (and hence exist) and ensuring that feasible solutions will produce feasible offspring (for example in Davidor 1991).

Repair Algorithms. Repair methods select the infeasible solutions in the population and alter them (move them in the design space) in order to make them feasible (the algorithm in Xiao et al. 1997 for example uses a repair operator). These methods usually lend themselves to combinatorial applications such as the traveling salesman problem in which it is relatively straightforward to alter a solution in order to make it feasible.

Separation of Objectives and Constraints. According to these methods, objective fitness and constraint violation are handled separately. Co-evolutionary methods, where two separate populations (one dealing with constraint violation and the other with fitness) are co-evolved according to a predator-prey model, are instances of such methods (Paredis 1994). Another instance is multi-objective methods, where constraints are treated as additional objectives in a multi-objective problem (see for example Coello Coello 2000b).

Hybrid Methods. This category contains methods which couple the evolutionary algorithm with some other technique in order to discover feasible solutions. Lagrange multipliers (Adeli, Cheng 1994) and fuzzy logic are examples of such techniques coupled with the EAs in order to handle constraints.

Categories such as special representations and operators and repair algorithms are normally problem-specific, an attribute which goes against our desire to create widely applicable tools. The constraint handling techniques formed here mainly stem from penalty methods and the separation of objectives and constraints.

3.1.1 Penalty Methods

Penalty methods are a common approach for constraint handling in EA's and other heuristic optimization methods. The general idea is to punish infeasible individuals by adding a penalty term to their objective function value. This term reflects the amount of constraint violation. Hence the fitness of infeasible individuals is worsened by an amount related to the extent of their constraint violation. The evolutionary algorithm then performs an unconstrained optimization process using this penalized objective as a fitness criterion. The co-existence of objective and constraint violation in the fitness function attempts to steer the population towards the optimal solutions, which are both feasible and well-performing.

Before we move on the constraint handling methods implemented in QMOO we will briefly discuss some basic elements of penalty methods which will be frequently used later on. Let us repeat the general problem defined in (2.1) at a fixed point in time:

$$
\begin{array}{llll}
\text{Find } \mathbf{x}^* \in \mathrm{X} \subseteq R^n & \text{which minimizes} & \mathbf{f}(\mathbf{x}) = \left[f_1(\mathbf{x}), ..., f_m(\mathbf{x}) \right] & \\
& \text{subject to} & l_i \le x_i \le u_i, & i = 1, ..., n \\
& & g_j(\mathbf{x}) \le 0, & j = 1, ..., q \\
& & h_j(\mathbf{x}) = 0, & j = q+1, ..., l
\end{array} \qquad (3.1)
$$

We can also denote as $S \subseteq R^n$ the search space defined by the design variable bounds l_i and u_i and as $F \subseteq S$ the feasible region defined by the equality and inequality constraints inside S.

Define a *violation function* for each constraint:

$$
viol_j(\mathbf{x}) = \begin{cases} \max\{0, g_j(\mathbf{x})\}, & j = 1, ..., q \text{ (inequality constraints)} \\ \max\{0, |h_j(\mathbf{x})| - \varepsilon_j\}, & j = q+1, ..., l \text{ (equality constraints)} \end{cases} \qquad (3.2)
$$

where ε_j is a finite tolerance for the j equality constraint. Then, the individual's penalized objectives to be used for ranking are:

$$
eval_k(\mathbf{x}) = f_k(\mathbf{x}) + r \sum_1^l viol_j^2(\mathbf{x}), \; k = 1, ..., m \qquad (3.3)
$$

$eval_k$ replaces f_k in the algorithm in order to specify fitness, and compare and rank individuals. As we can see a weighted quadratic penalty term has been added to the objective, worsening the fitness of infeasible individuals. The penalized objective in (3.3) is only one of many instances of penalty methods – for example, different forms of the violation function other than the quadratic have been tried in literature[1]. The *penalty weight* (or *penalty parameter*) r is a user-defined parameter that dictates the relative scaling between the objective and the constraint violation. A basic disadvantage of penalty methods is that r needs to be appropriately adjusted to each problem by the user, in order for the feasible optima to be discovered.

A feasible individual is ranked according to its objective function value only, since the constraint violation is zero. An infeasible individual has the penalty term added to its objective, making its rank worse than it would be based solely on its objective. This way the method attempts to favor feasible individuals and ensure their survival, in order to find the feasible optimal solution.

An illustration of the effect of the penalty on the objective terrain can be seen if we consider the following simple problem:

$$
\begin{array}{ll}
\text{minimize} & f(\mathbf{x}) = x_1^2 + x_2^2, \; \mathbf{x} = (x_1, x_2) \in \Re^2 \\
\text{subject to} & g(\mathbf{x}) = x_1^2 + x_2 - 1 \le 0 \\
& h(\mathbf{x}) = -x_1^2 + x_2 + 1 = 0
\end{array} \qquad (3.4)
$$

The violation functions are:

$$
viol_j(\mathbf{x}) = \begin{cases} \max\{0, g(\mathbf{x})\} \quad = \max\left\{0, \left(x_1^2 + x_2 - 1\right)\right\}, & j = 1 \\ \max\{0, |h(\mathbf{x})| - \varepsilon\} = \max\left\{0, \left|-x_1^2 + x_2 + 1\right| - \varepsilon\right\}, & j = 2 \end{cases} \qquad (3.5)
$$

If we add a quadratic penalty term to the objective function following (3.3), the penalized objective becomes:

[1] Some are listed in the survey by Coello Coello (Coello Coello, C. A. 2002).

$$
\begin{aligned}
eval(\mathbf{x}) &= f(\mathbf{x}) + r\sum_{1}^{2} viol_j^2(\mathbf{x}) \\
&= x_1^2 + x_2^2 + r\left(\max\left\{0,\left(x_1^2 + x_2 - 1\right)\right\}^2 + \max\left\{0,\left|-x_1^2 + x_2 + 1\right| - \varepsilon\right\}^2\right)
\end{aligned}
\tag{3.6}
$$

The penalty's effect can be seen in Figure 6. The feasible region is the segment of the equality curve which lies under the inequality curve (pointed out in black dash-dot line). For a penalty weight r = 10, the contours of the penalized objective function are shown, together with the two feasible optima. It is evident how the penalty term distorts the objective contours into guiding the solution towards the constrained optima, while otherwise the optimizer would lead the solution towards the unconstrained optimum at the origin.

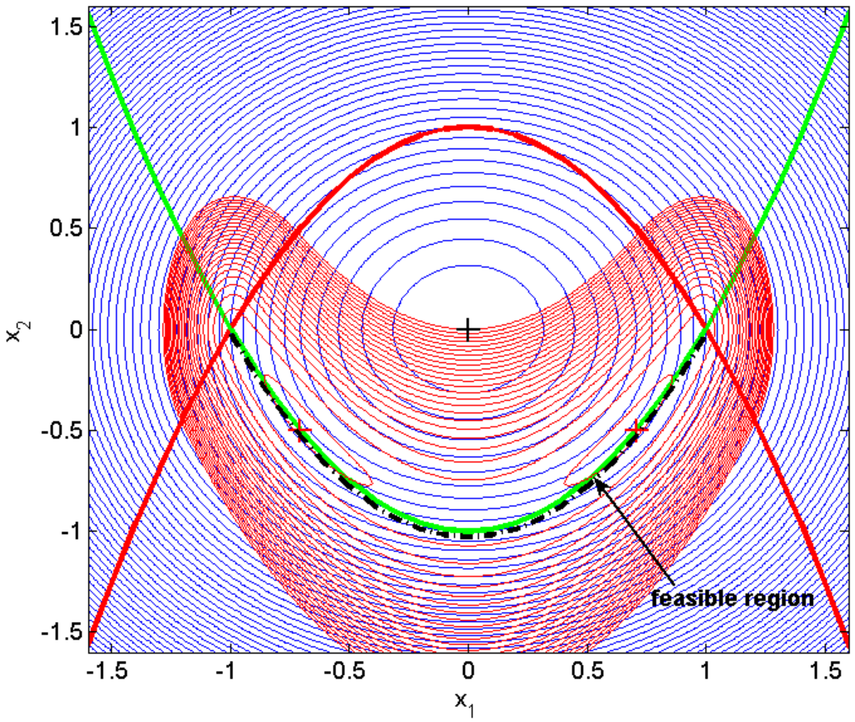


Figure 6. Distortion of the objective terrain due to the constraint violation penalty effect. The co-centric blue contours belong to the unconstrained objective. The red contours are the penalized objective. The green curve is the equality constraint and the red curve the inequality constraint. The feasible region is pointed out in black dash-dot line – it is the part of the equality curve lying under the inequality curve. We can see that the best penalized objective solutions are around the two red peaks (marked with red crosses), where three conditions coexist: The inequality and equality constraints are satisfied, and the unconstrained objective has the best possible value.

A case of overpenalization (where the penalty weight is too high) is illustrated in Figure 7. The same objective and penalized objective as in Figure 6 are shown in a three dimensional mesh view, only this time the penalty weight has been increased to 100. It is obvious how the objective's (lower surface) variations are relatively small compared to the penalized objective's variations – the constraint violation is the main fitness driver here. We can also see how, in the feasible region, the two surfaces converge to one since a feasible individual suffers no penalty.

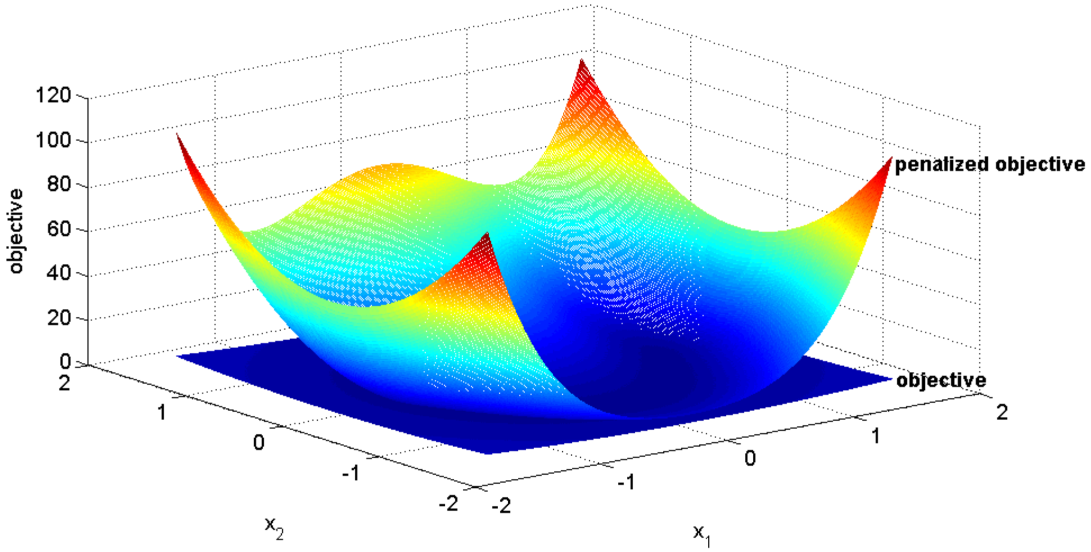


Figure 7. The result of an overly large penalty weight. The same problem as in Figure 6 is shown (with ten times larger penalty weight of 100) in a three dimensional view. The almost flat blue terrain is the unconstrained objective, and the curved surface the penalized objective. The constraint violation becomes the sole defining factor for the shape of the penalized objective terrain. The feasible solutions lie on the flat blue part of the penalized objective, and due to scaling they differ very little from each other, losing the objective's effect.

The method just described is an instance of a *static penalty method*. There are several other forms of penalty methods; some are simpler, such as *death penalty methods* where infeasible individuals are simply discarded from the population. Others are more complex, such as *adaptive* and *dynamic* methods, where the penalty changes as the solution advances. Penalty methods have been in general quite successful in handling a wide range of constrained optimization problems.

However, the success of penalty methods largely depends on the selected value of the penalty weights. Too small a weight will lead to a largely or fully infeasible population. Too large a weight can introduce other obstacles: if the penalty term in (3.3) is much larger than the objective such as the case in Figure 7 then we have a penalized objective terrain that ignores the objective values and only seeks to satisfy constraints. This can lead to premature convergence to a feasible but undesirable region of the search space. The need to tune a parameter is a disadvantage for an algorithm, especially when no rigorous method exists for the tuning and the users need to do it themselves. In fact, much of the effort in constrained optimization research has been towards finding methods which are independent of parameters that require tuning - see for example the Stochastic Ranking method of Runarsson and Yao (Runarsson, Yao 2000). This issue will be revisited later in this chapter.

A basic question when solving constrained problems with population-based algorithms, is whether the algorithm allows an infeasible individual to be ranked as better than a feasible one. A potential disadvantage of penalty methods is that they do not give a clear (yes-no) answer to this question – it all depends on the penalty weight selected by the user, and even with a constant weight the method's behavior regarding this aspect might change during the various stages of a single optimization run. A clear interpretation of the effect of the relative magnitudes of objective and penalty on the ranking of individuals is given by Runarsson and Yao (Runarsson, Yao 2000). The essence is that by reducing the weight *r* a feasible individual can be ranked as worse than an infeasible one – there is no conceptual mechanism excluding the possibility of an infeasible individual being fitter than a feasible one. So, penalty methods do not create a clear distinction between feasible and infeasible individuals. The superiority of feasible points described next addresses this issue.

3.1.2 Superiority of Feasible Points

Initially introduced by Powell and Skolnick (Powell, Skolnik 1993), the *superiority of feasible points* (SFP) has inspired various techniques, for example the one found in (Deb 2000). It can be considered as a penalty method, since it attempts to force the population on the feasible plane by altering the objective value and favoring feasible individuals over infeasible ones. However conceptually it is classified as a separation of objectives and constraints method, since the optimization process clearly discerns between feasible and infeasible individuals.

According to the SFP the following ranking scheme is used during the optimization process:

> *Among two infeasible individuals, the one with the lowest penalized constraint violation has a better rank.*
>
> *Among an infeasible and a feasible individual, the feasible one has a better rank.*
>
> *Among two feasible individuals, the other having a better objective value has a better rank.*

The penalized objective becomes:

$$eval_k(\mathbf{x}) = \begin{cases} f_k(\mathbf{x}), & \mathbf{x} \text{ feasible} \\ 1 + r\sum_1^l viol_j^2(\mathbf{x}), & \mathbf{x} \text{ infeasible} \end{cases} \tag{3.7}$$

where the objective is scaled so that feasible individuals lie in the range $(-\infty,1)$ and infeasible individuals lie in the range $(1,+\infty)$. This way the ranking scheme stated above is satisfied.

3.1.3 Weighted Superiority of Feasible Points

The constraint handling method initially implemented in D-QMOO is called *weighted superiority of feasible points* (WSFP). It is a hybrid between penalty methods and SFP. A detailed description of this variant can be found in (Fogel, Michalewicz 2002)[2]. Here we will give a brief presentation.

The main difference between the WSFP and the SFP methods is in the way infeasible individuals are compared. Specifically, the WSFP method retains an objective value term for the infeasible individuals. This way infeasible individuals are judged by a combination of their constraint violation and their objective, while at the same time a separate term ensures that feasible individuals always have better penalized objective than infeasible ones.

Define an additional term, the *superiority function*:

[2] This method is referred to by Fogel and Michalewicz simply as *superiority of feasible points*.

$$\theta_k(\mathbf{x}) = \begin{cases} 0, & \mathbf{x} \text{ feasible} \\ \max\{0, \max_{\mathbf{x}\in F}\{f_k(\mathbf{x})\} - \min_{\mathbf{x}\in S-F}\{f_k(\mathbf{x}) + r\sum_{1}^{l} viol_i(\mathbf{x})\}\}, & \mathbf{x} \text{ infeasible} \end{cases} \tag{3.8}$$

Then the penalized objective of an individual becomes:

$$eval_k(\mathbf{x}) = f_k(\mathbf{x}) + r\sum_{1}^{l} viol_j^2(\mathbf{x}) + \theta_k(\mathbf{x}) \tag{3.9}$$

The superiority term θ ensures that feasible individuals have a better objective value than the already penalized infeasible individuals. It does so by introducing a discontinuity in the penalized objective terrain: it takes the worst feasible individual and the best infeasible individual in terms of penalized objective, and calculates the difference. If this difference is positive (which means that, even after applying the penalty term, there are still some infeasible individuals who are fitter than some feasible ones) then it is added to every infeasible individual's objective. This way:

> *Among two infeasible individuals, the one with the lowest combination of penalized constraint violation and objective has a better rank.*
>
> *Among an infeasible and a feasible individual, the feasible one has a better rank.*
>
> *Among two feasible individuals, the other having a better objective value has a better rank.*

The infeasible individuals' penalized objective retains an objective value term in hope of guiding the infeasible part of the population towards regions of better objective. This way the infeasible solutions are encouraged to take short cuts through infeasible 'canals' in order to get to feasible regions of better objective values. This is to the benefit of computational efficiency. The price to pay is that the penalty weight factor remains and needs to be tuned. However its significance becomes smaller, since the superiority term ensures the dominance of feasible solutions.

Indeed, a distinct positive characteristic of the SFP and WSFP as opposed to penalty methods is that they conceptually distinguish feasible and infeasible individuals without requiring any parameter tuning to do so. Even if r was set to zero, the feasible population would still be ranked as better than the infeasible by virtue of the θ term. In order to visualize this, we can imagine that the SFP introduces a discontinuous step in the terrain at the boundaries of the feasible region, with a step height equal to θ.

Let us take a modified version of problem (3.4):

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) = x_1^2 + x_2^2,\ \mathbf{x} = (x_1, x_2) \in \Re^2 \\ &\text{subject to} && g(\mathbf{x}) = x_1^2 + x_2 - 1 \geq 0 \end{aligned} \tag{3.10}$$

The equality constraint is no longer there, and the inequality constraint has been inverted so that the feasible region is now above the g curve, as shown in Figure 8. First let us apply a penalty method such as (3.3):

$$\begin{aligned} eval(\mathbf{x}) &= f(\mathbf{x}) + r viol_1^2(\mathbf{x}) \\ &= x_1^2 + x_2^2 + r\max\left\{0, -\left(x_1^2 + x_2 - 1\right)\right\}^2 \end{aligned} \tag{3.11}$$

using, this time, a penalty weight r of 1. The penalized objective yields an 'optimal' solution that is infeasible. This can be seen in Figure 8 – the penalty weight is too small, and the penalized objective contours lead to a pair of solutions well within the infeasible region. If the fitness function is based on (3.11), the feasible optimal solution will not be discovered and the algorithm will converge on the infeasible solutions.

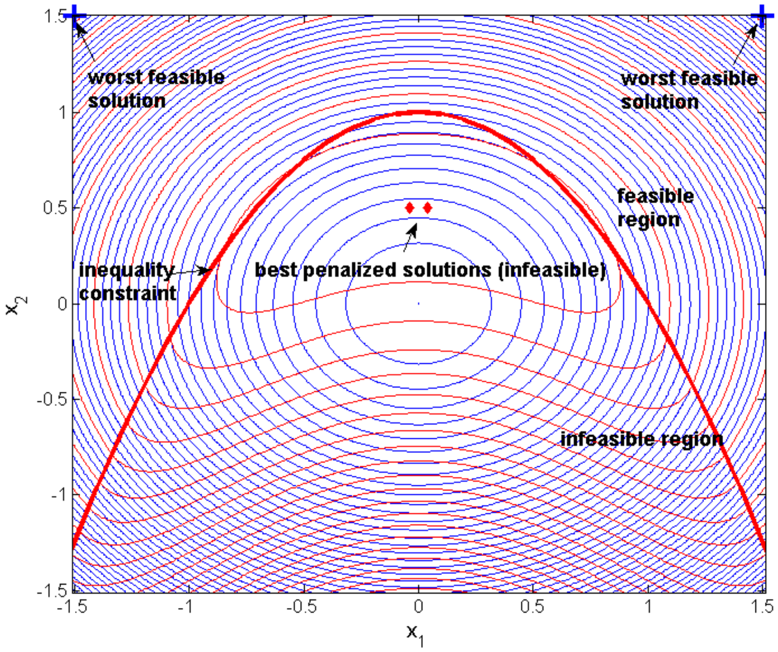


Figure 8. Contours of the objective and penalized objective terrain for problem (3.10), by simple application of a penalty method of the form (3.3). The objective contours are shown as co-centric blue lines, the penalized objective contours as distorted red lines and the inequality constraint as a thick red curve. Due to the *r* parameter being too small, the best penalized solutions are infeasible (diamond-shaped points).

Now let us apply the WSFP method and introduce the superiority function θ as shown in (3.8). In this hypothetical case, the worst feasible solutions in the design space (shown in blue crosses) and the best infeasible solutions are used to define θ. The fitness evaluation is now based on the function:

$$\begin{aligned} eval(\mathbf{x}) &= f(\mathbf{x}) + rviol_1^2(\mathbf{x}) + \theta(\mathbf{x}) \\ &= x_1^2 + x_2^2 + r\max\left\{0, -\left(x_1^2 + x_2 - 1\right)\right\}^2 + \theta(\mathbf{x}) \end{aligned} \tag{3.12}$$

The result can be seen in Figure 9. The superiority function has introduced a step in the penalized objective terrain, which ensures that infeasible individuals are worse than feasible ones. This has been done independently of the penalty weight *r*. The feasible optima now lie at the location with the minimum penalized objective value, and the algorithm can discover them.

3.2 Implementation of the WSFP method in QMOO and comparison with other algorithms

Often, optimization problems encourage the design of a customized constraint handling method in order for them to be solved by an evolutionary algorithm. Such customized methods are usually of the special representation and operator variety: the evolutionary algorithm is designed on the specific instance or type of problem so that the chromosome encoding and the selection, combination and mutation operators inherently satisfy constraints[3]. This approach provides good performance on the problems it is designed to solve; however it is capable of solving only those problems. Penalty methods on the other hand are more widely applicable since they pose almost no restriction on the problem characteristics. They have two main disadvantages though: first, their performance directly depends on the correct tuning of the penalty weights; second, they do not have an inherent way of separating feasible and infeasible solutions – the penalized objective $eval_k(\mathbf{x})$ in equation (3.3) essentially does not know if the individual **x** is feasible or not.

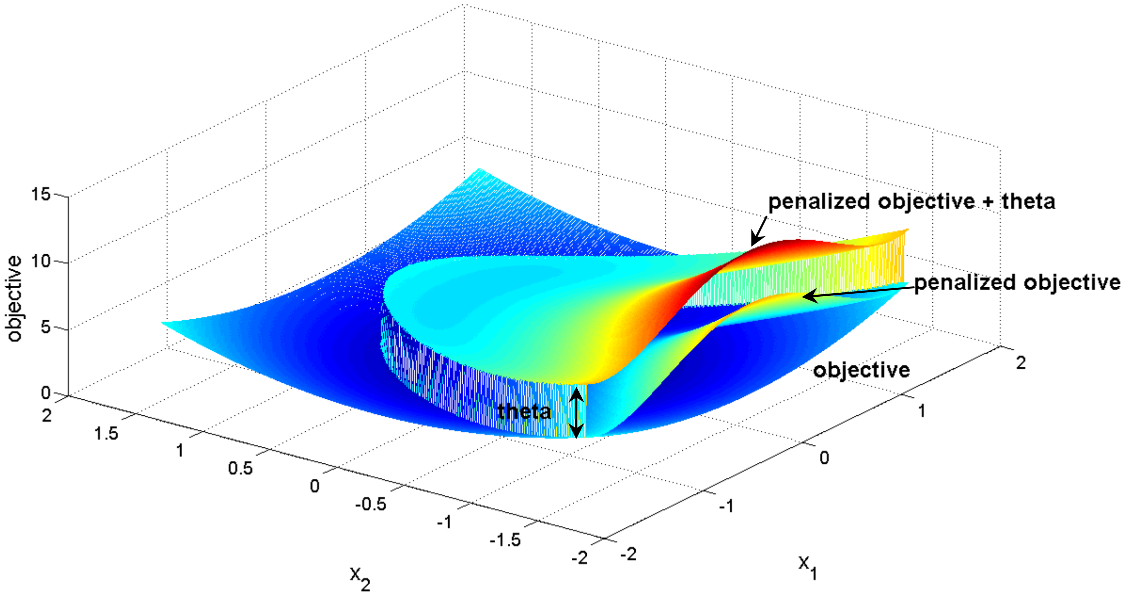


Figure 9. The superiority function θ introduces a step in the objective terrain, separating feasible and infeasible areas and ensuring that infeasible areas are at a worse (higher) level.

The WSFP method does not suffer from the second disadvantage, and only partly depends on penalty weight tuning. After experimentation on several problems with numerous other methods, the version of WSFP that was designed for D-QMOO was found to perform well. Therefore it was implemented in the algorithm in order to provide an initial constraint handling capability that is robust and widely applicable.

In this section we present results from solving a set of benchmark problems (Koziel, Michalewicz 1999, Michalewicz 1995) which have been widely used for testing the performance of evolutionary algorithms for constrained optimization[4]. We compare the performance of D-

[3] For example, the GENOCOP algorithm by Zbigniew Michalewicz and Cezary Janikow (Michalewicz, Janikow 1991) solves problems with linear constraints by starting from feasible solutions and ensuring through its specialized operators that subsequently created individuals are also feasible, while in the work of Davidor (Davidor 1991) a varying chromosome length genetic algorithm with specialized operators is designed for the creation of robot trajectories.

[4] Benchmark problem definitions are given in the appendix.

QMOO/WSFP with a number of other algorithms:

1. Constrained Optimization by Multiobjective Genetic Algorithms – COMOGA (Surry, Radcliffe 1997).
2. Vector Evaluated Genetic Algorithm – VEGA (Coello Coello 2000b, Shaffer 1985).
3. Multi-Objective Genetic Algorithm – MOGA (Coello Coello 2000a, Fonseca, Fleming 1993).
4. Niched-Pareto Genetic Algorithm – NPGA (Coello Coello, Mezura-Montes 2002, Horn, Nafpliotis & Goldberg 1994).
5. Inverted-Shrinkable Pareto Archived Evolution Strategy – IS-PAES (Aguirre et al. 2003).
6. Stochastic Ranking – SR (Runarsson, Yao 2000).

For each benchmark problem and each algorithm, a set of 30 runs was executed. The results of these runs and their statistics are shown in Table 1. Results from algorithms other than D-QMOO are as reported by Aguirre et al. (Aguirre et al. 2004). The performance of each algorithm regarding the best solution discovered is illustrated in Figure 10, where the top performers for each problem have been noted.

3.2.1 Discussion on performance

In general D-QMOO with WSFP performs well. It finds the best solution over all other algorithms in one case (g10) and also in two more cases it finds the best overall solution together with another algorithm: g05 (together with SR) and g06 (together with SR and IS-PAES). The best overall in problem g11 found by the various algorithms depends on the user-defined value of the equality constraint tolerance ε in equation (3.2), and given that a relatively stricter tolerance of 10^{-4} is used by D-QMOO, we can consider D-QMOO's performance to be equivalent to the other six algorithms. Hence in four out of six problems D-QMOO discovers the best design. In the other two problems, D-QMOO discovers the second or third best solution with a small difference from the overall best.

We must note D-QMOO's performance in the g02 problem. The global optimum in this problem is unknown – the best reported solution to date has been the one found the Stochastic Ranking method at -0.803619, and g02 is considered to be a very hard constrained optimization problem (Fogel, Michalewicz 2002, Coello Coello, 2002). D-QMOO finds a solution that is very close to the best known, at -0.803562 (around 0.01% difference). Actually, a design with an objective of -0.806000 was discovered while D-QMOO was being tested. This is better than any known published solution, but was outside the 30-run averaged set and is not reported as a result in Table 1.

3.2.2 Robustness and consistency

We can identify two measures of algorithm robustness that can be quantified using the available results. The first is the worst final solution reported from the 30-run sets (last column of Table 1), which expresses a worst-case scenario for the performance of the algorithm. The second is the ability of the algorithm to actually discover feasible solutions, which is especially important for problems with very small and awkwardly-shaped feasible regions.

D-QMOO shows good robustness under both measures. In three problems (g05, g06, g10) D-QMOO has the least bad worst solution. In one problem (g02) it has the second lowest after IS-PAES. Especially in g06, D-QMOO produced a worst solution value of -6961.8139 for each of the 30 runs, which is also the same as the best overall solution. In terms of the second measure, in all problems D-QMOO discovers feasible solutions, including the g13 problem where the majority of algorithms does not manage to find the feasible region.

Another positive attribute of D-QMOO's performance is its consistency: in four of the benchmark problems (g05, g06, g10, g11) it has the lowest – practically zero – standard deviation of the best discovered solution. In g02 D-QMOO's standard deviation is near the average of the seven algorithms. Only in g13 is it larger than IS-PAES and SR (the other algorithms did not find any feasible solutions). Especially in the case of the g10 problem D-QMOO finds the best overall design with a standard deviation that is four orders of magnitude lower than the next largest one - this is an encouraging result since g10 is considered in literature to be a hard problem (Coello Coello 2002).

Table 1. Comparative results for benchmark problems. 'Best' and 'Worst': the best and worst result each algorithm discovered in the 30 runs. 'Median', 'Mean' and 'St. Dev.' are the statistics of each algorithm's best results in the 30 runs. 'N.F.' means that the algorithm did not manage to discover a feasible solution. The best performing algorithms are highlighted in grey.

Test Problem	Algorithm	Optimal (known)	Best	Median	Mean	St. Dev.	Worst
g02	COMOGA	-0.803619	-0.021716	-0.017607	-0.016409	0.003410	-0.007805
	VEGA		-0.000212	-0.000048	-0.000077	0.000057	-0.000008
	MOGA		-0.680874	-0.569982	-0.58471	0.048400	-0.499295
	NPGA		-0.790404	-0.772691	-0.769520	0.012923	-0.739923
	IS-PAES		-0.803376	-0.793342	-0.793281	0.009000	-0.768291
	SR		-0.803619	-0.785800	-0.781975	2.0E-02	-0.726288
	D-QMOO		-0.803562	-0.795090	-0.793685	0.009821	-0.758566
g05	COMOGA	5126.4981	N/A	N/A	N/A	N/A	N/A
	VEGA		N/A	N/A	N/A	N/A	N/A
	MOGA		N/A	N/A	N/A	N/A	N/A
	NPGA		N/A	N/A	N/A	N/A	N/A
	IS-PAES		N/A	N/A	N/A	N/A	N/A
	SR		5126.497	5127.372	5128.881	3.500000	5142.472
	D-QMOO		5126.497	5126.497	5126.497	0.000000	5126.497
g06	COMOGA	-6961.8139	-6622.2803	-6157.5530	-6058.8651	436.7865	-4859.3311
	VEGA		-6941.9321	-6871.1799	-6873.1397	46.6093	-6743.4951
	MOGA		-6957.9507	-6906.5984	-6903.7747	29.7420	-6845.4321
	NPGA		-6956.9717	-6818.0115	-6776.4188	176.1811	-6310.1255
	IS-PAES		-6961.8140	-6961.8140	-6961.8130	0.00085	-6961.8100
	SR		-6961.8140	-6914.8140	-6875.9400	0.0130	-6350.2620
	D-QMOO		-6961.8139	-6961.8139	-6961.8139	0.0000	-6961.8139

Table 1 (continued).

Test Problem	Algorithm	Optimal (known)	Best	Median	Mean	St. Dev.	Worst
g10	COMOGA	7049.3307	11129.1709	15952.2603	15875.6988	2371.5133	20528.0488
	VEGA		11259.6113	13283.8696	14046.4097	2773.2631	22271.4883
	MOGA		7372.4600	8316.9732	8566.3080	1159.5131	12552.2305
	NPGA		8812.4356	9896.3452	11134.7271	2381.9406	15609.1631
	IS-PAES		7062.0190	7448.0140	7342.9440	140.0000	7588.0540
	SR		7054.3160	7372.6130	7559.1920	530.0000	8835.6550
	D-QMOO		7049.2480	7049.2499	7049.2720	0.0747	7049.6041
g11	COMOGA	0.7500	0.74901	0.74930	0.74930	0.00019	0.74988
	VEGA		0.74943	0.75290	0.76015	0.01865	0.81191
	MOGA		0.74900	0.74904	0.74906	0.00007	0.74931
	NPGA		0.74901	0.74905	0.74910	0.00014	0.74971
	IS-PAES		0.75000	0.75000	0.75000	0.00026	0.75000
	SR		0.75000	0.75000	0.75000	0.00008	0.75000
	D-QMOO		0.74999	0.74999	0.74999	0.00000	0.74999
g13	COMOGA	0.053950	N.F.	N.F.	N.F.	N.F.	N.F.
	VEGA		N.F.	N.F.	N.F.	N.F.	N.F.
	MOGA		N.F.	N.F.	N.F.	N.F.	N.F.
	NPGA		N.F.	N.F.	N.F.	N.F.	N.F.
	IS-PAES		0.05517	0.2779	0.28184	1.776E-01	0.5471
	SR		0.053957	0.057006	0.067543	3.1E-02	0.216915
	D-QMOO		0.056071	0.469560	0.488702	0.280058	0.969780

In terms of the ratio between feasible and infeasible individuals in the population, the make-everyone-feasible effect of the WSFP method became apparent during the experiments. In most runs, the average final feasible percentage was near 100%, and only in one case (g06) was it below 50%.

Overall these results are encouraging for the constraint handling ability of D-QMOO. Especially the fact that D-QMOO often produced results of comparable or better performance than Stochastic Ranking is very encouraging, since SR is considered today to be the state-of-the-art in constrained evolutionary optimization.

3.3 Constrained multi-objective problems

The previous results pertained to single-objective problems, for which constrained evolutionary optimization has been studied extensively. In the current section some results from the solution of a constrained multi-objective benchmark problem are given, since D-QMOO is a multi-objective algorithm. This area has been studied comparatively less, but a presentation of some benchmark problems can be found in (Deb, Pratap & Meyarivan 2002).

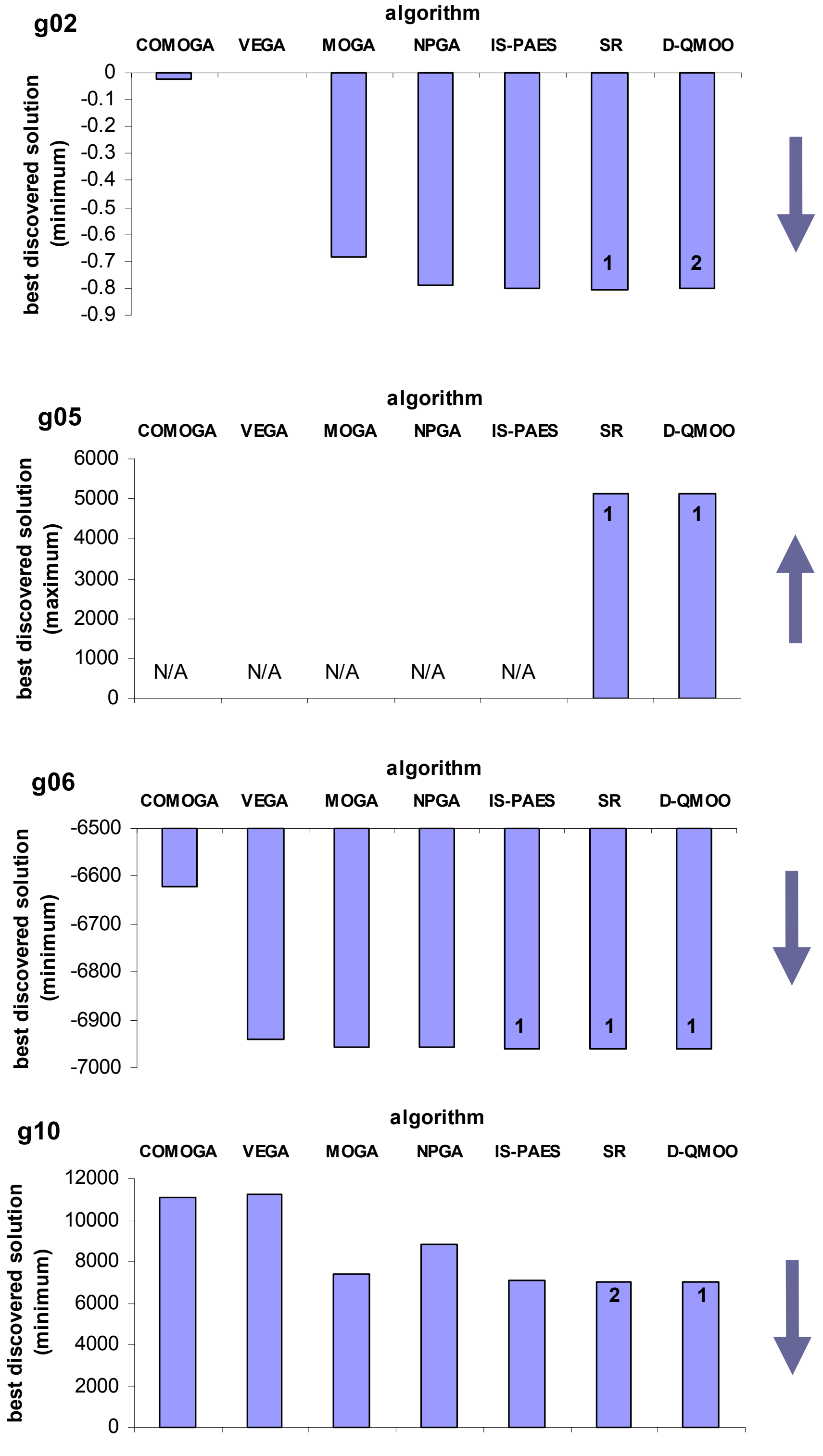


Figure 10. Graphic representation of the comparative results in the constrained benchmark problems. The best discovered solution is shown for each algorithm. The best performing algorithms are noted for each problem, together with the direction of optimization. Note that in *g11* the performance of all algorithms is considered equivalent.

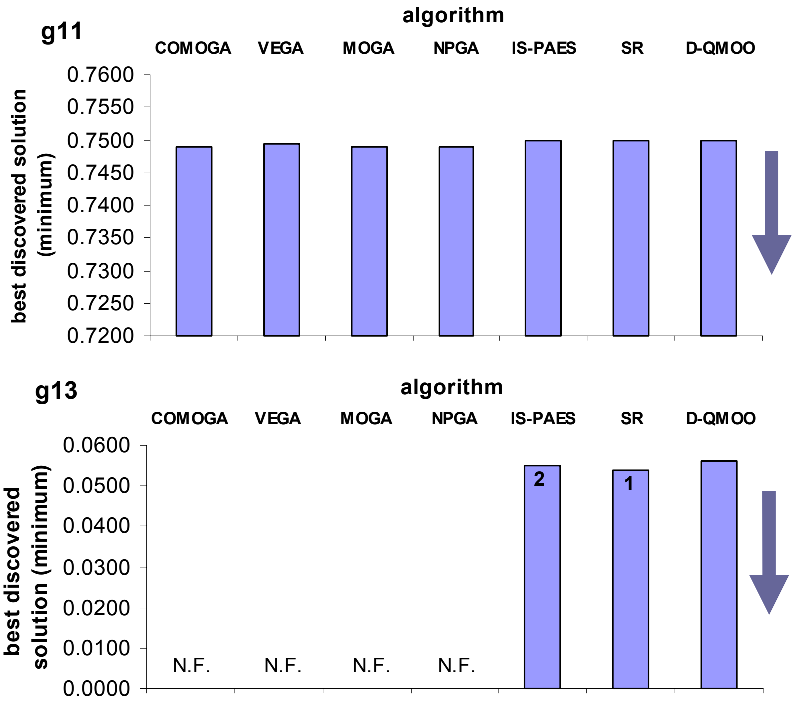


Figure 10 (continued).

In Figure 11 and Figure 12 we can see the Pareto front approximation by D-QMOO for the TNK problem[5] (Tanaka 1995). The algorithm discovers the non-dominated front of the feasible region. The two instances shown also illustrate the effect of the thinning method used for the front (recall the discussion in section 2.4). In the first case (Figure 11), crowded thinning is used – front individuals are selected for elimination based on how close they are to each other. This is a popular thinning method, used in several algorithms such as NSGA-II (Deb et al. 2000). In the second case (Figure 12), non-dominated volume thinning is used – front individuals are selected for elimination based on their contribution to the reduction of the front's non-dominated volume (individuals with the smallest contribution are eliminated – see section 6.1.3 for a detailed description). We can see in this case that the flat and vertical segments around the middle of the front are not covered by individuals, since they would not contribute much to the volume reduction.

3.4 Achieving further independence from penalty weights - Superiority of Feasible Points with Initial Feasibility Exploration

A basic problem was encountered during the previous experiments with the WSFP method in cases where the feasible region is very small and skewed compared to the design space, and/or the orders of magnitude of objective and constraint violation are very different. In those cases the code has great difficulty in discovering some initial feasible solutions, in order to calculate the superiority function θ and apply the WSFP. The remedy used was to tune the penalty weights until the violation penalties became strong enough to guide the population towards the feasible region(s). These are the only cases where WSFP is vitally dependent on penalty weights.

[5] The problem definition is given in the appendix.

In order to counter this issue, a method is implemented for which parameter tuning is not vital. This method has not previously appeared in literature, to the best of the author's knowledge, however it is not particularly revolutionary since other researchers have followed similar paths (see for example Coello Coello, Lamont 2004). We call this method *superiority of feasible points with initial feasibility exploration* (SFP-IFE).

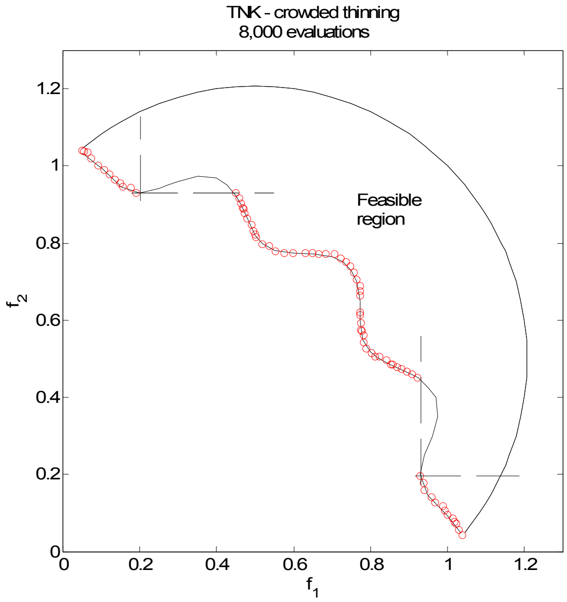


Figure 11: TNK constrained multi-objective problem Pareto front approximation (crowded thinning).

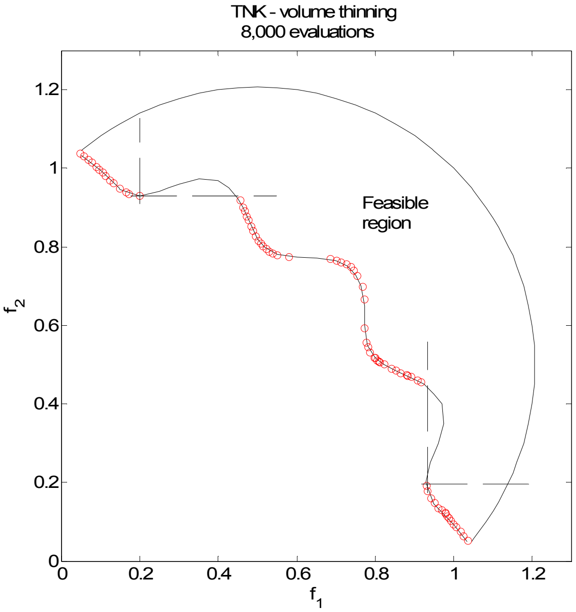


Figure 12: TNK constrained multi-objective problem Pareto front approximation (non-dominated volume thinning).

The SFP-IFE method works in two separate stages during the solution process, according to whether feasible individuals have been discovered or not:

> *When no feasible individuals exist in the population group (first stage):*
>
> > *Among two individuals, the one with the lowest constraint violation has a better rank.*
>
> *When at least one feasible individual has been found in the group, the WSFP method is used (second stage):*
>
> > *Among two infeasible individuals, the one with the lowest combination of penalized constraint violation and objective has a better rank.*
> >
> > *Among an infeasible and a feasible individual, the feasible one has a better rank.*
> >
> > *Among two feasible individuals, the other having a better objective value has a better rank.*

Hence initially each cluster in the population only searches for feasible regions in the design space, without paying any attention to the objective function values. This is what the original SFP method would do but in this case it is applied only during the initial stage of the solution process. The fitness of each individual is defined by the quadratic violation penalty term only. As soon as the first feasible individual appears in a cluster, the whole cluster changes its fitness evaluation to the WSFP values.

According to the SFP-IFE the fitness of an individual is defined as follows:

$$eval_k(\mathbf{x}) = \begin{cases} \sum_{1}^{l} viol_j^2(\mathbf{x}), & \text{if there are no feasible individuals} \\ & \text{in } \mathbf{x}\text{'s group (first stage).} \\ f_k(\mathbf{x}) + r\sum_{1}^{l} viol_j^2(\mathbf{x}) + \theta_k(\mathbf{x}), & \text{if there is at least one} \\ & \text{feasible individual in } \mathbf{x}\text{'s group (second stage).} \end{cases} \tag{3.13}$$

The penalty weight *r* is not required any more in the initial part of the search. The cluster searches for a feasible region using only constraint violation as fitness. Hence no comparison between the objective and constraint values is done, making the penalty weight redundant. When a feasible region is discovered by at least one individual, the WSFP method is switched on. The WSFP method together with D-QMOO's elitism ensure the survival of the feasible part of the population. Since the WSFP method introduces a fitness step between feasible and infeasible individuals based on their objective and constraint values, the penalty weight is not vital in this second stage as it would have been in the first stage when searching for feasible solutions.

When tested on the benchmark problems presented in the previous section the SFP-IFE method had identical performance to the WSFP method, except in hard problems where the feasible region is difficult to discover, such as *g13*. In those cases the SFP-IFE method discovered feasible solutions much faster without requiring any tuning. This is the main advantage of this method, together with the computational time (function evaluations) it saves during the discovery of the feasible region. Other than that, its performance and behavior is similar to that of the WSFP method. For problems with large, easy-to-discover feasible regions, there is no substantial difference from the WSFP method since feasible solutions are found almost immediately and the WSFP is used for fitness calculation.

This method attempts to retain only the positive effect of penalty weights. As the population is searching for feasibility (when penalty weight dependence would be a liability), there is no dependence on such weights. When feasible solutions are discovered, then the positive influence of combining the constraints and objectives in order to set the fitness of infeasible individuals is exploited. In practice, the weight r is often permanently set to 1 and hence discarded, unless it is believed that a significant gain in performance will happen if it is actively used.

Future enhancements to the SFP-IFE method could include some kind of autonomous tuning of the penalty weight (several methods exist in literature, see for example Coello Coello 2002) and a more detailed examination of how the relative constraint values affect the search for feasible areas[6].

3.5 Practical application: multi-objective optimization of a telescope array

A real-world application on the design of a telescope array is explored in this section. This problem is an interesting instance of a static constrained multi-objective problem, and serves as an initial practical demonstration of D-QMOO. The problem consists of configuring an array of radio telescope stations by deciding where to place each station. It is derived from the actual LOFAR (LOw Frequency ARray) which is being planned as a staged deployment project (Bounova, de Weck 2005). There are two conflicting objectives: the installation cost which is to be minimized, and the array's performance which is to be maximized.

The technique of multiple antenna radio astronomy[7] involves using a number of antennas that are linked together. These antennas create an aperture which resolves features with much smaller angles and thus probes deeper into space. Projects such as the Very Large Array shown in Figure 13 are examples of such apertures. LOFAR is designed for astronomical observations at radio-frequencies below 250 MHz, and consists of a large number of simple stations (Bounova 2005).

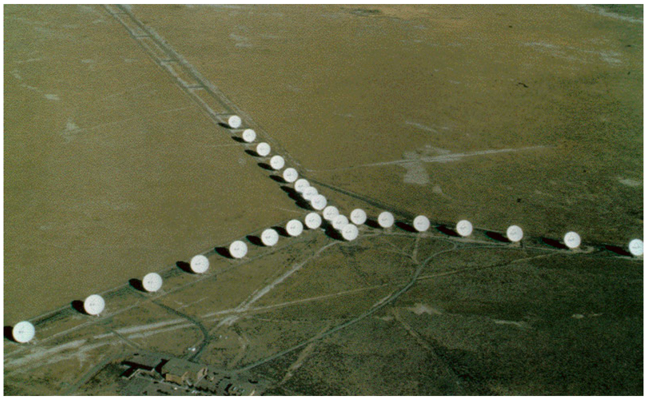


Figure 13. Very Large Array (VLA), San Agustin NM (photo courtesy of NASA/JPL).

There is no known analytic solution to this array design problem. Up to now it has been approached with a multi-objective genetic algorithm which was custom-designed for this application. Cohanim et al. first approached it in 2004 (Cohanim, Hewitt & de Weck 2004); Bounova and de Weck have more recently treated its multi-stage version using a combination of a GA and simulated annealing (Bounova, de Weck 2005).

[6] However, constraint violation functions have an inherent advantage in terms of scaling: their desired value is always zero, in contrast with objective functions which may span any range of values.

[7] Known as *interferometry*.

For the single-stage version of the problem D-QMOO was able to produce solutions which in several cases significantly improved the existing results, especially in the middle region of the Pareto front where a solution that will actually be implemented is likely to lie. In the results that follow D-QMOO's solutions are compared with the best previously known solutions, derived with Cohanim's algorithm.

3.5.1 Problem statement

The problem definition is given in Table 2. Each solution consists of an array configuration and is expressed by the x-y coordinates of every station. The telescope array performance is expressed by the normalized uv density metric *P*, which must be minimized in order to maximize performance (Bounova, de Weck 2005). The cost objective expresses the cable length connecting the stations. A geometric constraint for the problem requires all stations to lie inside a circle with a 400·*N*/60 radius, where *N* is the number of stations. There is also a performance constraint requiring a maximum 0.70 uv density metric.

The superiority of feasible points with initial feasibility exploration as described in the previous section is used as a constraint handling method.

Table 2. Telescope array optimization problem definition.

	LOFAR Problem Statement
minimize	$\mathbf{f}(\mathbf{x})=[Cost,P]$
subject to:	each station lying in a $400\cdot N/60$ radius circle (geometric constraint) $P\leq 0.70$ (performance constraint)
design vector:	$\mathbf{x}=\begin{bmatrix}x_1\\y_1\\x_2\\y_2\\\vdots\\x_N\\y_N\end{bmatrix}$, where x_i, y_i are the 2D coordinates of a single station on the ground. N: number of stations
objectives:	$Cost(\vec{x},\vec{y},A)=\sum_{(i,j)\in A}\sqrt{(x_i-x_j)^2+(y_i-y_j)^2}$, A: arc set of $(\vec{x},\vec{y})$ where $u_{i,j}=\dfrac{x_i-x_j}{\lambda}, v_{i,j}=\dfrac{y_i-y_j}{\lambda}, \forall i,j,$ and λ is the observation wavelength $P(\vec{x},\vec{y},\lambda)=1-\dfrac{N_{uv\ actual}}{N_{uv}}$: uv density

3.5.2 Numerical experiments

Two basic observations regarding D-QMOO are derived from the numerical experiments: first, the algorithm produced better performing solutions than the ones previously known and second, it managed to discover specific geometric concepts when starting from a random population, without needing to be warm-started with seed designs.

Performance

Three separate instances of the array with a different number of stations (27, 60 and 99) were optimized. The number of stations was selected in order to compare with existing results (Bounova, de Weck 2005). D-QMOO's performance was very encouraging as it improved the existing designs. In terms of extreme solutions (anchor points) D-QMOO found better best-performance (uv density *P*) solutions than the existing results. It did as well or at times slightly worse at the other extreme of the Pareto front, the lowest cost solutions. The most significant improvement was around the middle of the Pareto front where designs that dominated the previously known ones by a significant margin were discovered, filling some large gaps in the front.

In Figure 14, Figure 15, and Figure 16 we can see a sample of the non-dominated front discovered by D-QMOO and compared to the existing solutions. The improvement effected by D-QMOO is apparent, especially along the middle of the Pareto front.

Optimization experiments with different algorithm parameters produced different results. It is interesting to observe in Figure 17 the various non-dominated fronts that D-QMOO produced by changing the total population size, front/cruft ratio and thinning method.

Concept discovery

The array configurations that solve this problem change through a number of different geometric concepts as one moves along the Pareto front. The best performance solutions are roughly in the shape of a circle. The lowest cost solutions are roughly in the shape of a 'Y' (hook- or VLA-shaped). The intermediate solutions take forms such as triangle, Releux triangle and log-spiral shape (see Bounova, de Weck 2005 for more details).

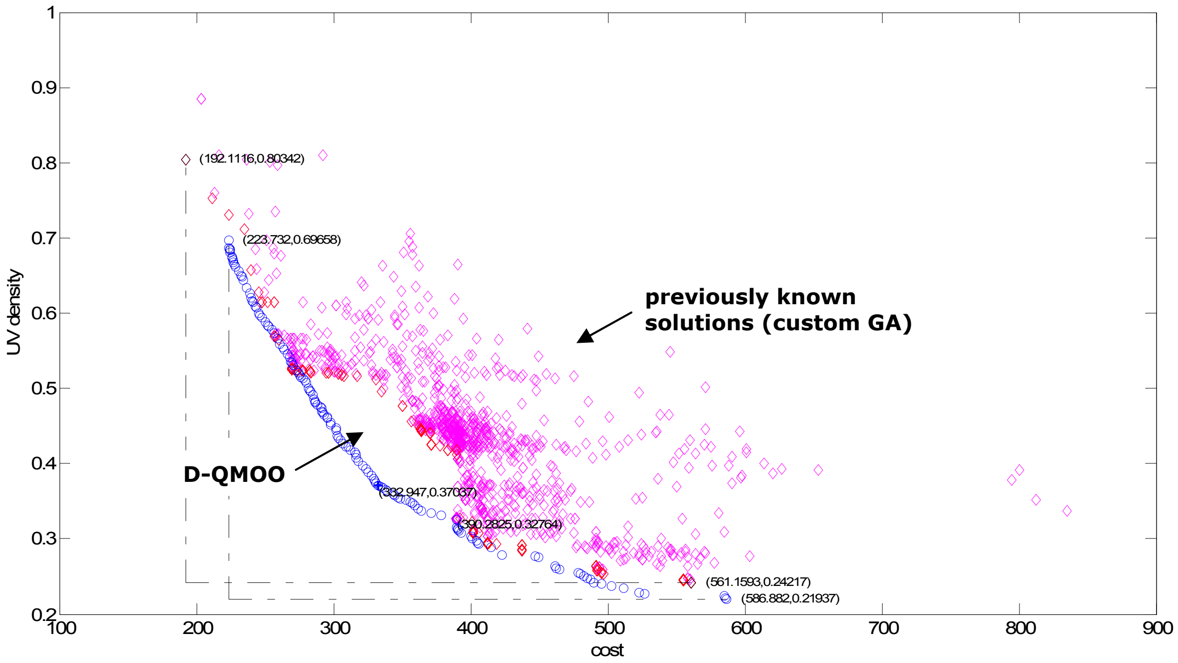


Figure 14. Non-dominated front comparison for 27 stations (blue circles: D-QMOO, red and purple diamonds: previously known solutions). The improvement in the middle region of the front can be seen.

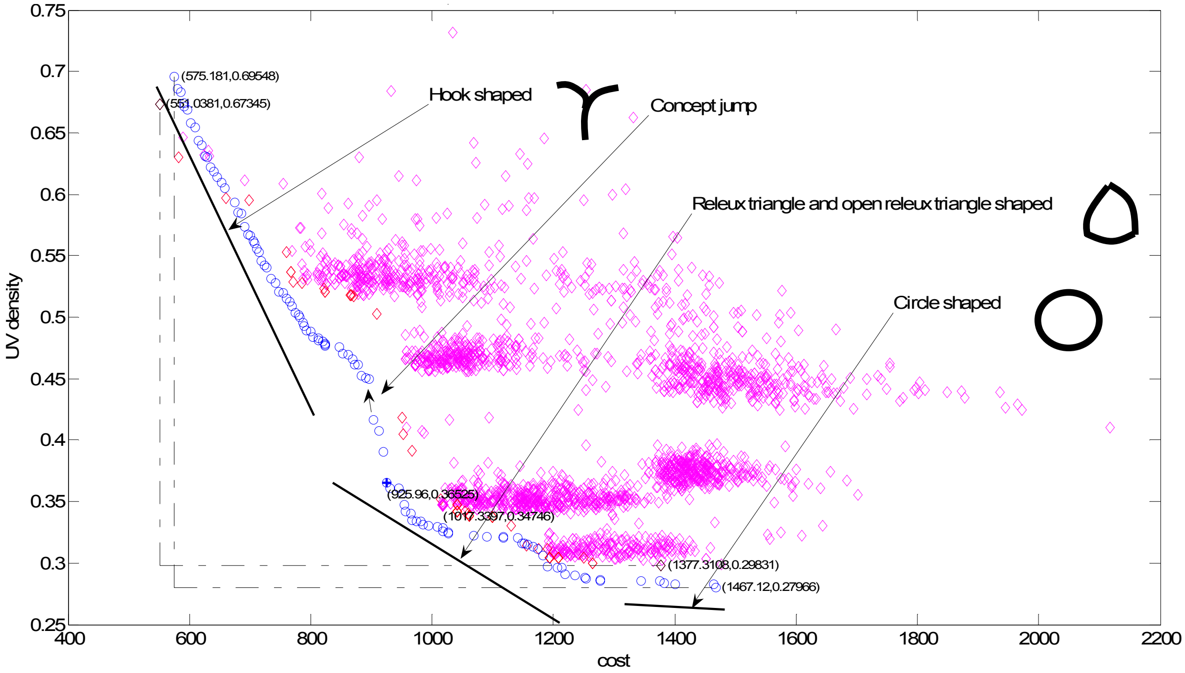


Figure 15. Non-dominated front comparison for 60 stations (blue circles: D-QMOO, red and purple diamonds: previously known solutions). The various geometric design concepts discovered by D-QMOO are noted.

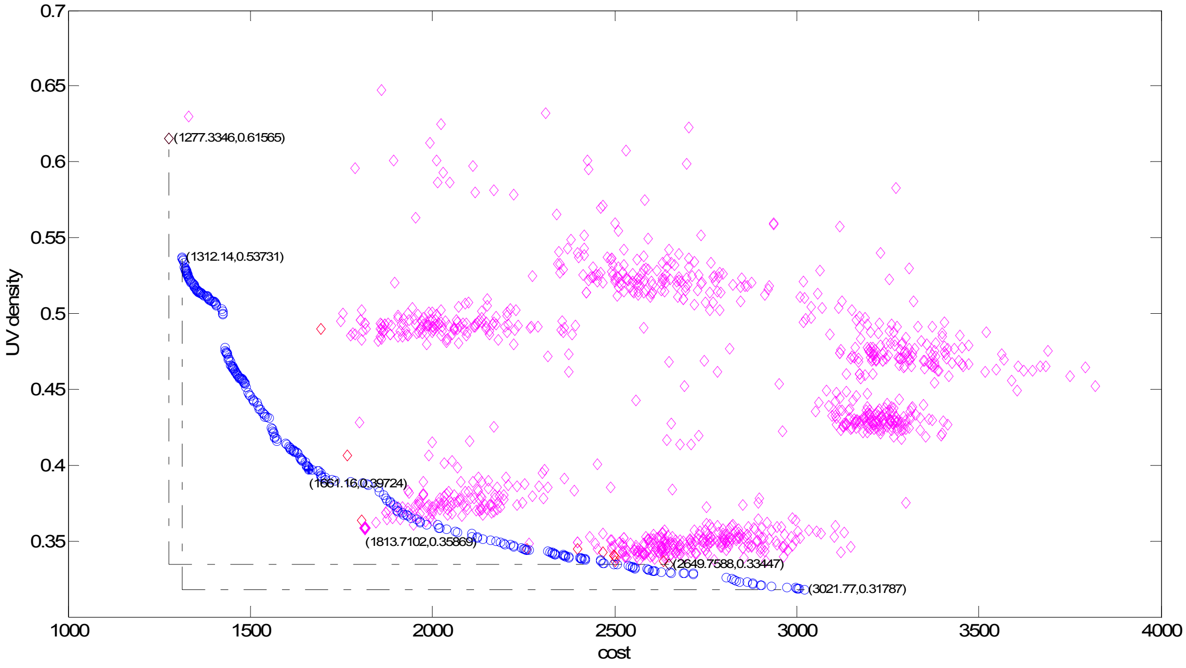


Figure 16. Non-dominated front comparison for 90 stations (blue circles: D-QMOO, red and purple diamonds: previously known solutions). In this case, the largest improvement happened in the high-uv density area (similar performance with much lower cost for D-QMOO).

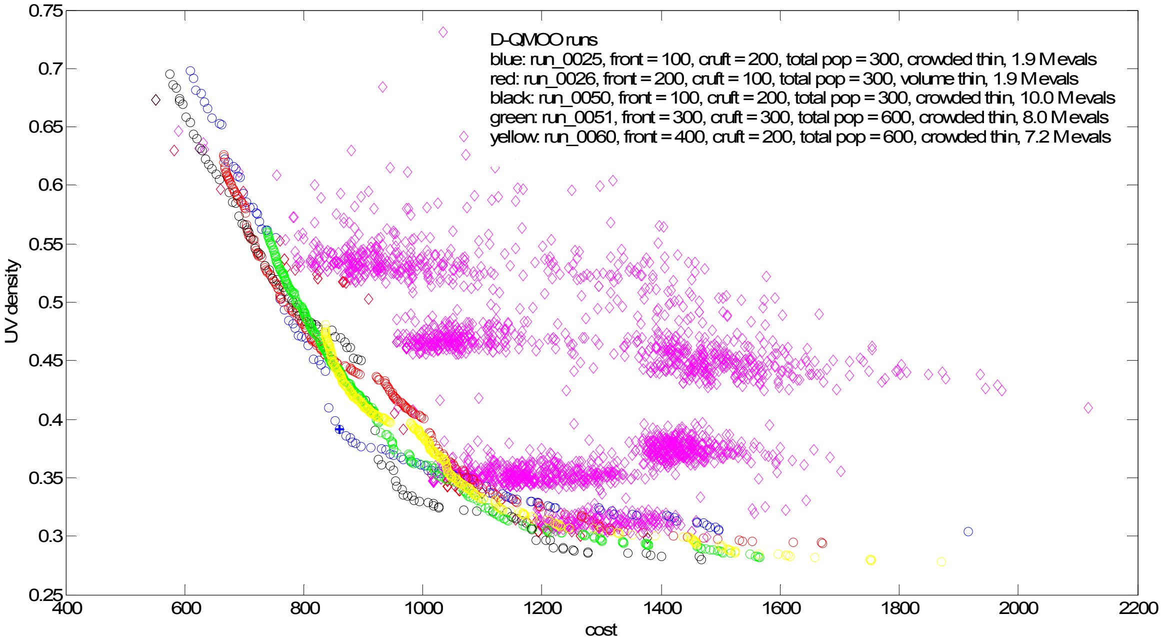


Figure 17. Non-dominated solution comparison for 60 stations. Several D-QMOO runs plotted together (with varying population, front/cruft ratio, and total evaluations). Selecting the non-dominated solutions from this group of runs yields a significantly improved front compared to the existing solution.

An important attribute of D-QMOO's performance is that it discovered most of these dominant geometric concepts starting from a random population of designs. Given the huge sampling space of the problem (in the order of 10^{200} - 10^{500} designs depending on the number of stations), this is a very encouraging fact. On the contrary, most of the previously existing results had been obtained by seeding (warm-starting) the custom-designed GA with the various geometric concepts and letting it refine them – the GA however was not able to discover these concepts from scratch. The drawback is that D-QMOO took more computational time to solve than the custom GA – each run took 10-20 hours with a million or more function evaluations, compared to minutes or hours for the GA. In Figure 15 we can see some of the design concepts annotated along the Pareto front for the 60 station problem. In Figure 18, we can see the minimum cost (circular) and maximum performance (VLA-shaped) designs for the 60 station problem, as discovered by D-QMOO. In Figure 19 we can see how D-QMOO discovers geometric concepts that resemble the seed designs used to warm-start the custom GA (a seed VLA solution is plotted against a VLA-like solution discovered by D-QMOO from a random initial population).

Grouping and population size

Due to the large size of the problem (54 to 198 variables, depending on the number of stations), design space grouping did not perform well (recall section 2.4). We found that the best solutions emerged using a large single-group population of 300 or more individuals, which discovered the various geometric design concepts along the Pareto front in a single run. The fact that the problem is multi-objective allowed the algorithm to obtain a large diversity in a single population without the help of grouping.

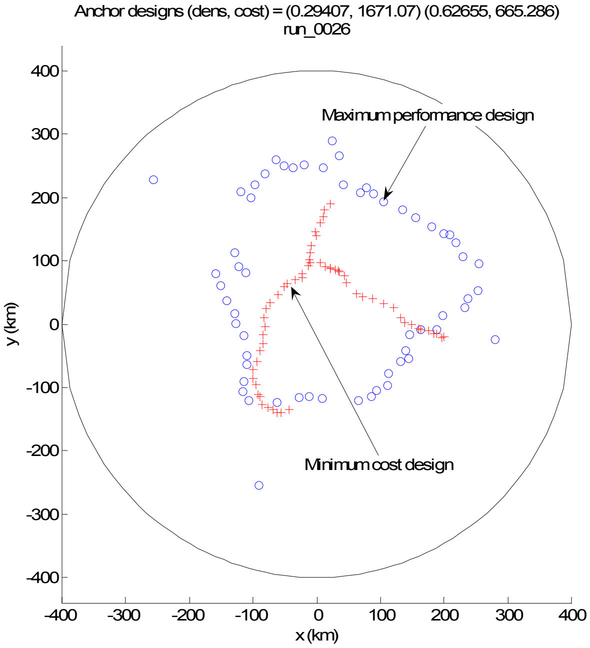


Figure 18. Concept discovery by D-QMOO: anchor designs for the 60 station problem. The minimum cost design is VLA-like, and the maximum performance design is roughly circular. Each symbol marks a radar antenna location.

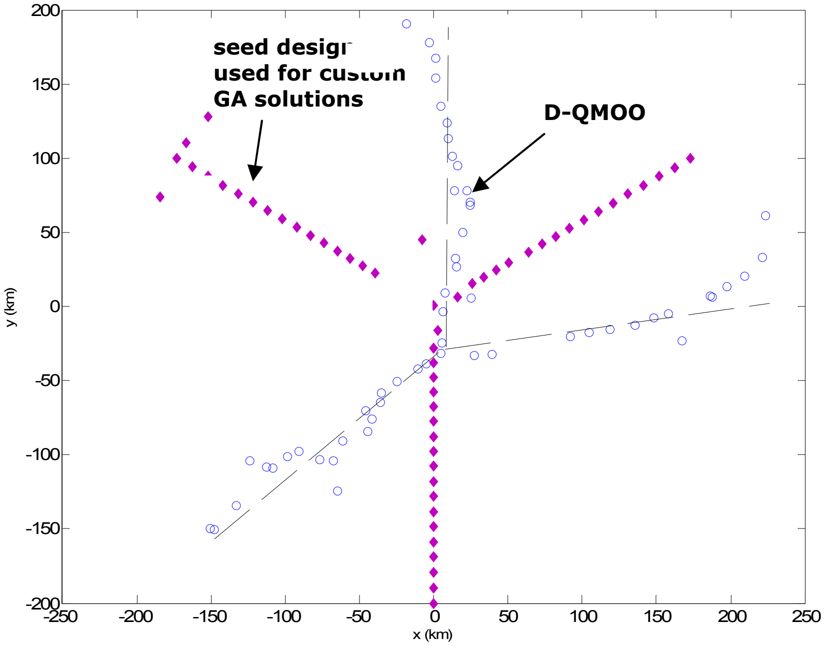


Figure 19. Concept discovery by D-QMOO in the 60 station problem: the seeded VLA shape from the original solution, and D-QMOO's VLA-like solution (which was discovered from a random initial population). Each symbol marks a radar antenna location. The rotation around the origin has no effect on performance – in fact, the problem is rotationally symmetric around the origin. It would be interesting to develop a design vector encoding which takes this rotational symmetry into account and considers designs which are rotated with respect to each other to be equivalent.

This is an instance of the *elitist diversity* that characterizes multi-objective problems, which was discussed in section 2.4. A large number of very different designs that all share the common virtue of being non-dominated provide the population with the diversity it needs in order to discover well-performing solutions. This form of diversity is different (though not necessarily better) from the largely random diversity of the cruft group.

3.6 Conclusion

In this chapter the subject of constraint handling is studied. After a brief general discussion on constraint handling with EAs, the development of constraint handling methods for D-QMOO is described. The method implemented initially is based on the weighted superiority of feasible points, a separation of objectives and constraints technique with elements from penalty methods. This method has the conceptual advantage of clearly discerning between feasible and infeasible individuals in a straightforward quantitative way. The combination of D-QMOO and WSFP is benchmarked against other constraint handling algorithms with encouraging results.

In order to provide greater independence from the need to tune penalty parameters, a second method is subsequently developed and implemented in D-QMOO. This method, the superiority of feasible points with initial feasibility exploration, works in a staged manner. It performs an initial search for the feasible regions of the design space, and then applies the WSFP method when it discovers them. This method helps retain D-QMOO's wide applicability since it requires minimal problem-specific tuning. The chapter concludes with a practical application on the design of a telescope array, where D-QMOO/WSFP-IFE is used to discover improved solutions to a static multi-objective problem.

4 The Combination of Anticipation and Diversity Preservation as a Time-Changing Evolutionary Optimization Concept

The solution of time-changing problems is the principal focus of this thesis. In this chapter, an evolutionary optimization concept for the solution of such problems is introduced.

As noted before, Evolutionary Algorithms are conceptually well-suited to handle problems that change in time since they simulate natural evolution, a process of continuous adaptation to a changing environment. Specific algorithmic architectures are needed however in order to create EAs that perform well in changing environments. In this chapter, some ideas to that end are presented and discussed. The proposed concept is based on the combination of *anticipation*, through the use of forecasting in order to predict the optimal solution's motion in time and increase the algorithm's performance, and *balance between population convergence and diversity*, which provides robustness to the time-changing algorithm.

Let us repeat the general statement of the problem solved by a non-stationary multi-objective Evolutionary Algorithm from definition (2.1):

$$\begin{array}{lll} \text{Find } \mathbf{x}^*_t \in \mathrm{X} \subseteq R^n & \text{which minimizes} & \mathbf{f}(\mathbf{x},t) = \left[f_1(\mathbf{x},t), ..., f_m(\mathbf{x},t) \right] \\ & \text{subject to} & l_i \le x_i \le u_i, i = 1,...,n \\ & & g_j(\mathbf{x},t) \le 0, \qquad j = 1,...,q \\ & & h_j(\mathbf{x},t) = 0, \qquad j = q+1,...,l \end{array} \tag{4.1}$$

where **x** is an *n*-dimensional design vector defined on a search space X and **f** is an *m*-dimensional objective function. In the multi-objective case (*m* > 1), $\mathbf{x}^*_t$ belongs to the set of Pareto-optimal solutions in the variable space, called Pareto optimal set (POS). The POS maps onto the Pareto optimal front (POF) of non-dominated points in the objective space[1]. The *g* and *h* functions express a total of *l* inequality and equality constraints. The temporal evolution may either be discrete or continuous. As was discussed in section 2.3 in this work we address discrete problems where time advances through a series of timesteps {…, *t*-2, *t*-1, *t*, *t*+1, …}. The objective landscape changes each time *t* advances to its next value. An amount of computational time is available during each timestep between changes in *t*. Since most real-world problems involve

[1] Pareto optimality, the Pareto optimal set, the Pareto optimal front and the concept of dominance were defined in section 2.3.

costly objective function evaluations we measure timestep length by the number of objective evaluations that can be performed, as will be discussed in section 4.1.

It must be noted here that problems such as the one in equation (4.1) are usually referred to as *dynamic* in evolutionary computation literature (hence the name of the D-QMOO algorithm). However, dynamic problems in the wider optimization community are ones in which the decision affects the next stages of the problem itself, warranting for example a dynamic programming approach. This is not the case for the problems treated in this work, nor for the problems treated in the vast majority of past work on evolutionary optimization for problems that change in time. For this reason, we refer to the problems of the form (4.1) treated here as *time-changing problems*. In the context of this work the two terms can be considered equivalent.

The goal of an evolutionary algorithm in a time-changing environment is to follow an optimal solution on a moving landscape[2] as closely as possible. This optimal solution may be a global optimum, a set of local optima, a non-dominated set, or a group of non-dominated sets, depending on the problem and on the desired outcome. For the practical purposes of algorithm design, this goal can be separated into two partial objectives as we can see in Figure 20: the algorithm must converge to the current optimal solution and when the landscape moves it must explore the design space for the optimal solution's new location. These two functions are normally competitive towards each other. This competition can be seen, for both static and time-changing problems, as a balance between *population convergence* and *diversity* or between *exploitation* and *exploration*. Handling the balance between these two functions will form one element of the concept proposed in this chapter.

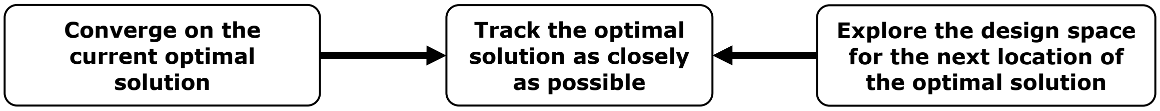

Converge on the current optimal solution
Track the optimal solution as closely as possible
Explore the design space for the next location of the optimal solution

Figure 20. The two basic goals of an evolutionary algorithm in time-changing environments.

In order to satisfy the goal of following the optimal solution as closely (and quickly) as possible, it makes sense to use all available information to the greatest extent. This brings us to the second main element of our proposed concept, which involves using past information in order to forecast the optimal solution's motion and improve the algorithm's performance.

Before we continue, it should be noted that while D-QMOO is the algorithm used as a development and testing platform, most of the ideas and architectures proposed in this and the next chapter are abstract enough to be applicable to other population based algorithms. This is one of the positive attributes of this work, and leaves a wide range of opportunities for future research.

The nature of changing environments will be briefly discussed next, followed by an overview of existing work in the area. The rest of the chapter holds a description of the basic concepts formulated in this work.

[2] The term *moving landscape* is used in a liberal way here; it might refer to a discontinuous or even disjoint design space, or to a mixed-integer problem for example. The term is kept because it provides an easy mental visualization of the problem.

4.1 The nature of time-changing environments

The nature of the objective landscape's motion is one of the most important elements in time-changing optimization. A useful set of categorizations for nonstationary environments has been given by Branke et al. (Branke 2002, Branke, Salihoğlu & Uyar 2005). Time-changing problems can be classified by their:

- *frequency of change* (measured by the number of objective function evaluations between objective changes, as we will see later).
- *severity of change* (measured by the distance between successive locations of the optimal solution).
- *predictability* (expressed by the potential for identification and exploitation of structure in the objective's temporal change pattern by a forecasting model).
- *periodicity* (expressed by the return of the optimal solution to past locations).

The first two categorizations are quantitative, measuring how fast the landscape changes and by how much. The other two are qualitative, assessing whether there is some form of structure in the objective landscape's motion.

Recall the moving landscape in Figure 3. The discovery of the successive locations of the POS can be seen in two ways: as a series of independent, unrelated static problems (one for each timestep *t*), or as a time-changing landscape with past and future.

Problems with no structure at all in their temporal change pattern can be solved by simply re-starting a static algorithm from scratch at each timestep. In such a case there is no benefit in the use of a time-changing algorithm, since no information is carried from one stage of the problem to the next and there is nothing for the algorithm to exploit in order to provide better performance than simply re-starting from scratch. Thus, the development of time-changing optimization algorithms is generally focused on problems with some amount of structure in their temporal pattern. The exploitation of predictability in the objective's motion will be of specific importance in this work.

Importance of the frequency of change

The frequency of change of a moving landscape, or *objective change frequency*, is a vital characteristic. This frequency is the speed with which the landscape changes. Since in this work we are treating discrete time problems, frequency could be measured for example by the number of objective changes per second.

We seek to solve the problem using an optimization algorithm and a numerical model of the system we are treating. Models of real-world problems tend to take a lot of computational time to evaluate – think for example of the time it takes to run a computational fluid dynamics simulation. Optimization processes in turn tend to require many simulations in order to evaluate the objective values of numerous candidate designs. Indeed, when large scale optimizations are run for complex industrial products or operations the limiting factor is usually the number of objective function evaluations that can be performed in the given time.

Objective change frequency therefore relates to the availability of computational resources and directly translates into a decision maker's potential for solving the problem, since it expresses the computational time available between changes in the objective. If infinite time was available, optimization algorithms would not be needed; an exhaustive search would eventually discover the optimum. Time is limited though, and for this reason an important measure of merit for an algorithm is the computational time it requires to find a solution of given performance, or the performance of the solution it discovers in a given length of time. In time-changing problems, computational efficiency is measured by the performance achieved as a function of objective change frequency. This frequency is therefore measured by the number of objective function evaluations that can be performed between objective changes. The exact measure used in this thesis is *objective function evaluations per timestep*[3], which is strictly an objective change *period* since a low value implies a high change frequency.

An important part of this work is devoted to computational efficiency: reducing the time the algorithm requires to produce well-performing solutions. In the context of this work, *algorithmic performance* can be defined as the quality of the solutions produced for a given objective change frequency or inversely the objective change frequency which allows a given quality of solutions.

4.2 Existing work on evolutionary optimization for time-changing environments

During the past 20 years there has been a significant research interest in the solution of dynamic problems with evolutionary algorithms. Helen Cobb (Cobb 1990, Cobb, Grefenstette 1993) introduces the technique of hypermutation, one of the first methods of diversity control for the solution of time-changing problems. This technique basically consists on increasing the mutation rate of a genetic algorithm when a change in the landscape arrives, in order to intensify diversity in the population and hence assist in the discovery of the new optimum. Grefenstette (Grefenstette 1992) elaborates on this method proposing a partial hypermutation, and the method of *random immigrants*. As early as 1987, David Goldberg and Robert Smith (Goldberg, Smith 1987) discussed the use of memory in the form of *diploidy* for the enhancement of an EA's performance in time-changing environments. Ramsey and Grefenstette (Ramsey, Grefenstette 1993) provide one of the first instances of explicit memory, using case-based reasoning to distinguish between environments. When a change arrives, individuals in the memory who have previously been successful in similar environments are reinserted in the population. Vavak and Fogarty (Vavak, Fogarty 1996) compare a generational and a steady-state EA in a dynamic environment, finding the steady-state algorithm superior. They also propose a Variable-Range Local Search (VLSI) technique for the solution of time-changing problems (Vavak, Fogarty & Jukes 1996, Vavak, Fogarty & Jukes 1997). This is a diversity control method aiming to match the level of diversity introduced into the population with the severity of environmental change, by increasing population diversity in a gradual way. Bäck (Bäck 1998) investigates the behavior of evolution strategies and evolutionary programming in time-dependent environments, and finds a lognormal self-adaptation rule to perform satisfactorily in the context of evolution strategies. Yamasaki (Yamasaki 2001) argues that in very fast-changing landscapes the non-stationary problem converges to a quasi-static multi-objective problem, where the fitness function value in consecutive past moments can used to create an objective vector and then treated with Pareto analysis. Ronnewinkel, Wilke and Martinetz (Ronnewinkel, Wilke & Martinetz 2001) study the mathematical behavior of simple genetic algorithms (no crossover) in time dependent environments with infinite populations. Optimal mutation rates are extracted and the formation of quasi-species is observed. Simões and Costa (Simoes, Costa 2002) study the various memory and diversity preservation techniques for handling dynamic problems, using the 0-1 Knapsack problem as a benchmark. They look into hypermutation, transformation and random immigrants, finding the first two more successful. Branke et al. (Branke et al. 2000) propose a multi-population technique called Self-Organizing Scouts (SOS) in order to track moving optima peaks in a dynamic environment. In SOS the main population searches for new peaks while peak neighborhoods brake off into new sub-populations. SOS is found to out-perform a standard

[3] This measure is established in literature, together with the equivalent measure of *generations per timestep* in the case of genetic algorithms. The importance of objective change frequency in time changing evolutionary optimization has also been discussed in literature (see for example De Jong 2006).

generational GA in the Moving Peaks benchmark problem. Morrison (Morrison 2004) provides a comprehensive study of the subject, focusing on the treatment of diversity, memory and change detection with ‘sentinel’ individuals. Bui et al. (Bui, Branke & Abbass 2004) use multi-objective methods to tackle single-objective time-changing problems. They try out different choices for the second (artificial) objective, but in general the second objective expresses diversity in different ways (three different distance metrics, age – favoring the oldest individual, inverted objective – favoring the worst individual, and random assignment).

Relatively little attention has been directed to multi-objective optimization in time-changing environments. The work by Marco Farina, Kalyanmoy Deb et al. (Deb, Rao & Karthik 2006, Farina, Deb & Amato 2004) is one of the very few such examples.

Two categories of methods

This brief literature survey by no means claims to be exhaustive[4]. It does however provide a general overview of existing work on time-changing evolutionary optimization, and makes it apparent that a large portion of the existing methods can be classified into one of two broad categories.

Relating to the discussion in this chapter’s introduction, the first category of methods addresses the control of the two basic functions of the algorithm’s population in a time-changing environment: convergence to the current global optimum, and exploration of the design space for the optimum’s next location or for new optima when the objective landscape changes. As noted before, the competitive interaction of these two functions can be viewed as a balance between population convergence and diversity. Some of the techniques developed in the past pertain to this balance and to the control of diversity. Hypermutation is such an example. The balance between convergence and diversity has also been explicitly examined as a multi-objective problem, with convergence to the current optimum being the first and population diversity being the second objective (Bui, Branke & Abbass 2004).

The second broad category of approaches is concerned with the exploitation of past information. This information is usually the location of past fit solutions, which might again become useful as the problem evolves. The various memory methods mentioned earlier are instances of such techniques. The main idea here, is to avoid having the algorithm do the same job more than once. Memory methods perform especially well in periodic problems where the optimum returns to previous locations which have been stored in the memory, and the algorithm does not need to spend time discovering them. The value of past information, such as the position of prior optima, has also been demonstrated by Branke et al. (Branke, Salihoğlu & Uyar 2005) in their discussion of changing environments. Although memory methods have been designed in many different forms and one should not attempt to generalize on the way they work, they have been broadly classified into *implicit* and *explicit* methods. Implicit memory methods are those that utilize an indirect way of storing design vector information. Diploidy for example is an instance of an implicit method, where the individual’s chromosome has a second set of inert genes that becomes active if the environment encourages it. Explicit memory methods on the other hand store the location of past fit solutions in a dedicated database which is external to the population; these solutions are recalled and re-inserted in the population when the shape of the objective landscape favors them.

4.3 Arriving at the proposed concept

The approach proposed in this work aims at reaping the benefits of both classes of techniques just

[4] The reader is referred to the comprehensive surveys by Branke and Jin (Branke 1999a, Branke 2001, Jin, Branke 2005) and to the book by Branke (Branke 2002) for a more detailed report on the area.

described, diversity control and exploitation of past information. First, it takes advantage of any predictability present in the objective's temporal change pattern to accelerate convergence and improve the performance of the time-changing optimization algorithm. Second, it controls the balance between population convergence and diversity in order to handle any unpredictable change and be able to discover the optimum even if it moves in an unstructured way.

The technique of exploiting predictability in the objective's temporal change pattern is called *Feed-forward Prediction Strategy (FPS)*. The Feed-forward Prediction Strategy consists of using the history of the optimum's path over time to predict its location in the next timestep, and it will be described in detail in the next section. As we will see, the FPS provides a way of using the solution information available from the past as memory methods do, but with potentially better performance.

As we can see in Figure 21, the FPS is not intended to be completely self-sufficient as a time-changing optimization technique. This is because the prediction might not be successful – the objective might be moving in an unpredictable way, or in a way that cannot be identified by the forecasting model. For this reason a convergence-diversity balance technique is used in parallel with the FPS so that the new optimum can be discovered even if the forecast is not successful. In the core of this convergence-diversity balance method lies the existence of two separate groups in the population, one of which seeks to converge to the current optimum by employing elitism while the other explores the design space by enforcing different forms of diversity. These are the *front* and *cruft* groups respectively, as defined in section 2.2. This method, which is similar to other diversity control methods proposed in the past, will also be described in detail later.

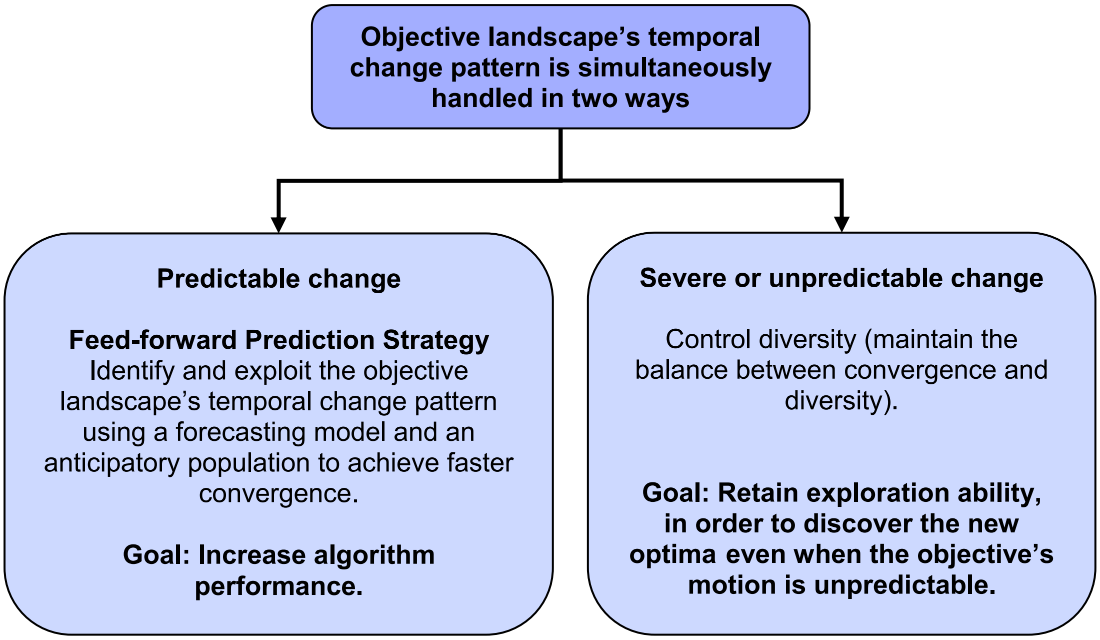

Objective landscape's temporal change pattern is simultaneously handled in two ways
Predictable change
Feed-forward Prediction Strategy
Identify and exploit the objective landscape's temporal change pattern using a forecasting model and an anticipatory population to achieve faster convergence.
Goal: Increase algorithm performance.
Severe or unpredictable change
Control diversity (maintain the balance between convergence and diversity).
Goal: Retain exploration ability, in order to discover the new optima even when the objective's motion is unpredictable.

Figure 21. Proposed concept for the solution of time-changing problems with population-based algorithms.

This combination of approaches lies at the core of this work. The goal is to achieve good algorithmic performance by making the best possible use of the information obtained from solving the problem up to the current timestep, and simultaneously to achieve robustness by providing the algorithm with an exploration ability independent of any pattern or predictability in the objective's motion.

4.4 Anticipation and convergence-diversity balance

In this section a detailed description of the time-changing optimization concept is given, consisting of the anticipation (Feed-Forward Prediction Strategy) and convergence-diversity balance methods.

4.4.1 Feed-Forward Prediction Strategy.

According to the Feed-Forward Prediction Strategy, the sequence of the past optimal solution locations is cast in the form of a time-series. This time series is used as input to a forecasting model, which produces an estimate for the location of the next optimum. This estimate is used to create an anticipatory set of individuals. This set is called *anticipatory population* (or *prediction set*), and the individuals consisting it *anticipatory individuals* (or *prediction individuals*). As soon as the next timestep arrives and the objective function changes, the anticipatory population (AP) is inserted into the algorithm's general population. If the prediction is successful then the anticipatory individuals are close to the next optimal solution. This provides a sense of direction, assisting the algorithm in discovering the new optimal solution quickly. Bosman (Bosman 2005) states the importance of learning and of preparing the population in anticipation of the objective's future behavior. A simplified illustration of this concept for a two dimensional variable space is shown in the sketch of Figure 22.

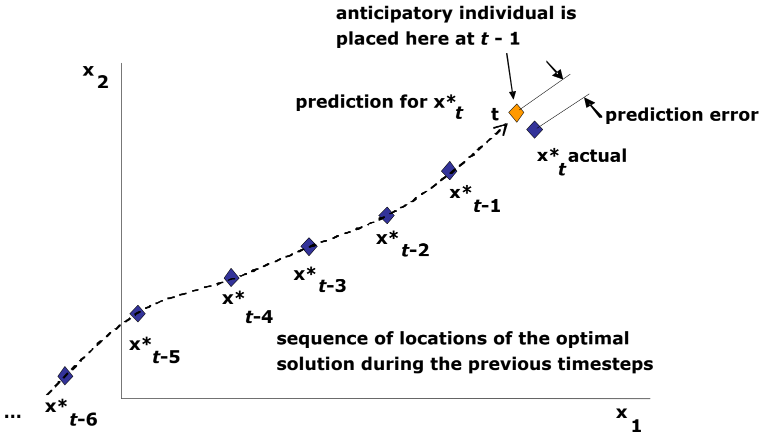


Figure 22. The sequence of past optima is used to create a prediction for the next timestep *t*.

The FPS is outlined in Figure 23 and in the following pseudo-code.

```
At the end of timestep $t$-1:
1. Using the time series of the current and past locations of the
   optimal solution {$\mathbf{x}^*_{t-1}$, $\mathbf{x}^*_{t-2}$, ...} and a forecasting method,
```

create a prediction for the location of the next optimal solution $\mathbf{x}^*_t$.
2. Using the prediction a group of individuals (the anticipatory population) is created.

At timestep *t*:

3. The anticipatory population created in *t*-1 is inserted into the population.
4. The optimization algorithm runs for the available function evaluations to discover the new optimum $\mathbf{x}^*_t$.
5. At the end of the timestep the best discovered solution is stored as the approximation of the optimum at time *t*, $\mathbf{x}^*_t$, and the time series is updated.
6. Return to 1.

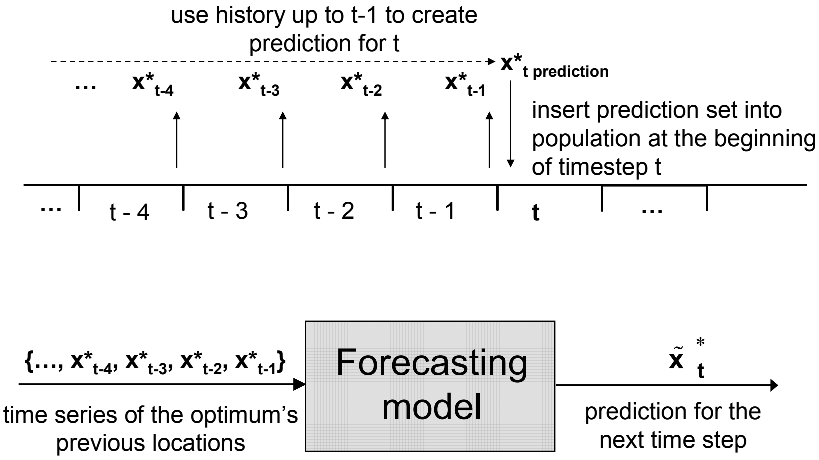

use history up to t-1 to create prediction for t
x*t prediction
...
x*t-4
x*t-3
x*t-2
x*t-1
insert prediction set into population at the beginning of timestep t
...
t - 4
t - 3
t - 2
t - 1
t
...
{..., x*t-4, x*t-3, x*t-2, x*t-1}
time series of the optimum's previous locations
Forecasting model
x̃*t
prediction for the next time step

Figure 23. Outline of the Feed-forward Prediction Strategy.

FPS and the source of the problem's variation in time

As we can see in equation (4.1), the objective landscape may change in time due to changes in the objective vector itself, in the constraint functions which affect the shape of the feasible region, or both. Accordingly, the optimal solution's motion can stem from any of these causes – for example, if the optimal solution is lying at a negative peak of the landscape it will move if the peak moves, and if it is located at the edge of the feasible region it will move if the active constraint functions change.

It is important to clarify that the Feed-forward Prediction Strategy exploits predictability in the motion of the optimal solution. It does not matter whether this motion is caused by variations in the objective vector, in the constraint functions, or both. What matters is whether the forecasting model can identify some kind of structure in that motion and predict the optimal solution's next location with enough accuracy.

4.4.2 Maintaining the balance between convergence and diversity

Maintaining a balance between convergence to the current optimal solution and population diversity provides a robust exploration ability that allows the discovery of each timestep's optimal solution, even in the case when this solution has moved in a severe or unpredictable way from its previous location. This balance ensures the preservation of diversity; it is a widely used concept in both static and time-changing evolutionary optimization (see for example section 4.2 on existing work) and can be accomplished in several ways. The concept proposed here does not necessarily call for a specific technique – any convergence-diversity balance method can potentially be used in order to fulfill this role. Here the method implemented in the D-QMOO algorithm will be described.

As described in section 2.4 the algorithm's population is composed of two parts: a group of non-dominated individuals (the *front*) and a group of dominated individuals (the *cruft*). The front's function is to preserve elitism and converge to the current Pareto optimal front. Dominated solutions enter the cruft, whose function is to preserve diversity. This subgroup is formed using the individuals' age and variable space crowding as criteria instead of dominance rank, making it a diverse sub-population that explores the design space for new optima. Overall the convergence-diversity balance mechanism is provided by the existence of these two population groups, by the control of their relative size, and by the choice of criteria used to thin the cruft group as will be described later.

It must be noted here that although in a real-valued problem diversity in a population of designs is defined through the closeness of individuals in Euclidean space, in literature there is a number of different metrics that quantify it. Toffolo and Benini (Toffolo, Benini 2003) and Bui et al. (Bui, Branke & Abbass 2004) provide some examples. In this work, the Distance to Closest Neighbor (DCN) metric is used, as defined by Bui et al. (Bui, Branke & Abbass 2004).

Control of the cruft is achieved through the *thinning criterion*, the method used to select which individuals to eliminate when the cruft has reached the maximum allowed size. The thinning criteria are two: age, where oldest individuals are eliminated or crowding, where individuals in more crowded areas are eliminated. Each time an individual must be eliminated, a Bernoulli (biased coin flip) trial is executed with $P_{birthday}$ probability of eliminating the individual by age and $1\text{-}P_{birthday}$ probability of eliminating it by crowding (Figure 24).

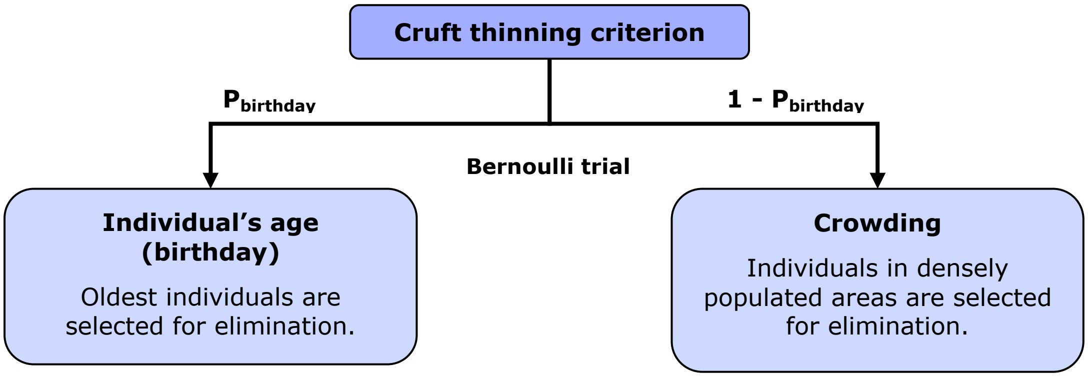

Cruft thinning criterion
$P_{birthday}$
$1 - P_{birthday}$
Bernoulli trial
Individual's age (birthday)
Oldest individuals are selected for elimination.
Crowding
Individuals in densely populated areas are selected for elimination.

Figure 24. The cruft thinning criterion is selected by a Bernoulli (biased coin flip) trial. The externally defined $P_{birthday}$ probability provides bias towards one or the other criterion.

Elimination by age has the logic of giving each individual an equal chance of contributing to the solution, by ensuring all individuals have a similar lifespan. Elimination by crowding provides a direct way of spreading the cruft population over the whole design space. By tuning the $P_{birthday}$ parameter, elimination can be biased towards one or the other criterion.

There is a number of ways to control the selection of cruft thinning method. The simplest is to keep a constant $P_{birthday}$ value. Another way is to use an open-loop control technique like the one shown in Figure 25. Since spatial diversity is needed most right after the landscape has moved, $P_{birthday}$ is held constant at a 'steady-state' value during part of each timestep. Then when a change in the objective arrives, $P_{birthday}$ drops to a lower value favoring the crowding criterion and hence spreading the cruft individuals over the design space. $P_{birthday}$ then rises again to its steady state value following some form of curve (linear, exponential etc.). The steady-state value is usually in the order of 0.50, and the minimum in the order of 0.10. This is an open-loop method since it does not use any feedback from the state of the population in order to regulate the thinning criterion. Other methods such as closed-loop control where the population convergence state controls the thinning criterion can also be used. However, in most of the numerical experiments in this work the $P_{birthday}$ value was kept constant at 0.50 throughout the whole run, in order to make it easier to discern the effect of other algorithmic options.

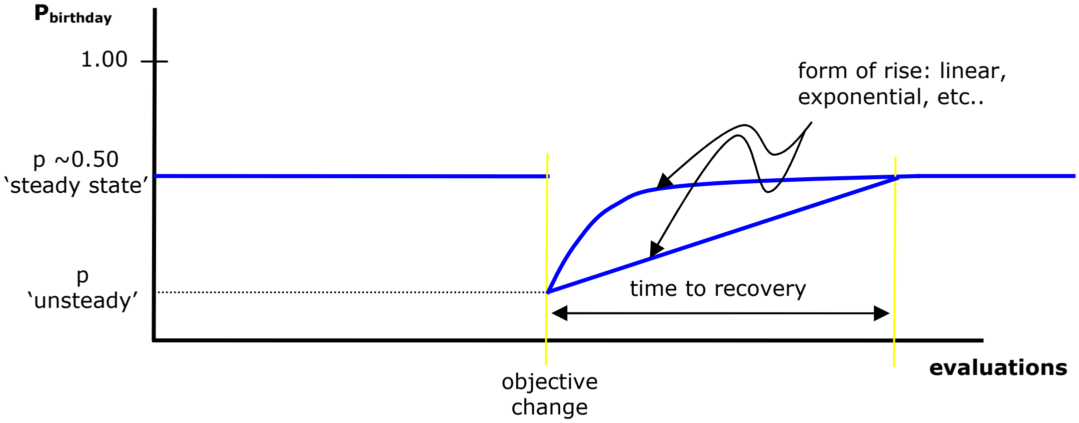


Figure 25. Open-loop regulation of the $P_{birthday}$ parameter.

4.4.3 Population subgroups

The algorithm's population is therefore normally composed of the front and cruft subgroups. When the FPS is used, a third group in the form of the anticipatory population joins the rest of the population, placing a team of individuals in the forecasted neighborhood of the next optimum in order to achieve faster discovery and convergence. Hence, the population at the beginning of a timestep is composed of three subgroups, as can be seen in Figure 26.

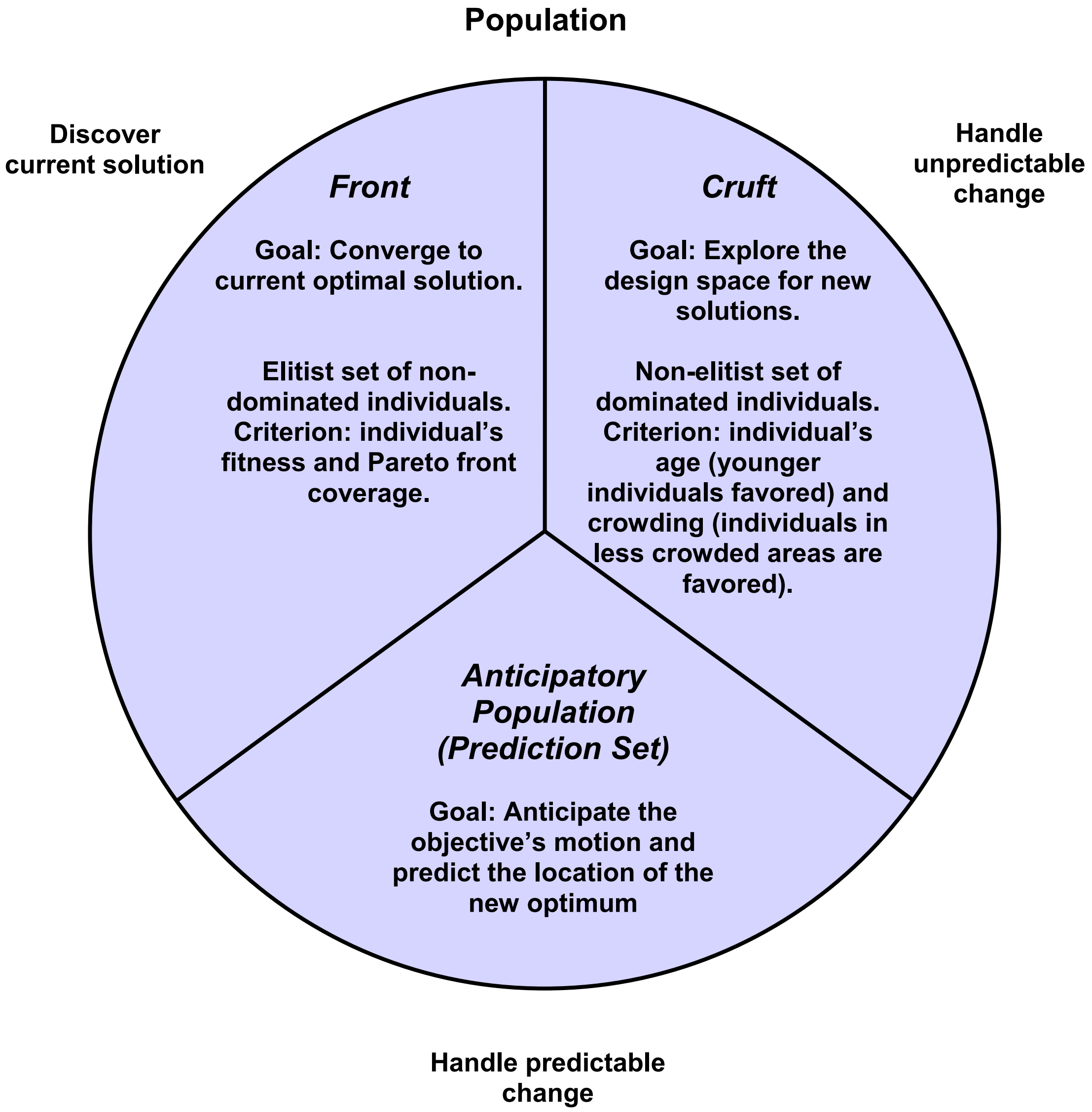

Population
Discover current solution
Handle unpredictable change
Front
Goal: Converge to current optimal solution.
Elitist set of non-dominated individuals. Criterion: individual's fitness and Pareto front coverage.
Cruft
Goal: Explore the design space for new solutions.
Non-elitist set of dominated individuals. Criterion: individual's age (younger individuals favored) and crowding (individuals in less crowded areas are favored).
Anticipatory Population (Prediction Set)
Goal: Anticipate the objective's motion and predict the location of the new optimum
Handle predictable change

Figure 26. Population subgroups.

4.4.4 Advantages of the proposed concept

The proposed concept has two core functions that work in parallel for the solution of dynamic problems.

First, the existence of a convergence-diversity balance mechanism can potentially handle unpredictable changes and aid in the discovery of moving optima, even if the forecast contains a large error or if different optima suddenly appear in completely different locations in the variable space. Conceptually, this function encompasses the advantages of approaches such as hypermutation (Cobb 1990) or of other convergence/diversity balance techniques (Bui, Branke & Abbass 2004, Branke et al. 2000).

Second, if there is predictability in the objective's change pattern, the Feed-forward Prediction Strategy places the prediction set in the neighborhood of the forecast and thus points the rest of the population in the right direction for the next optimum. Since the FPS does this by using past information in order to obtain a forecast, it offers in an abstract way some of the benefits of memory methods. However there is an important distinction between memory methods and the FPS that can be deduced from Figure 22. If we are at *t*-1 and the objective function is about to change making $\mathbf{x}^*_{\mathrm{t}}$ the optimal solution, the FPS will seed the population with anticipatory individuals such as the one shown, using the forecasting model to extrapolate an estimate from the time series $\{..., \mathbf{x}^*_{t-3}, \mathbf{x}^*_{t-2}, \mathbf{x}^*_{t-1}\}$. If there is an amount of predictability in the temporal change pattern the anticipatory individuals will be close to $\mathbf{x}^*_t$ and will help discover it quickly. On the other hand, a memory method would only have the option of recalling one or more of the past fit solutions $\{\mathbf{x}^*_{t-2}, \mathbf{x}^*_{t-3}, \mathbf{x}^*_{t-4}, ...\}$ which, in this case, would not benefit the solution at *t*.

4.4.5 The anticipatory population

In a practical implementation of the FPS the question that arises is *which part of the solution to include in the anticipatory population?* In the context of a population-based algorithm, the question turns to *which individuals should be tracked and their future location predicted?* The question that follows directly is, *given a forecast for the location of the individuals, how should the anticipatory population be created? Where should the anticipatory individuals be placed?*

These questions pertain to the topology of the anticipatory population which is an important issue and will be addressed in detail in the following chapter. Here, a brief outline of the basic elements regarding the creation of the anticipatory population will be given.

It should be noted that the terms *anticipatory population* and *prediction set* are used interchangeably in this work, and can be considered equivalent.

Selection of prediction individuals in single-objective problems

The main factor behind selecting which part of the solution to track and predict is the user's desired outcome from the optimization process. In a single-objective problem, the user might seek for instance to discover the global optimum, a set of local optima, or a set of designs satisfying a specific performance requirement. Selecting the members of the anticipatory population is straightforward here. In the first case the anticipatory population could be composed, for example, of the best discovered solution, which is the individual with an objective value closest to the global optimum.

Selection of prediction individuals in multi-objective problems

In a multi-objective problem the solution is a set of non-dominated designs. In this case it is not straightforward how to select individuals in order to form the anticipatory population. In theory one would wish to predict the motion of the whole non-dominated set in the variable space. However this would probably be computationally expensive since a forecasting process must be performed for each prediction point. Hence, a limited number of prediction points need to be selected from the non-dominated set.

A rational way to make this selection is to pick points that are topologically distinct. A good example of such points are the extreme solutions on the Pareto front – the best solution for each of the objectives. For example two anchor points exist in the case of a two objective problem (Figure 27). The anchor points lie at the edges of the Pareto front in the objective space, and hence are relatively easy to single out from the rest of the population.

The issue of selecting prediction individuals in order to track the non-dominated front will be discussed in detail in the following chapter where a more comprehensive coverage of the Pareto front by the prediction individuals will be sought. In the numerical experiments presented in this chapter, a prediction set consisting of the two anchor points[5] as shown in Figure 27 is used.

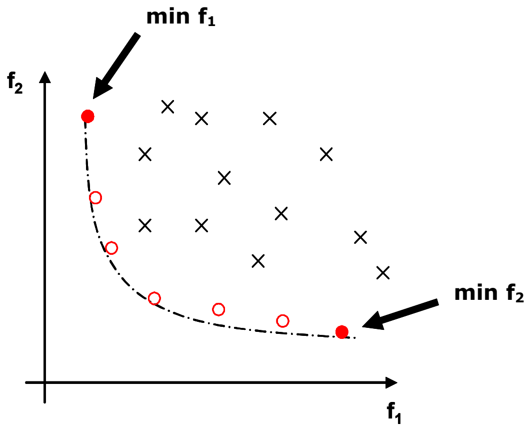


Figure 27. Anchor points.

Populating the neighborhood of the forecast and creating the anticipatory population

Assuming we have selected which points to track and predict, and that we have created a forecast for their location in the design space, we need to decide where exactly to place the anticipatory individuals. The simplest answer to this problem is to place a single individual on each forecast location. However more elaborate methods can be used to provide a comprehensive coverage of the predicted optimum's area. The logic behind this is that there is usually an amount of error in the forecast, as will be discussed in section 4.4.7, and placing a number of individuals around the forecast location might assist in finding the actual optimum. For example a hypercube of individuals can be formed around the predicted location of the next optimum, and information on the error and the confidence margins of the forecast can be used to dimension the hypercube so that there is a high likelihood it includes the actual optimum. These techniques will be discussed in the next chapter. The anticipatory population in this chapter's results consists of a single individual on the forecast coordinates.

It becomes apparent that the anticipatory population is in general much smaller than the total population of the algorithm, which in usually in the order of 100 individuals. For this reason, the evolutionary algorithm must exhibit some form of *elitism* in order to encourage the survival of the anticipatory individuals. This issue is discussed in more detail in section 5.3.

4.4.6 Forecasting model

In theory any mathematical process that produces a one-step-ahead estimate for the time series of the best solution's location could serve as a forecasting model for the Feed-forward Prediction Strategy. The choice of forecasting model depends on the nature of the time-changing problem and the intended range of problems the algorithm will be called to solve. For example if an algorithm is focused on a specific problem, it is also likely that a specific forecasting method exists which performs well on this type of problem.

[5] Anchor points in multi-objective problems are defined in section 2.3.

On the other hand if the algorithm is intended to be as widely applicable as possible then a robust, widely applicable forecasting model should be used. Such a model might not provide the best possible performance but it will be able to handle a wide range of problems. In this case one needs to use a forecasting model which requires as few assumptions as possible on the nature of the objective's temporal change pattern[6]. One also needs to assume that the only information available to the forecasting model is the location of the previous timesteps' best discovered solutions.

Several kinds of forecasting models have been used throughout this work, in order to explore their merits and shortcomings and to address different time-changing problems. In the numerical experiments in this chapter a stochastic time series model is employed. These models are products of time series analysis and forecasting methods which can be found in statistics and econometrics (Armstrong 2001). They include Autoregressive (AR) and Moving Average (MA) techniques (often called Box-Jenkins methods – see Box, Jenkins & Reinsel 1994) and a number of their variants (Autoregressive Integrated Moving Average models (ARIMA), Vector (multivariate) Autoregressive models (VAR) etc.). Time series methods have been developed extensively (see for example Hamilton 1994). They are intended to be used with random processes and hence, in our context, they can be applied to stochastic optimization problems. A disadvantage of time series methods is that they require certain conditions to be met regarding the statistical characteristics of the data – for example, autoregressive models require the data to be mean and covariance-stationary (Akaike, Nakagawa 1972, Hamilton 1994). However it is possible to identify and treat the cases when these conditions do not hold (Harris, Sollis 2003, Hamilton 1994).

In addition to econometric methods, other forecasting model candidates range from a simple polynomial extrapolation to artificial neural networks. We will have the chance to see the application of different models later in this work.

Apart from forecasting models, different approaches for the creation of an anticipatory population can be used alternatively. For example, if the problem allows, a computationally inexpensive optimization process (such as a single-objective linear program) can be applied to discover parts of the solution, which can then be used to create the anticipatory population. Such an approach can be used in cases where forecasting is very difficult and inaccurate, such as the portfolio optimization problem in chapter 6.

4.4.7 The two sources of prediction error

Ideally, the forecasting model would predict the exact location of the each member of the anticipatory population (the global optimum or a point on the Pareto optimal set, for instance) for the next timestep. However the actual prediction will have an amount of error in it as the sketch on Figure 22 shows. There are two main sources for this error (Figure 28):

Accuracy of the optimal solution's history

The first source of error is due to the fact that the algorithm might have not converged fully in the past timesteps and hence the best discovered solutions might not coincide with the actual past optima. As we have described earlier (see Figure 23), the forecasting model's input is the time series of the optimal solution's location during the previous timesteps. The actual optimum is of course unknown – the available data is the best discovered solution at each timestep. If the evolutionary algorithm has not converged properly, then the time series of the best discovered solutions does not coincide with the time series of the true optima, which induces error in the

[6] Such assumptions could demand for example the objective to move in a linear or periodic manner.

input of the forecasting model.

Accuracy of the forecasting model

The second source of error is the performance of the forecasting model itself, since no forecasting method is perfect even if the input data is perfectly accurate. There is always a possibility that the forecasting model will produce an inaccurate prediction – this depends on the nature of the problem and the type and quality of the model. For example, if the problem is linear and deterministic (i.e. the global optimum traces a straight line segment in the variable space) and we are using a first-order polynomial extrapolation as a forecasting model, then there will be no error in the forecast of the optimum's location in the next timestep. If, on the other hand, the environment's temporal change pattern is stochastic then the forecast will normally not coincide with the next timestep's optimum.

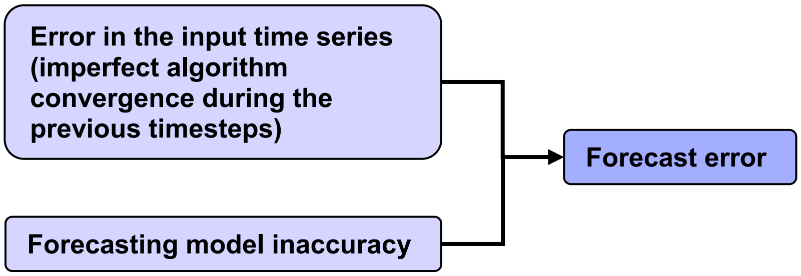

Error in the input time series (imperfect algorithm convergence during the previous timesteps)
Forecasting model inaccuracy
Forecast error

Figure 28. Sources of forecast error.

4.5 Numerical experiments

Results from the first implementation of our proposed concept for time-changing evolutionary optimization are shown in this section. The goal of this set of experiments was first, to show that our proposed concept can handle time-changing problems and second, to show that the Feed-forward Prediction Strategy results in a significant increase of performance for a time-changing algorithm. Before we continue to experimental results, some implementation details such as the forecasting model will be discussed.

4.5.1 Forecasting model

The forecasting models used with the FPS in this set of experiments are Autoregressive (AR) models. These models do not require any external data or problem-specific knowledge in order to produce a forecast – they only utilize the given time history.

In this work the AR models created by Schneider and Neumaier (Schneider, Neumaier 2001) are employed. Two different ways of predicting an individual's location are used:

- The whole design vector is treated as a vector time series and a multivariate vector autoregressive (VAR) model is used for forecasting. In this case, the forecast for the t – timestep's design vector $\mathbf{x}_t$ is (Schneider, Neumaier 2001):

$$\tilde{\mathbf{x}}_t = \mathbf{w} + \sum_{i=1}^{p} \mathbf{A}_i \cdot \mathbf{x}_{t-i} \tag{4.2}$$

where $\mathbf{x}$ is the n×1 design vector, $\mathbf{A}_i$ are the $n \times n$ autoregressive coefficient matrices and p is the order of the autoregressive model, selected by Schwarz's Bayesian Criterion (Schneider, Neumaier 2001). The $n \times 1$ vector $\mathbf{w}$ is an intercept term which allows for a non-zero mean of the time series. The algorithm using this method is called D-QMOO/VAR.

- Each design variable is treated as a single time series and a univariate autoregressive model is applied for the forecasting of each variable $\mathbf{x}_{j,t}$ of the design vector separately:

$$\tilde{x}_{j,t} = w_j + \sum_{i=1}^{p} a_{j,i} \cdot x_{j,t-i}, \text{ for } j = 1,...,n \qquad (4.3)$$

  As before but in scalar form, $a_{j,i}$ are the autoregressive coefficients and w_j is the intercept term. The algorithm using this method is called D-QMOO/AR.

In total three versions of the dynamic optimization algorithm are tested and compared: the multivariate D-QMOO/VAR, the univariate D-QMOO/AR, and the D-QMOO algorithm without the Feed-forward Prediction Strategy.

Initially the algorithm is run for a fixed number of timesteps (training period), as shown in the sketch of Figure 29. In this section's experiments the training period is 100 timesteps. The sequence of best discovered solutions at each timestep is collected into a time series and the AR model is fitted (the $\mathbf{A}_i$ or $\alpha_{j,i}$ autoregressive coefficients are defined). Subsequently at the end of each timestep the best discovered solution is added to the time series, and the AR model is used to forecast the location of the optimal solution for the next timestep.

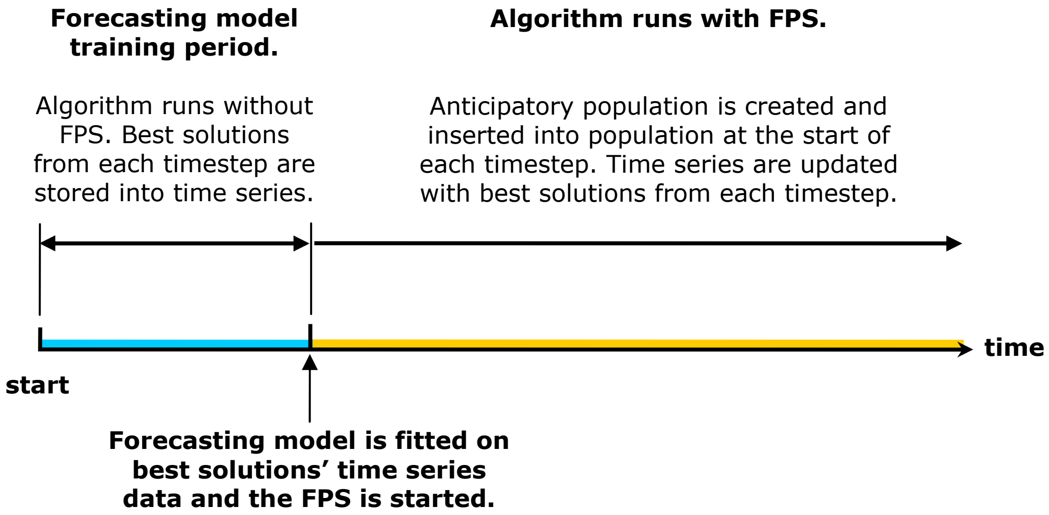

Forecasting model training period.
Algorithm runs without FPS. Best solutions from each timestep are stored into time series.
Algorithm runs with FPS.
Anticipatory population is created and inserted into population at the start of each timestep. Time series are updated with best solutions from each timestep.
time
start
Forecasting model is fitted on best solutions' time series data and the FPS is started.

Figure 29. Training and running stages of the FPS.

4.5.2 Anticipatory population

The results of this section are from a two-objective benchmark problem. As mentioned in section 4.4.5, one of the simplest possible topologies was used. The prediction set consists of the forecasts for the two Pareto front anchor points, as shown in Figure 27. Hence two time series are stored, one for each anchor point. The two forecasting configurations described in section 4.5.1 are used. Hence, either an AR model is fitted onto the anchor point's whole design vector time series, or onto each separate design variable's time series. These models are used in order to get a prediction for the location of each anchor point in the next timestep. Two corresponding individuals are created forming the prediction set. This anticipatory population is inserted into the general population as soon as the objective changes onto the next timestep. If the prediction is successful, then the anticipatory individuals are close to the actual anchor points, helping the population converge faster to the new Pareto front.

4.5.3 FDA1 benchmark problem

FDA1 is the benchmark problem used for this set of results. It is a time-changing two-objective problem from the dynamic multi-objective suite by Farina, Deb and Amato (Farina, Deb & Amato 2004). The Pareto optimal set is harmonically moving in time between two extremes for all but the first of the design variables (see Figure 30).

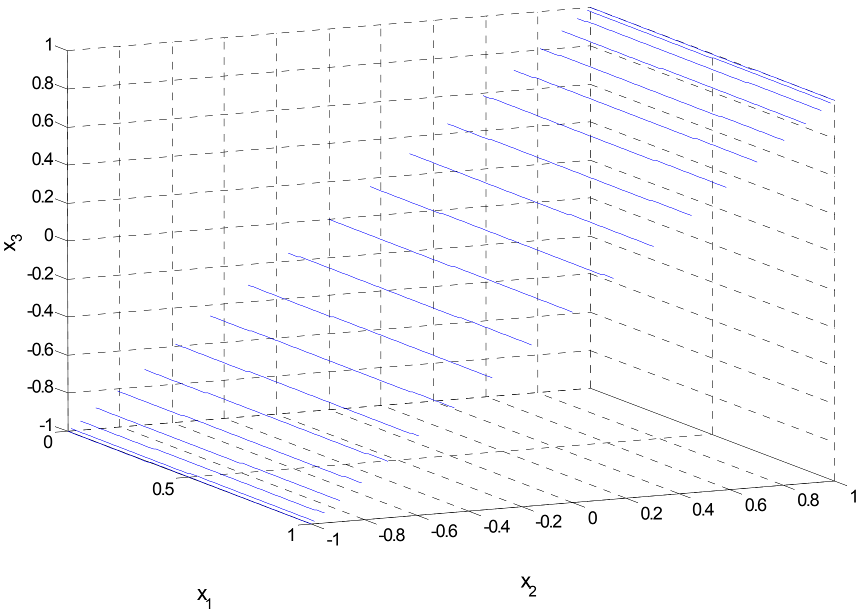


Figure 30. FDA1 solution for the first three variables. A design vector with 10 variables is used in this section's experiments.

The FDA1 problem is defined as follows (Farina, Deb & Amato 2004):

$$\begin{aligned}
&\text{minimize} && \mathbf{f}(f_1(\mathbf{x_I},t), f_2(\mathbf{x},t)) \\
&\text{where} && f_1(\mathbf{x_I},t) = x_1 \\
& && f_2(\mathbf{x},t) = g(\mathbf{x_{II}},t)\cdot h(\mathbf{x_{III}}, f_1(\mathbf{x_I},t), g(\mathbf{x_I},t)) \\
&\text{and} && \mathbf{x} = \left[\mathbf{x_I}\ \mathbf{x_{II}}\right]^{\mathrm{T}} = \left[x_1\ x_2\ \cdots\ x_n\right]^{\mathrm{T}} \text{ is the design vector}
\end{aligned} \tag{4.4}$$

The *g* and *h* functions are defined:

$$g(\mathbf{x_{II}}) = 1 + \sum_{\mathbf{x} \in \mathbf{x_{II}}} (x_i - G(t))^2 \tag{4.5}$$

$$h(f_1, g) = 1 - \sqrt{f_1 / g} \tag{4.6}$$

$$\begin{gathered} G(t) = \sin(0.5\pi t),\ t = \frac{1}{n_t} \left\lfloor \frac{\tau}{\tau_T} \right\rfloor \\ \mathbf{x}_I = (x_1) \in [0,1],\ \mathbf{x}_{II} = (x_2, \ldots, x_n) \in [-1,1] \end{gathered} \tag{4.7}$$

In this definition, τ is the current number of function evaluations, τ_T is the number of function evaluations for which the time t remains fixed and the objective remains constant (objective change period), and n_t is the number of distinct timesteps in one time unit. The Pareto optimal front is analytically solved to be:

$$f_2 = 1 - \sqrt{f_1} \tag{4.8}$$

And the Pareto optimal set is:

$$\begin{gathered} x_1 \in [0,1] \\ x_{2,\ldots,n} = G(t) = \sin(0.5\pi t) \end{gathered} \tag{4.9}$$

Hence, according to the analytic solution of the problem, x_1 spans [0,1] evenly in order to cover the Pareto optimal set and the rest of the design vector oscillates harmonically in time between -1 and 1, as we can see in Figure 30.

The design vector has a dimension n = 10. A time discretization of n_t = 10 timesteps per time unit is employed. The n_t parameter controls the problem change severity, since it dictates the number of timesteps in one full cycle (which corresponds to four time units). The change period is τ_T = 500 evaluations per timestep. Severity is kept constant throughout this set of experiments in order to observe the effect of changes in the number of evaluations per timestep. All results in this section are averages of 20 runs for each case.

4.5.4 Results

This set of numerical results is encouraging as both stated goals are satisfied: The time-changing optimization concept works satisfactorily, and the use of an anticipatory population through the Feed-forward Prediction Strategy results in a significant improvement of the algorithm's performance.

The different algorithmic arrangements used (D-QMOO, D-QMOO/AR and D-QMOO/VAR as described in section 4.5.1) serve to underline the fact that the FPS is used as an 'add-on' tool with the algorithm, and consequently that it is a concept which can potentially be applied to other time-changing population based algorithms in order to improve performance.

Numerical experiments with constant objective change frequency

In this set of experiments, the objective change frequency is kept constant at τ_T = 500 evaluations per timestep. The problem is run for a duration of 300,000 evaluations (600 timesteps), and the performance of the different algorithmic versions is compared. Table 3 shows the Feed-forward Prediction Strategy to have a positive effect. The effect is stronger when using the univariate model, where a separate autoregressive model is used to predict the next timestep's value for every variable in the design vector: using the D-QMOO/AR algorithm brings a reduction of 31%

in the Pareto front error[7] and almost 50% in the design vector error when compared to D-QMOO without FPS. This signifies a very positive effect from the FPS. The multivariate model (D-QMOO/VAR) also has a positive but much smaller effect: 2.5% reduction in the Pareto front error and 3.4% reduction in the design vector error.

A possible reason for the better performance of the univariate model is that each design variable's forecast is isolated from the other variables. In contrast, in the multivariate model each variable is affected by the whole design vector and hence an error in one design variable can propagate to the others. This is a problem-specific result. A forecasting model which takes into account the linkage between design variables might perform better in problems with strongly coupled variables.

Table 3. Objective (Pareto front convergence) and design vector error. The objective (Pareto) and the design vector errors are calculated as described in (Farina, Deb & Amato 2004), expressing the deviation of the current population from the actual solution. $\overline{e}_f$ and $\overline{e}_x$ denote time averages of the errors during each run, and their mean and standard deviation over 20 runs of 300,000 evaluations is shown.

	objective error		design vector error	
algorithm	$\overline{e}_f$ **mean**	$\overline{e}_f$ **st. dev.**	$\overline{e}_x$ **mean**	$\overline{e}_x$ **st. dev.**
D-QMOO/AR (univariate)	0.02984	$9.44\ 10^{-4}$	0.00298	$1.62\ 10^{-4}$
D-QMOO/VAR (multivariate)	0.04231	$7.34\ 10^{-4}$	0.00573	$1.70\ 10^{-4}$
D-QMOO (no predictor)	0.04336	$6.94\ 10^{-4}$	0.00593	$1.41\ 10^{-4}$

Numerical experiments with varying objective change frequency

In this set of numerical experiments, the frequency of change for the objective function is varied. This parameter controls the number of objective function evaluations between changes in the objective (number of evaluations per timestep). Recalling the discussion in section 4.1, the frequency affects the solution quality in a time-changing optimization problem since it defines the amount of time available to the algorithm for convergence during each timestep. A measure of performance in time-changing environments is how closely the algorithm converges to the true Pareto front at each timestep for a given frequency or, conversely, the maximum frequency for which the algorithm can provide a given solution accuracy. Solution accuracy can be quantified in this problem by the Pareto and design vector errors. The experiments presented here show that

[7] Objective and design vector errors were calculated as prescribed by Farina et al (Farina, Deb & Amato 2004) using the known analytic solution of the problem. It must be noted here that a spacing of 0.05 in the x_1 dimension was used to discretize the analytic solution. The choice of spacing affects the error magnitude, and a consistent spacing needs to be used in order to compare results.

the FPS can especially augment the performance of an evolutionary algorithm in a dynamic environment at high change frequencies (low change periods), where the algorithm would otherwise perform poorly.

In this set's results the change period is varied from 500 to 30,000 evaluations per timestep. Results are averaged over 20 runs. The error at each time instant is measured at the end of the timestep just before the objective changes. The positive effect of using the feed-forward prediction strategy is apparent if one looks at Figure 31. As soon as the FPS is switched on at t = 10, the mean objective error drops by almost 37%, from 43.1x10^{-3} to 27.2x10^{-3}. At the same time, the fact that the algorithm can track the solution at this high frequency even without the FPS (before *t* = 10) shows the effectiveness of the diversity control used in D-QMOO.

The effect of a decreasing change frequency can be seen in Figure 32 and Figure 33 where the change period is increased to 2000 and 5000 respectively. The benefit from using the FPS attenuates as the change period increases. In Figure 33 (5000 evaluations per time-step) the effect of the FPS is almost imperceptible. The average error is smaller as the period increases since the algorithm has more function evaluations available and hence a better chance to converge to the Pareto optimal set during each timestep, independent of the FPS. However the effect of the FPS is obvious in high frequencies as we saw in Figure 31, especially when using the univariate autoregressive model (D-QMOO/AR algorithm). This is also apparent if we examine the average objective and design vector error in Figure 34 and Figure 35. In low periods D-QMOO/VAR performs better than D-QMOO, and the univariate D-QMOO/AR performs significantly better than the first two. As the period increases, the performance of the three algorithms converges to the same level.

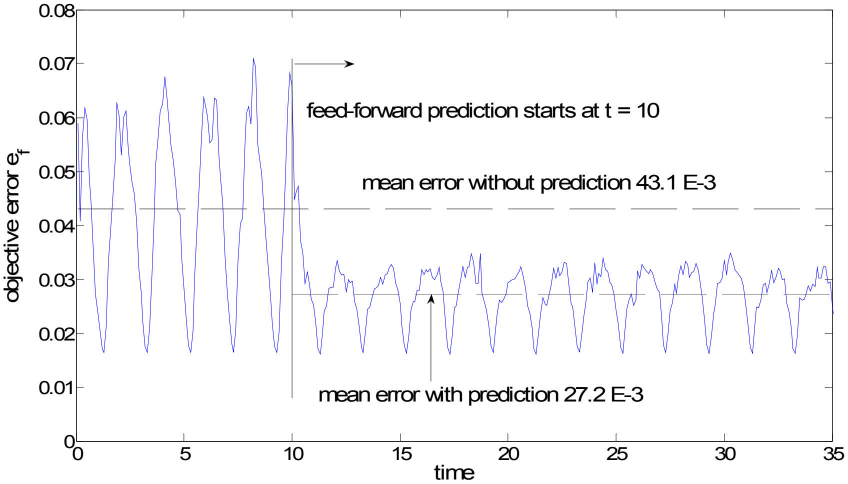


Figure 31. Objective error during the run. Change period 500 evaluations per timestep. D-QMOO/AR algorithm. The positive effect of FPS in high frequency is apparent.

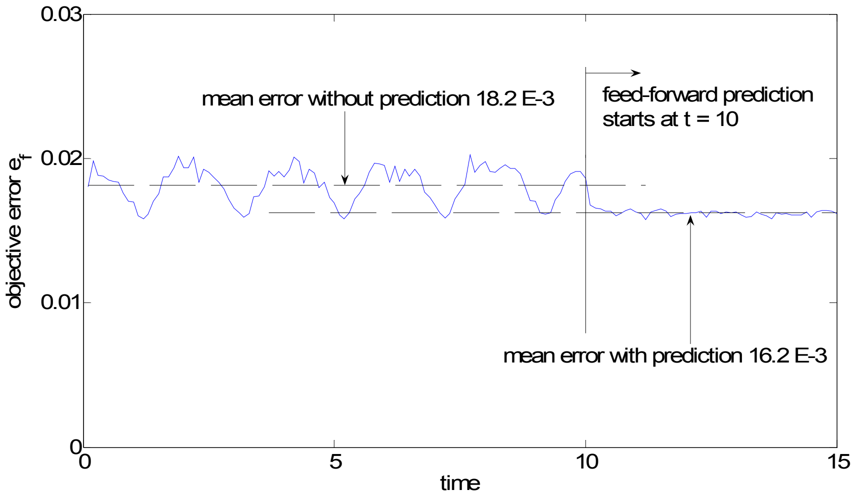


Figure 32. Objective error during the run. Change period 2000 evaluations per timestep. D-QMOO/AR algorithm.

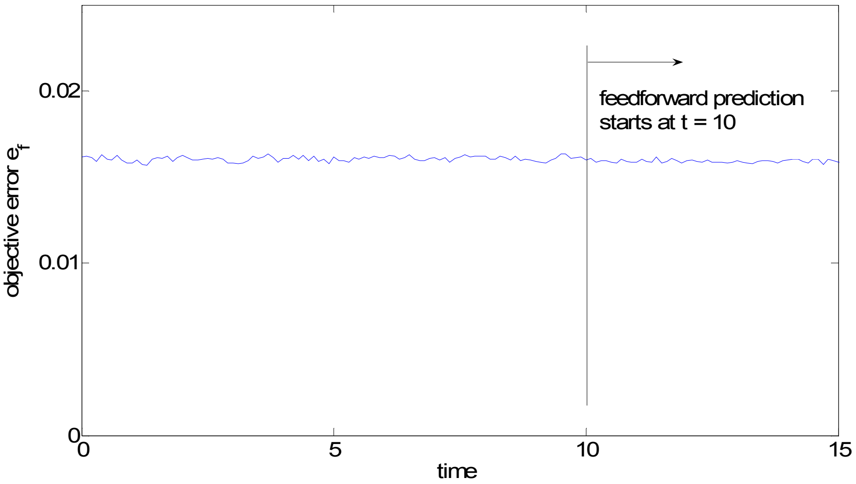


Figure 33. Objective error during the run. Change period 5000 evaluations per timestep. D-QMOO/AR algorithm. The prediction's effect in this lower frequency is negligible.

The positive effect of the FPS is directly tied to the forecast quality which determines how close the prediction individual is placed to the best discovered individual and the actual optimum. In Figure 36 the time history of the second design variable for the prediction individual of the first anchor point is shown, together with the time history of the best discovered individual. It is evident that in this case the prediction individual often coincides with the best discovered solution, and both are very close to the actual solution. The frequency here is 2000 evaluations per timestep, which gives the algorithm adequate time to converge and also to fit an accurate AR model, resulting in good forecasting accuracy.

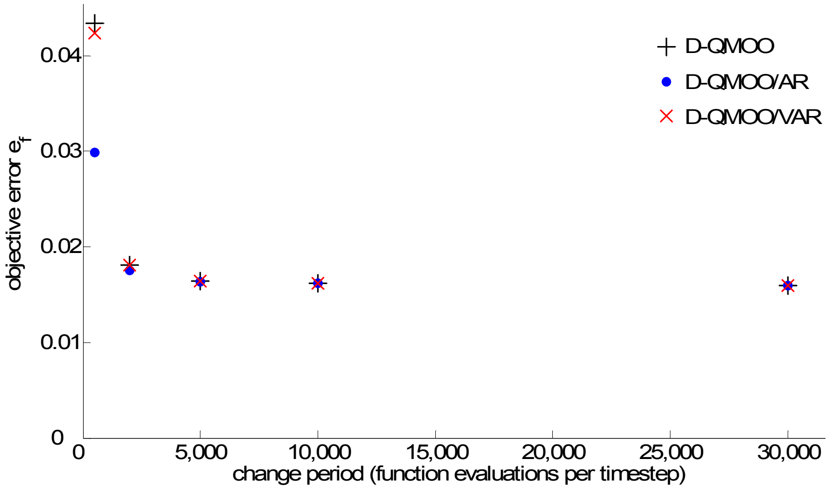


Figure 34. Objective (Pareto) error with change period. Initially D-QMOO/AR has a significantly smaller error than D-QMOO. The performance of the three algorithms converges as the period increases and the effect of the FPS attenuates.

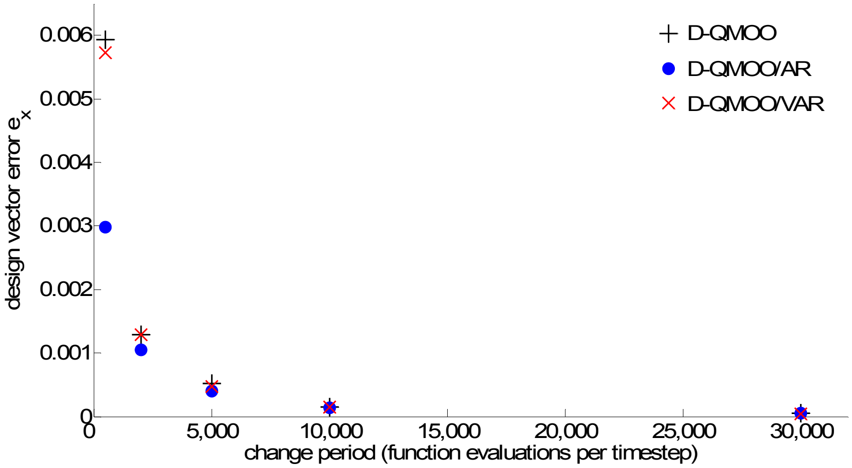


Figure 35. Design vector error with change period.

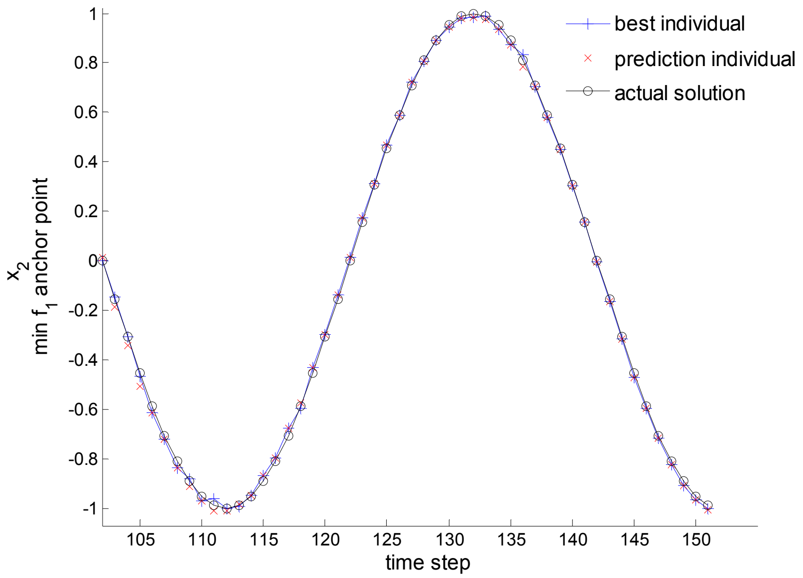


Figure 36. Time history of the forecast (anticipatory individual) and the best discovered individual, along with the analytical solution. (D-QMOO/AR, 2000 evaluations per timestep).

An illustration of the function of the anticipatory population can be seen in the snapshots of Figure 37, where the analytical solution and the population are shown at the beginning and the end of the same timestep. At the beginning of the timestep, the bulk of the population is still around the previous timestep's locus of the Pareto optimal set. The two anticipatory individuals however already lie near the actual solution of the current timestep. They have been created at the end of the previous timestep using the autoregressive model's forecast, and their function is to show the way to the rest of the population towards the new solution. At the end of the timestep, the front individuals have almost all converged onto the Pareto optimal set. The effect of the prediction set can also be seen in the Pareto front approximations of Figure 38. These instances are from the high change frequency of 500 evaluations per timestep. Right after a change in the objective (left column) the front individuals are at a distance from the actual solution (black line). However the anchor point anticipatory individuals are already closer to the actual front. At the end of the timestep (right column) the whole front population is closer to the actual solution, as close as the high change frequency allows.

4.6 Conclusion

The proposed concept for the solution of time-changing problems was presented in this chapter. The two most important attributes of this concept are the use of anticipation in the form of the Feed-forward Prediction Strategy for the improvement of a time-changing algorithm's performance, and the combination of anticipation and convergence-diversity balance as a robust and well-performing architecture for population based algorithms in time-changing problems.

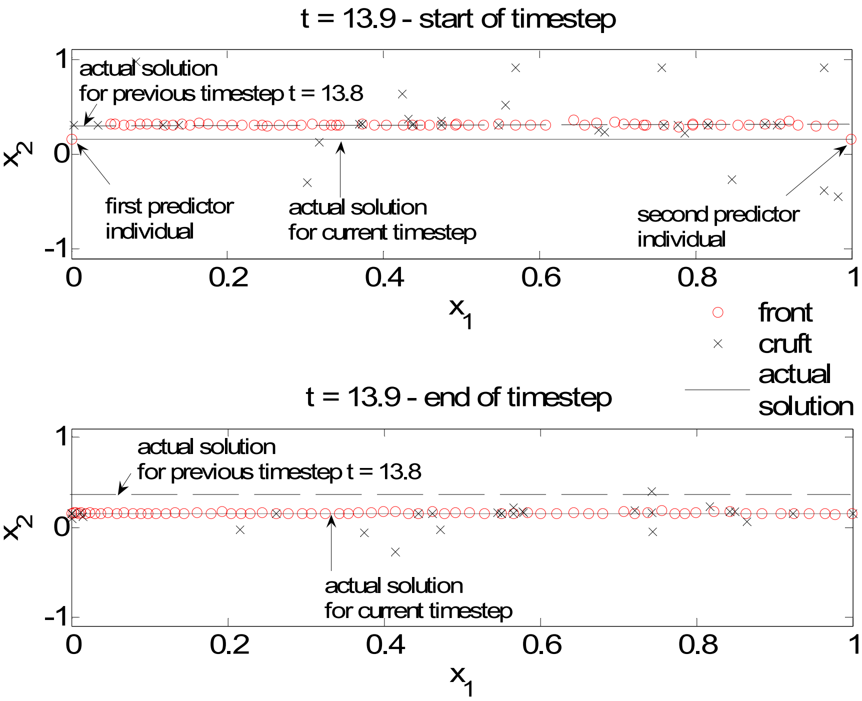


Figure 37. In the top figure, the time has just advanced to 13.9 and we are at the start of the new (current) timestep. The two anticipatory individuals, one for each anchor point of the Pareto plot, already lie near the current actual solution (shown as a continuous line), while the rest of the population is still around the previous timestep's Pareto optimal set (shown as a dashed line). At the end of the timestep (bottom figure), the rest of the front individuals have followed the anticipatory population and converged onto the current Pareto optimal set. (D-QMOO/AR, 5000 evaluations per timestep)

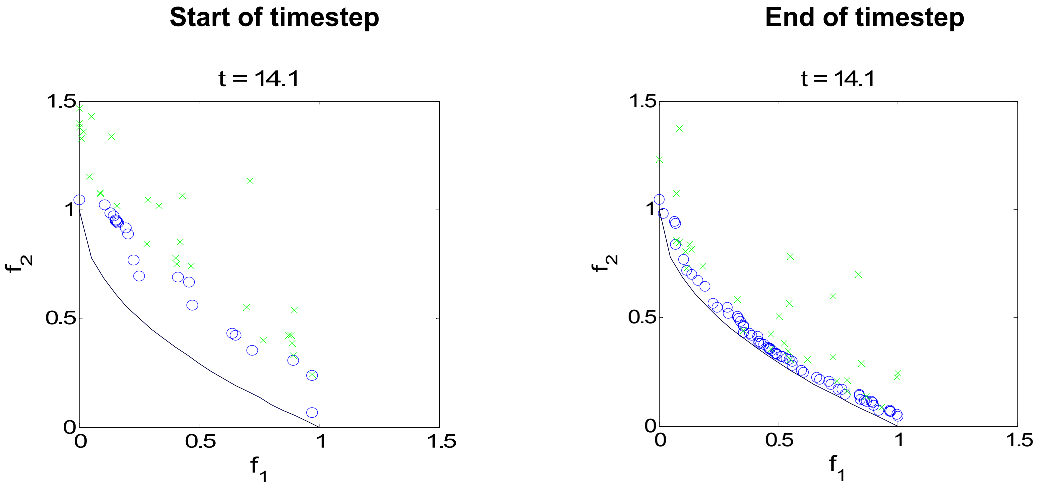


Figure 38. Pareto front approximation at the beginning and end of timesteps (D-QMOO/AR, 500 evaluations per timestep). Blue circles: front individuals, green x's: cruft individuals, black line: actual Pareto front.

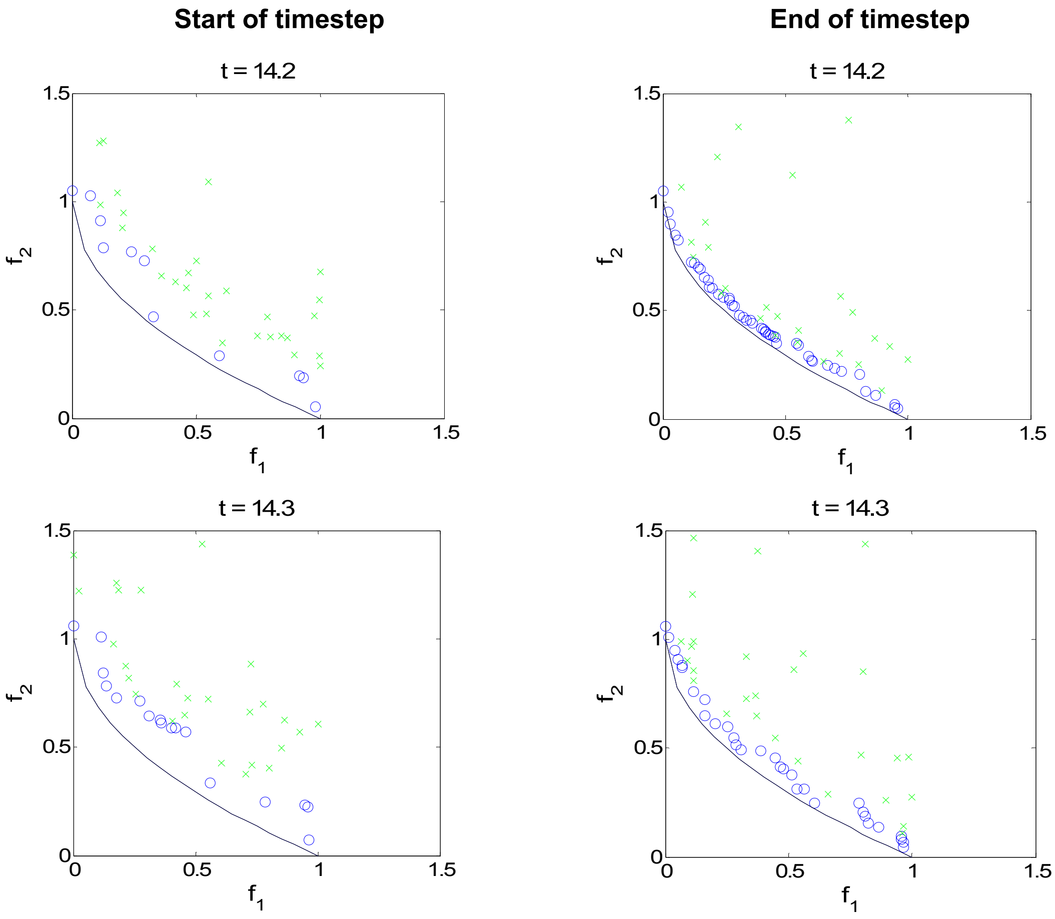


Figure 38 (continued).

The FPS adopts a forward-looking attitude in which the optimization algorithm exploits past information and prepares for the change before it arrives instead of simply reacting to it. Initial results from the implementation of the method are promising. The FPS improves the solution accuracy, especially in the critical cases where the frequency of change is high and the algorithm has otherwise little time to converge to the Pareto front. The ideas and implementation presented in this chapter are only a subset of a large scope of possible forms that the Feed-forward Prediction Strategy and the overall time-changing optimization concept can take.

In the next chapter, the topology of the anticipatory population is discussed.

5 Topology of the Anticipatory Population

The anticipatory population is arguably the most important element of the Feed-forward Prediction Strategy, the time-changing optimization method proposed in this work which exploits predictability in the objective's change pattern through the combination of a forecasting technique with a population-based algorithm. In this chapter the topology of the anticipatory population is studied in detail. Specifically, the two basic questions posed in section 4.4.5 of the previous chapter are addressed:

- *Which part of the solution to include in the anticipatory population?* This question becomes especially pertinent in multi-objective problems where the solution is a non-dominated front, and it is not clear which individuals from the front should be included in the anticipatory population.
- *Given a forecast for the location of the individuals, how should the anticipatory population be created?* This question becomes important when the forecasting error is high, and an anticipatory population more complex than a single individual on the forecast coordinates is required in order to include the true optimum in the neighborhood of the prediction set.

An illustration of these issues can be seen in the sketch of Figure 39. Regarding the first question, we propose the inclusion of additional individuals in the anticipatory population. These individuals are spaced as evenly as possible along the Pareto front in order to cover it efficiently. Regarding the second question, we propose surrounding the forecast coordinates with anticipatory individuals. These individuals can be placed, for example, at the corners of a hypercube which is centered on the forecast and dimensioned according to the estimated forecast error. Populating the forecast neighborhood in such a way has the intention of including the true optimum in the space enclosed by the prediction set, thereby increasing the probability that the anticipatory individuals will lead to the optimum's discovery.

5.1 Selecting anticipatory individuals in order to cover the non-dominated front

We wish to create a selection rule in order to pick regions of the design space (in the form of individuals) which will form the anticipatory population. In a single-objective problem, the answer is clear as one simply needs to track the global optimum, or a set of desired local optima. In a multi-objective problem though, the solution is composed of the non-dominated Pareto-optimal set (POS) which maps to the Pareto front in the objective space. Hence we need to decide which part of the POS to track and predict.

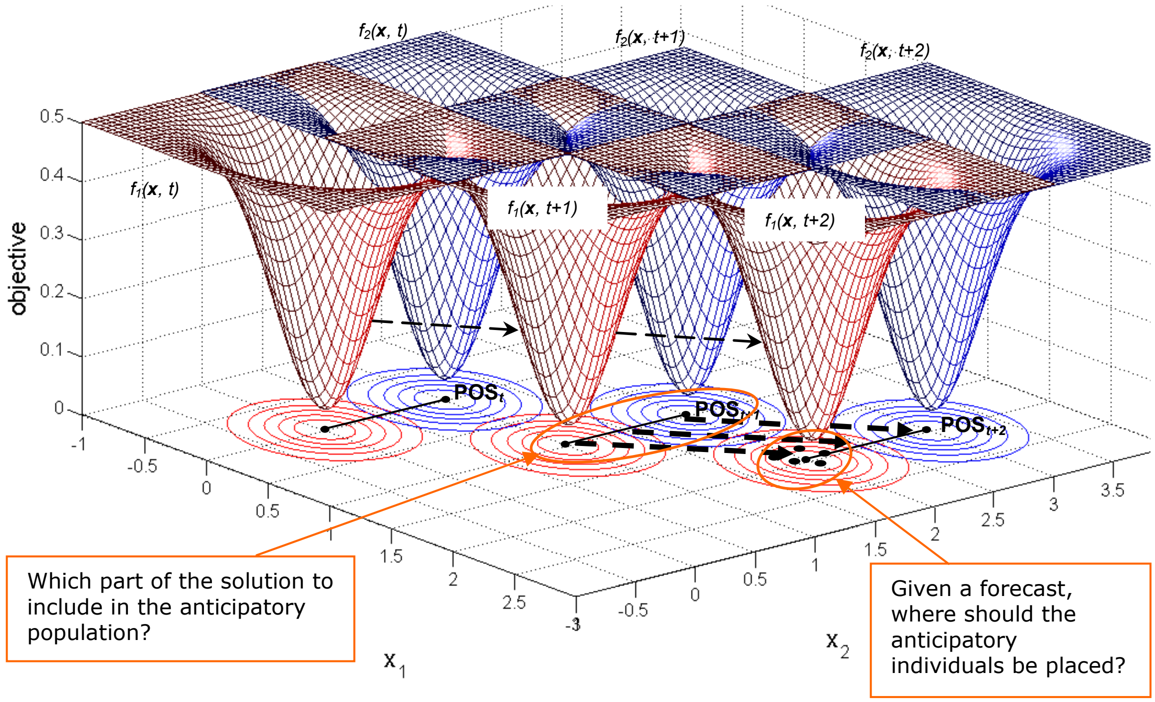


Figure 39. The two issues regarding the topology of the anticipatory population.

The simple approach taken in the experiments of the previous chapter was to form an anticipatory population from the extremities of the Pareto front (anchor points). These points correspond to the best solutions for each of the objectives (minf_1 and minf_2 in a two-objective problem, as shown in Figure 40). However it is obvious that such an approach leaves a large portion of the POS uncovered. Tracking only the anchor points might locally help the population discover the extremities of the POS, but the effect on the discovery of other regions of the POS will be limited. Therefore it makes sense to create a prediction set that offers a better coverage.

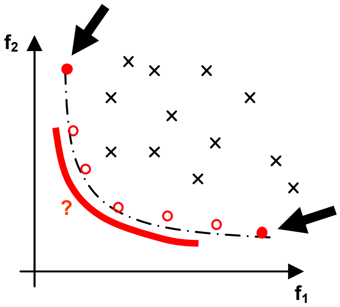


Figure 40. A prediction set consisting of only the anchor points leaves large areas of the front uncovered.

In order to accomplish this, additional prediction points need to be selected from the POS. However, the criterion to use for the selection of those points is not clear. The anchor points have a distinct topological attribute since they are the extremities of the front. Any other point selected should similarly have some kind of attribute that can help the algorithm discern it from the rest of the non-dominated solutions. Such a candidate is the point on the Pareto front which is closest to the ideal point[1]. This point can become the third member of the prediction set as shown on the sketch of Figure 41. We call this the Closest-To-Ideal (CTI) point.

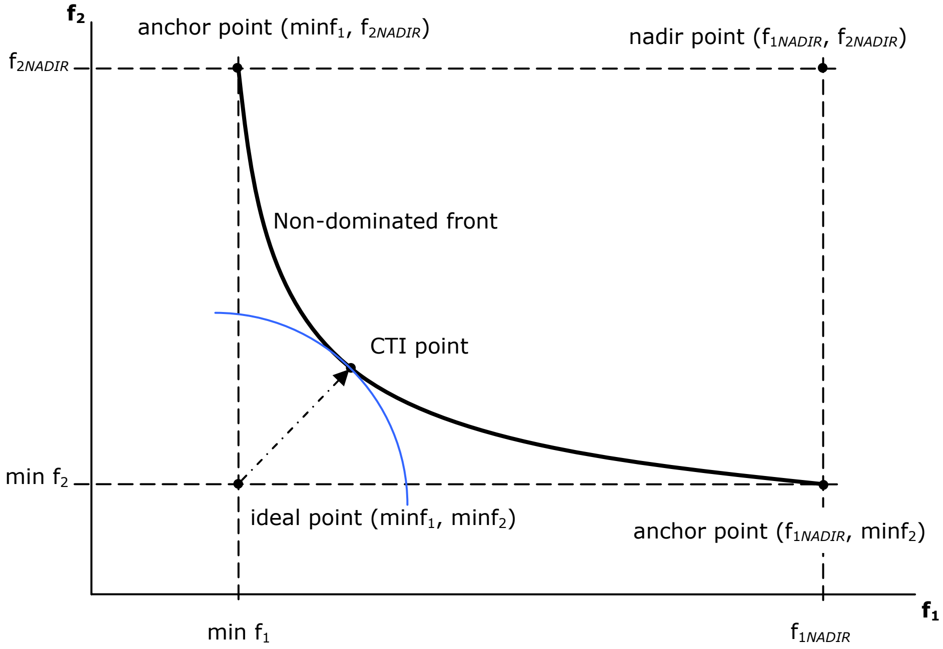

f₂
anchor point ($\min f_1$, f_{2NADIR})
nadir point (f_{1NADIR}, f_{2NADIR})
f_{2NADIR}
Non-dominated front
CTI point
$\min f_2$
ideal point ($\min f_1$, $\min f_2$)
anchor point (f_{1NADIR}, $\min f_2$)
$\min f_1$
f_{1NADIR}
f_1

Figure 41. Prediction set including the CTI point for a two-objective problem.

In many cases, this technique provides an intermediate point which lies near the middle of the Pareto front, making for an even coverage of the front by the prediction set. Geometrically the selection of this point is similar to the approach used in the classical multi-objective optimization method of compromise programming (see for example Deb 2001). An advantage of the CTI point is that it usually yields a trade-off design that balances both objectives, being practical to implement. As we will see in the numerical experiments that follow, the CTI point is a useful addition to the anticipatory population.

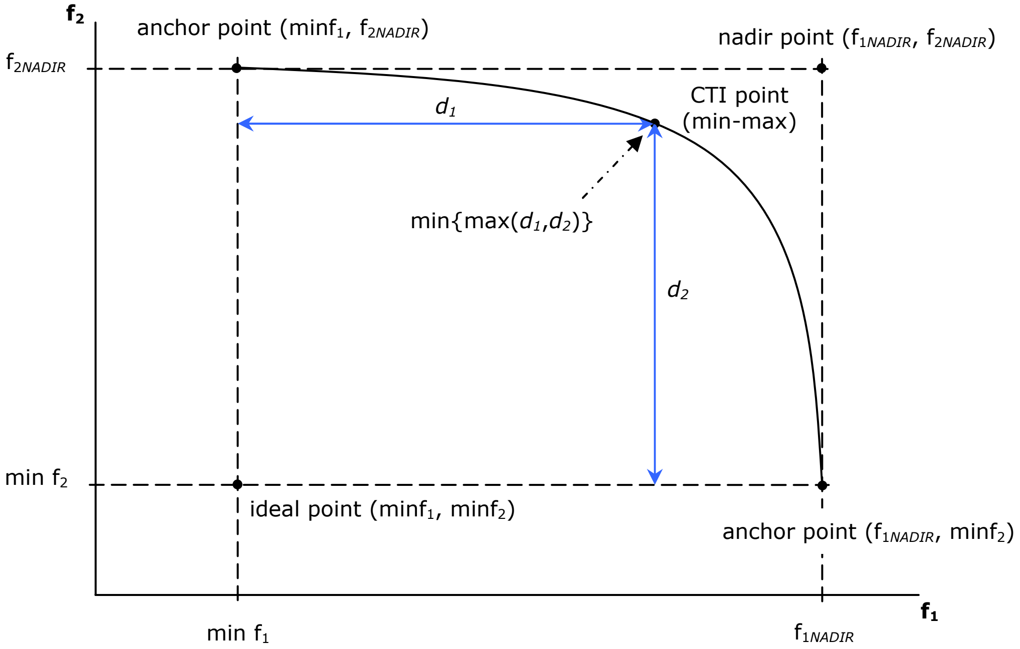
Anticipatory population and geometry of the non-dominated front

An anticipatory population consisting of the anchor points and the CTI point as described in the previous section is expected to work best in problems where the non-dominated front is continuous and convex or not excessively concave. The anticipatory populations used in this

[1] The *ideal point* is defined by the best discovered coordinates for each of the objectives, as shown in Figure 41.

work are of this form, always consisting of the anchor points and in some cases including the CTI point. However difficulties might arise when the front's geometry is different. Here we discuss some cases in which the front's geometry might pose problems in the creation of the anticipatory population, and propose (without implementation) some potential solutions.

One case arises when the algorithm has trouble selecting the CTI point because the front is very concave and large parts of it have a similar distance to the ideal point, or because the front displays several strong changes of curvature. As a solution, a min-max approach[2] could be used instead of a minimum distance in order to discover the CTI point, as shown in Figure 42.



f_2
anchor point (minf_1, f_{2NADIR})
nadir point (f_{1NADIR}, f_{2NADIR})
f_{2NADIR}
d_1
CTI point
(min-max)
min{max(d_1,d_2)}
d_2
min f_2
ideal point (minf_1, minf_2)
anchor point (f_{1NADIR}, minf_2)
f_1
min f_1
f_{1NADIR}

Figure 42. Min-max approach to finding the CTI point could perform better in the case of concave Pareto fronts.

Disjoint non-dominated fronts offer another example. If the front is disjoint, it consists of two or more components at a finite distance from each other in the objective space. In this case, a way to create anticipatory populations could be to use a clustering algorithm in the objective space in order to divide the front into its components, and subsequently to track each component separately with its own CTI and anchor points.

Numerical experiments

A set of numerical experiments using the FDA1 problem show that the inclusion of the CTI point in the prediction set improves performance noticeably, especially in the high frequency case when it is hard for the algorithm to converge on the front in the available number of function

[2] An idea proposed to the author by Dr. Michael Yukish (Pennsylvania State University) in a personal discussion during the 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference.

evaluations. The results for different objective change frequencies can be seen in Table 4. For example in the frequency of 500 evaluations per timestep, including the CTI point in the prediction set results in an error reduction[3] of around 17% for the objective and 23% for the design vector. At low frequencies, more than 2000 evaluations per timestep, the two prediction sets (anchor points only and anchor points plus the CTI point) yield similar performance since in these cases the algorithm has sufficient time to converge to the solution for each timestep without requiring help from the prediction set. Recall that the same behavior was encountered in the numerical experiments of the previous chapter, where the effect of the FPS was stronger in high objective change frequencies.

Table 4. Anticipatory population performance with different objective frequencies. Results are averaged over 20 runs for each experiment. A design vector of dimension 10 is used.

		objective error		design vector error	
Objective change frequency	Anticipatory population	$\overline{e_f}$ mean	$\overline{e_f}$ st. dev.	$\overline{e_x}$ mean	$\overline{e_x}$ st. dev.
500 evals/timestep	Anchor points	0.0300	9.440E-4	2.980E-3	1.620E-4
	Anchor points and CTI	0.0254	5.845E-4	2.302E-3	1.073E-4
2000 evals/timestep	Anchor points	0.0175	1.361E-3	1.044E-3	3.340E-5
	Anchor points and CTI	0.0173	7.54E-5	9.437E-4	2.390E-5
5000 evals/timestep	Anchor points	0.0164	6.406E-5	3.987E-4	3.779E-5
	Anchor points and CTI	0.0164	4.572E-5	3.051E-4	9.347E-6
10000 evals/timestep	Anchor points	0.0162	4.720E-5	1.336E-4	4.404E-5
	Anchor points and CTI	0.0162	3.212E-5	4.640E-5	1.882E-6
30000 evals/timestep	Anchor points	0.0160	3.850E-5	5.130E-5	2.451E-5
	Anchor points and CTI	0.0159	3.370E-5	1.103E-6	1.134E-7

An illustration of the effect of the anticipatory population can be seen in Figure 43. The Pareto front discovered by the algorithm is shown for three cases: no use of the FPS, FPS with a prediction set consisting of the anchor points and FPS with a prediction set including the CTI point as well. The snapshot is at the beginning of a timestep. In the first case, all individuals are at a large distance from the actual Pareto front since they are still around the previous timestep's solution in the variable space. In the second case, the anchor point anticipatory individuals have already been placed near the new solutions for the anchor points. In the third case, the CTI prediction individual has been added to the prediction set and is already near the actual Pareto front, guiding the rest of the solution.

[3] As in the previous chapter, objective and design vector errors were calculated as prescribed by Farina et al (Farina, Deb & Amato 2004) using the known analytic solution of the problem and a spacing of 0.05 in the x_1 dimension.

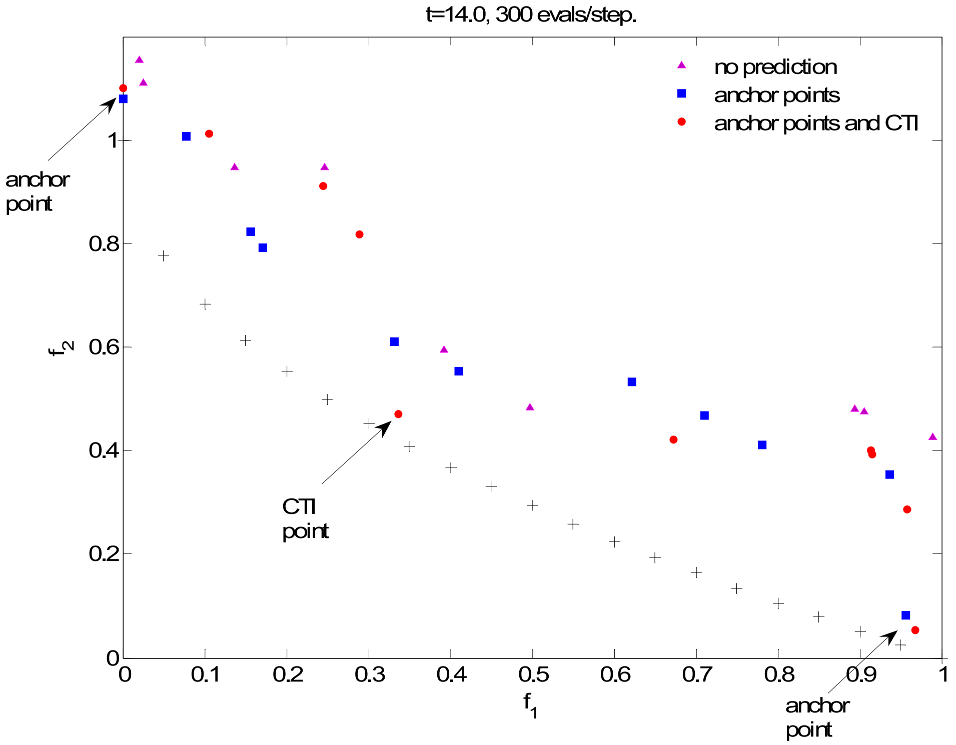


Figure 43. A snapshot at the beginning of a timestep for three different algorithm versions (no FPS, FPS with anchor point prediction set, FPS with anchor and CTI points prediction set). The effect of the anticipatory individuals is evident.

5.2 Populating the neighborhood of the forecast in order to reduce the effect of forecast error

Having decided which parts of the solution and hence which individuals to include in the prediction set, and having created forecasts for their location in the next timestep, we need to determine exactly where in the design space to place the anticipatory individuals. A straightforward approach for the topology of the prediction set is to use a single individual placed on the forecast coordinates. This option has the advantage of inducing the least cost in terms of objective function evaluations since only one individual needs to be evaluated for each prediction point, and it is the approach used in the previous chapter where the FPS was introduced.

Due to the forecast error[4] however, there is a distance between the prediction and the actual location of the next timestep's optimum (see the sketch of Figure 44). Hence it is potentially helpful to distribute a number of individuals in the forecast's neighborhood instead of only placing a single individual on the forecast coordinates, in order to aid the discovery of the next optimum. Here we will examine two different ways of doing that.

Hypercube prediction set

One way to populate the forecast's neighborhood is to form a hypercube with the forecast coordinates at its center, as shown in the sketch of Figure 45. Individuals are placed at the center

[4] see section 4.4.7

and at each of the hypercube corners. Forming such an anticipatory population offers the possibility of surrounding the true optimum with the anticipatory individuals. The crossover operators used by D-QMOO[5] make the interior points of a hypercube reachable by individuals lying at its corners. So if the true optimum's location is enclosed by the hypercube it can potentially be reached by the anticipatory individuals.

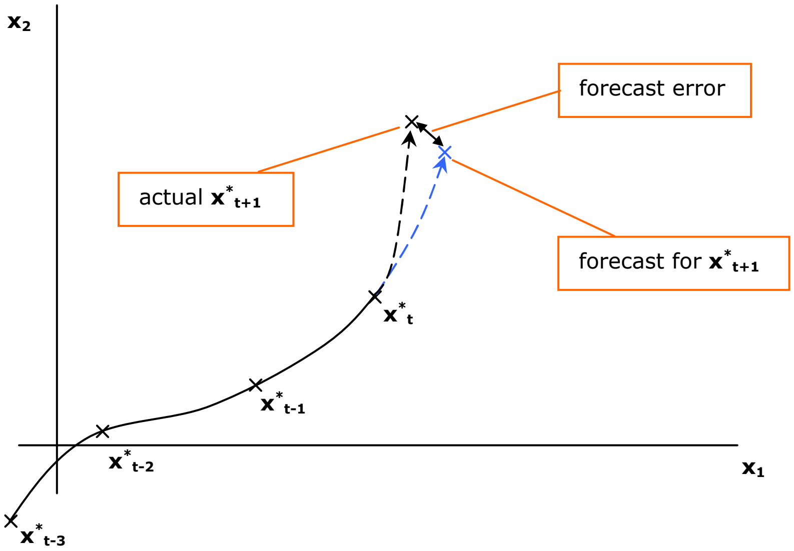


Figure 44. Forecast error.

In order to determine the size of the hypercube, it is rational to use an estimate of the forecast error. A low forecast accuracy implies a large expected forecast error, and a large forecast error requires a large hypercube in order to enclose the actual optimum.

Various methods can be employed in order to get an estimate of the forecast accuracy. For example the forecasting model itself might have a way of creating an error estimate. This is the case with autoregressive models since, apart from the forecast, they create an estimate of the error covariance matrix (Akaike, Nakagawa 1972, Schneider, Neumaier 2001). This estimate can be used to extract confidence intervals for the forecast for each variable, as described by Lütkepohl (Lütkepohl 1994).

Specifically, if we denote $\boldsymbol{\varepsilon}_t$ the forecast error at the t timestep:

$$\boldsymbol{\varepsilon}_t = \mathbf{x}^*_t - \tilde{\mathbf{x}}^*_t \qquad (5.1)$$

the AR model provides an estimate of the error's covariance matrix:

$$\operatorname{cov}(\boldsymbol{\varepsilon}_t) = [\sigma_{ij}] \qquad (5.2)$$

Then, the p-probability confidence interval for each variable of the design vector is:

[5] Recall the description of QMOO's operators in section 2.4, and also see (Leyland 2002, Reeves, Rowe 2003) for an analysis of the operators' behavior.

$$\begin{aligned} &\tilde{x}_{k,t} \pm z_p \sigma_k, && \text{for } k = 1,\ldots,n \\ &\text{where} && z_p = \sqrt{2} \cdot erf^{-1}(p) \end{aligned} \tag{5.3}$$

In (5.3) erf^{-1} is the inverse error function, and it has been assumed that the error is normally distributed with zero mean (Lütkepohl 1994). In order for the hypercube to include the true optimum with probability p we need to place its corner individuals at a distance of $z_p \cdot \boldsymbol{\sigma}_\mathbf{k}$ from the forecast location in each k dimension of the design space, as we can see in Figure 45. Then the actual optimum has a probability p of lying inside the hypercube.

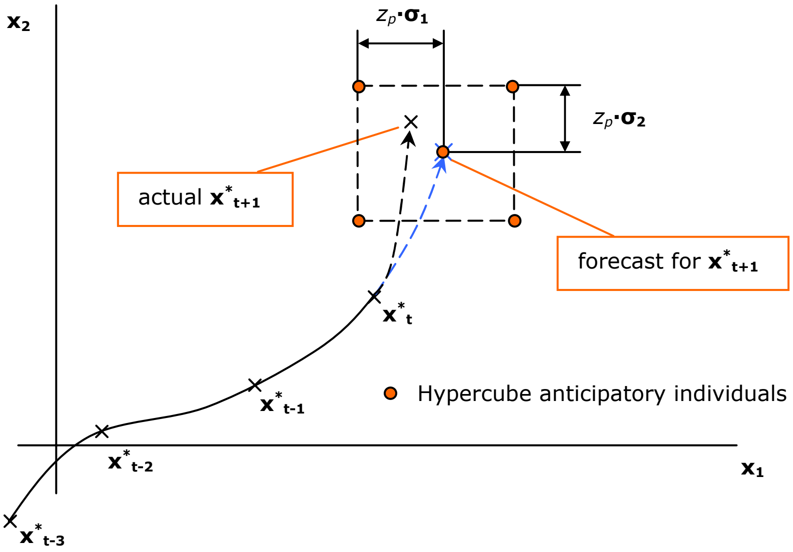

x₂
$z_p \cdot \boldsymbol{\sigma}_1$
$z_p \cdot \boldsymbol{\sigma}_2$
actual $\mathbf{x}^*_{t+1}$
forecast for $\mathbf{x}^*_{t+1}$
$\mathbf{x}^*_t$
$\mathbf{x}^*_{t-1}$
● Hypercube anticipatory individuals
$\mathbf{x}^*_{t-2}$
x₁
$\mathbf{x}^*_{t-3}$

Figure 45. Hypercube prediction set for a two-dimensional variable space.

The main disadvantage of the hypercube shaped anticipatory population is its computational cost as the dimension of the design vector increases. The number of corners for an n-dimensional hypercube is 2^n+1. In a 6-dimensional design space for example, the anticipatory population requires 65 individuals for each prediction point. The total population of the evolutionary algorithm is usually in the order of 100 individuals. Therefore, if we assume that we are solving a two-objective problem and that we track and predict the two anchor points of the Pareto front, any problem of dimension 6 or more implies that the prediction set will need to be larger than the total population, making it impractical and costly.

Latin hypercube prediction set

A compromise solution to the hypercube's dimensionality problem is to sample some locations around the forecast with a two-level Latin Hypercube (LH – McKay, Beckman & Conover 1979) instead. As before, confidence intervals can be used in order to set the size. The anticipatory population around the forecast is in this case composed of three individuals, independently of the design space dimension: the centre point and the two LH points. Figure 46 illustrates this prediction set for a two-dimensional problem. Each variable of the LH points draws its value exactly once from either of two levels:

$$\mathbf{x}_{tLH} = \left[\tilde{x}_{1,t} \pm z_p\sigma_1 \;\; \tilde{x}_{2,t} \pm z_p\sigma_2 \quad \ldots \quad \tilde{x}_{n,t} \pm z_p\sigma_n \right] \tag{5.4}$$

The Latin hypercube prediction set offers a less effective coverage of the forecast's neighborhood because of the smaller number of individuals. However every point in the volume enclosed is still theoretically reachable by the anticipatory individuals through the crossover operators (Reeves, Rowe 2003).

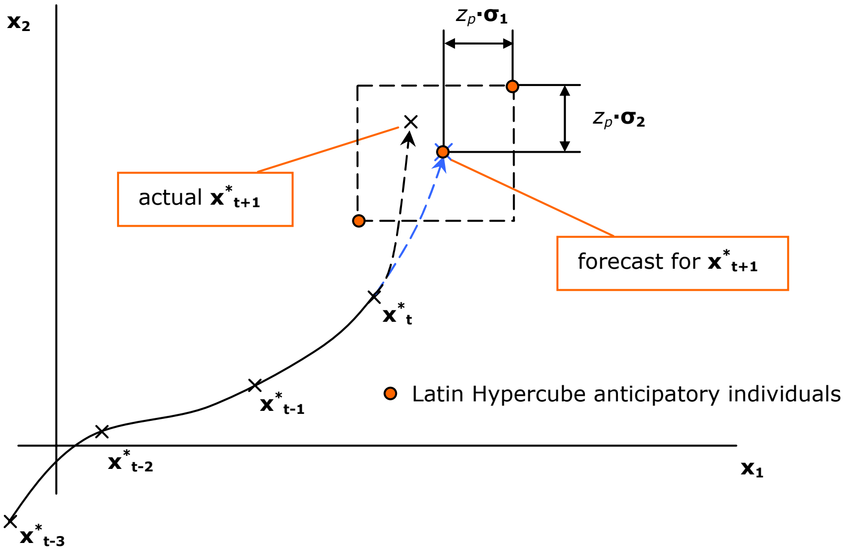


Figure 46. Latin hypercube prediction set for a two-dimensional variable space. This prediction set is always composed of three anticipatory individuals, regardless of the variable space dimension.

Numerical experiments

The following results elucidate the performance and behavior of the three anticipatory population topologies we just discussed (single individual, hypercube and Latin hypercube). Numerical experiments were done using the FDA1 problem, at an objective change frequency of 300 function evaluations per timestep.

In Figure 47 and Figure 48 we can see the solution accuracy for four different cases. In the first three D-QMOO is used with the Feed-forward Prediction Strategy, and the three different prediction set topologies (hypercube set with $2^n + 1$ individuals per prediction point, Latin hypercube set with 3 individuals per prediction point, and single point prediction set). In the fourth case D-QMOO is used without the FPS, relying only on the convergence-diversity balance to track the moving POS.

The hypercube has the best accuracy (lowest error) for low design vector dimensions since it offers the fullest coverage of the forecast location's area. However when n is greater than 5 its performance decreases dramatically – it even has larger error than D-QMOO with no FPS. This is due to the fact that for n = 6 the prediction set size is 130 individuals, larger than the total population of 100. The result is that most of the individuals end up concentrated around the prediction points and the rest of the Pareto front remains uncovered. The Latin hypercube which is composed of only three individuals, independently of the design vector dimension, has consistently better accuracy than the single-point set and than D-QMOO without FPS and has the best overall accuracy for $n \geq 6$. This reduction in error becomes more significant as n increases. Therefore the LH topology emerges as the best overall performer.

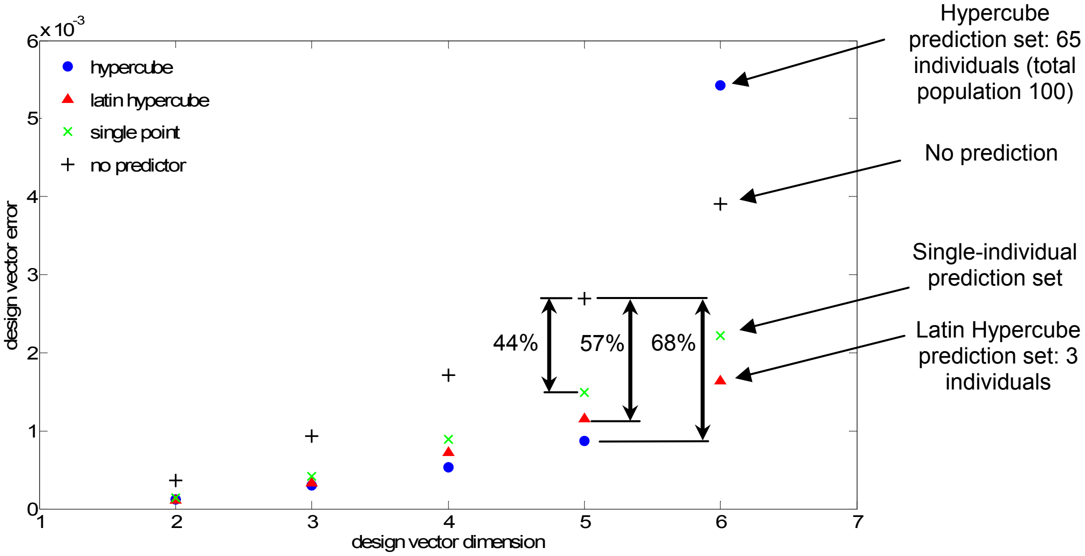


Figure 47. Design vector error for different topologies of the prediction set, with increasing design vector dimension. Although the hypercube performs well in small dimensions, it has the worst performance above a dimension of 5. The Latin hypercube emerges as the best overall performer.

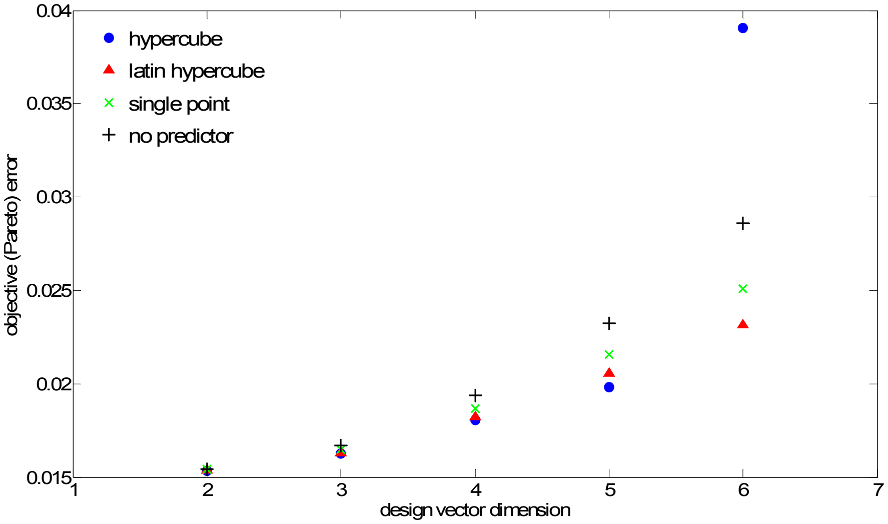


Figure 48. Pareto error for different topologies of the prediction set, with increasing design vector dimension.

Combining the techniques described in this and the previous section (5.1) an anticipatory population can be formed, for example, by tracking and forecasting the locations of the anchor and CTI points, and creating a Latin Hypercube of individuals around each forecast.

5.3 Elitism and the anticipatory population's impact on the solution in relation to its size

As the preceding discussions reveal, the anticipatory population size is quite small. For example if the CTI point is included and a Latin hypercube topology is used then the anticipatory population size is $3(m + 1)$, where m is the number of objectives. For a two-objective problem with a total population of 100 individuals the anticipatory population is less than 10% of the total number of individuals.

However, according to the FPS the anticipatory population's impact on the solution is intended to be much stronger than its relative size, since it attempts to discover the successive locations of the POS. If we assume that the forecast is successful enough, the only condition required in order for the anticipatory population to have its intended impact is that the evolutionary algorithm be *elitist*[6]. If the algorithm is elitist and the anticipatory individuals are closer to the new timestep's solution than the rest of the population then there is a large probability that at least some of them will be non-dominated. The algorithm's elitism will then ensure that they survive and their lifespan is long enough for them to attract the rest of the population towards the optimal solution.

This impact is less certain if a non-elitist evolutionary algorithm is used. In this case, parent selection criteria such as roulette wheel selection can be adjusted in order to give preference to fit anticipatory individuals. However the large degree of randomness that these criteria carry in conjunction with the anticipatory population's small size endangers the survival of the anticipatory individuals. A remedy could be to increase in some way the size of the anticipatory population in order to augment its effect on the gene pool. However this would entail additional computational cost without adding any useful information to the solution process.

As it was discussed in 2.4, D-QMOO is an intensely elitist algorithm and provides the anticipatory population with the opportunity to have its intended impact.

5.4 Coping with cases in which the objective's direction of motion is unpredictable

Anticipatory populations with shapes such as the Latin hypercube can increase the algorithm's performance and deal with errors in the forecast. However the objective temporal change pattern might be such that any kind of forecasting either has a very large error or is simply impossible. In this case techniques such as the ones described previously will not be helpful.

Here we discuss a technique for the creation of anticipatory populations, which can be used when the direction of the objective temporal change pattern cannot be forecasted. This approach complements the convergence-diversity balance which normally handles unpredictable objective motion.

Even if it is impossible to predict the direction towards which the optimal solution will move in the next timestep, information such as the magnitude of the optimal solution's motion can be used to create an anticipatory population. The idea proposed in this section is to estimate in some way the magnitude of the optimum's upcoming motion and to place anticipatory individuals at a radius similar to that magnitude, as shown in the sketch of Figure 49.

This approach does not direct the population towards a specific region of the design space, as the FPS normally does. It is essentially a diversity control mechanism, taking advantage of

[6] Repeating from 2.1, an elitist evolutionary algorithm does not, in general, allow a fit individual to be eliminated unless it is replaced by a fitter one.

knowledge from the objective's temporal change pattern (in the form of the magnitude of motion) in order to increase diversity more efficiently than in a completely random way. This technique shares common ground with some other methods. In Cobb's hypermutation for example (Cobb 1990), the mutation rate is increased when a change is sensed, spreading the population over a larger area of the design space in a random way. In Vavak et al. (Vavak, Fogarty & Jukes 1996, Vavak, Fogarty & Jukes 1997) the Variable-length Local Search (VLS) is presented, a 'learning' version of which seeks the best local search range given the change magnitude. Angeline (Angeline 1996) and Weicker and Weicker (Weicker, Weicker 1999) discuss different ways of controlling the mutation rate (which in turn controls the spread of the individuals and the population diversity), some of which use feedback from the objective or the design space.

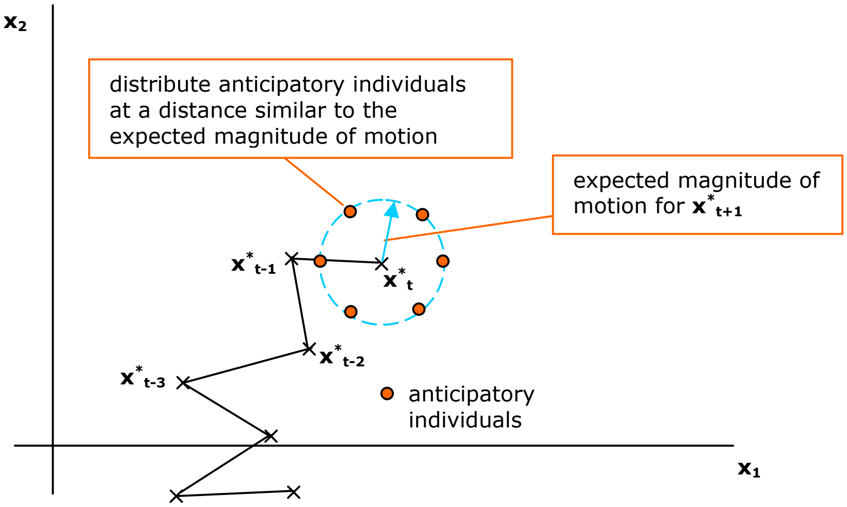

x_2
distribute anticipatory individuals at a distance similar to the expected magnitude of motion
expected magnitude of motion for x^*_{t+1}
x^*_{t-1}
x^*_t
x^*_{t-2}
x^*_{t-3}
● anticipatory individuals
x_1

Figure 49. The anticipatory individual's are distributed in some way (not necessarily spherical as shown here) at a radius similar to the expected magnitude of motion.

Apart from estimating the magnitude of motion, the biggest challenge this technique presents is how to populate the design hyperspace, given the desired radius. This is shown in the sketch of Figure 49 with a spherical distribution. Depending on the topology used, this method could potentially have dimensionality problems like the hypercube's.

For this reason we used a Latin hypercube shape, given its good performance from the previous section. The cube dimensioning is either done independently for each dimension, or using a common aggregate range for all dimensions.

Numerical experiments (Moving Peaks benchmark)

A nonstationary optimization problem with controllable predictability in the objective's direction but with a fixed magnitude of motion is modeled in the Moving Peaks benchmark. This problem is a good testing ground for the techniques just described.

The Moving Peaks is a single objective nonstationary problem initially proposed by Branke[7] (Branke 1999a, Branke 1999b, Branke 2006). A snapshot of the objective landscape can be seen

[7] The problem definition is given in the appendix.

in Figure 50. The evolutionary algorithm is called to discover a number of peaks that move around and change in height and width as time advances. Due to the height changes, the global optimum might not remain with the same peak so a successful algorithm must keep track of all peaks.

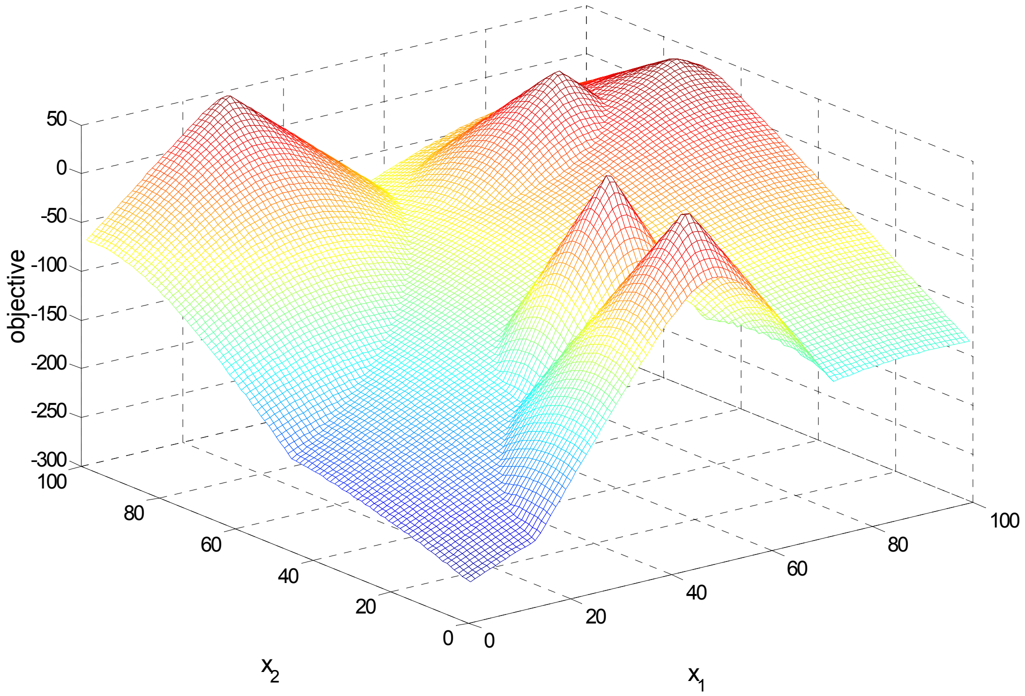


Figure 50. An instance of the Moving Peaks terrain. Maximization of the objective is sought. The peaks move around and change in height with every timestep. In this case the design vector is two-dimensional and the landscape has five peaks. In the experiments that follow, a five-dimensional problem with ten peaks is solved.

The parameters of Branke's scenario 2 (Branke 2006) are used, in order to be consistent with Branke and Schmeck's results (Branke, Schmeck 2003) and make comparisons. The basic parameters are repeated in Table 5.

Table 5. The Moving Peaks parameters used are as in Branke's scenario 2.

Moving Peaks parameters	
Number of peaks:	10
Number of design variables:	5
Evaluations per timestep:	5000
Correlation (lambda):	0.0 ÷ 1.0

The peaks move independently in a way that resembles (but is not strictly) a random walk. At each timestep the direction of motion is correlated to the previous timestep's by a user-defined correlation coefficient (the *lambda* coefficient in Table 5). When lambda is zero the direction of motion is completely random and follows a uniform distribution. When lambda is one, the peaks move along a straight line.

In Figure 51 we can see the comparative results for the *offline error* performance measure. The offline error is defined as the average distance of each timestep's best individual from the optimum (Branke 2006). A lower offline error indicates a higher performance. Results other than D-QMOO's are as reported by Branke and Schmeck (Branke, Schmeck 2003). Several algorithmic versions for D-QMOO are tested, as shown in the list of Table 6. Note that in one of the configurations, a simple linear extrapolation is used along with the Autoregressive model as a forecasting method. This forecast was used because when the correlation parameter lambda approaches one, the peaks move in a linear manner and a linear extrapolation has potentially better forecasting performance than an Autoregressive model.

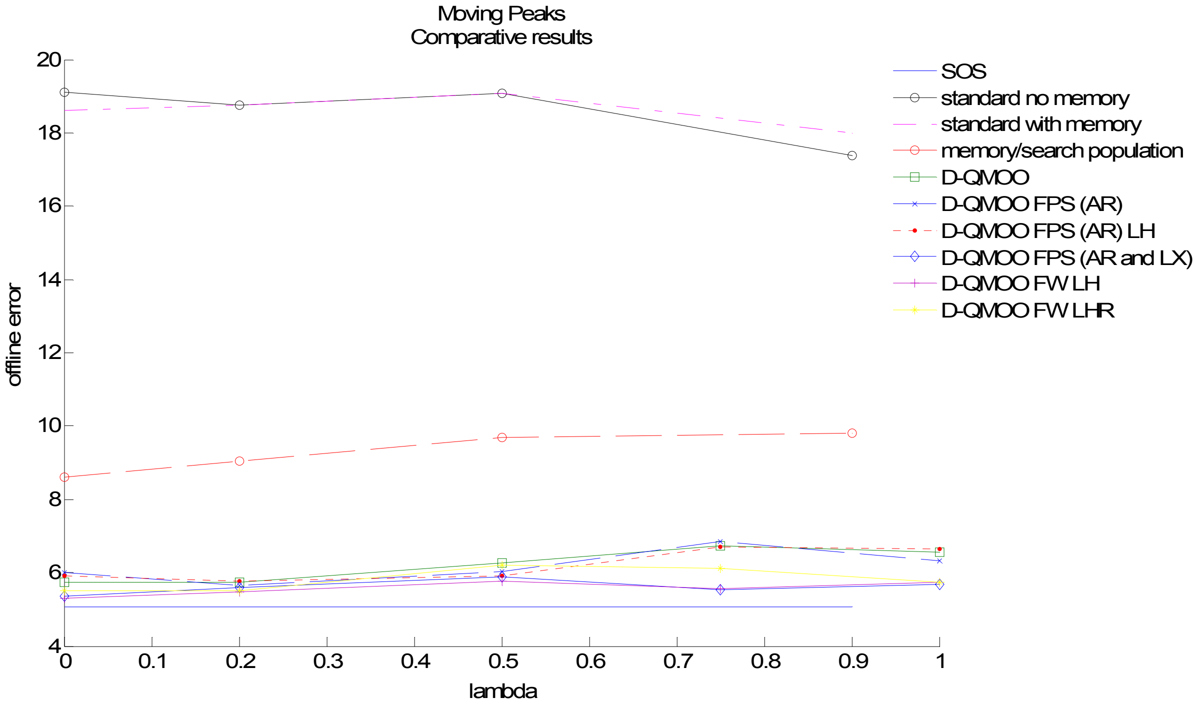


Figure 51. Moving Peaks benchmark. The offline error is shown as a function of the correlation coefficient lambda. A smaller offline error denotes a higher performance. Results are averages of 20 runs each. See Table 6 for a description of the various versions of D-QMOO.

Among the different versions of D-QMOO the technique proposed here (D-QMOO FW LH and LHR) performs best, especially when the correlation coefficient lambda is very small or zero and the optimum's direction of motion is completely unpredictable. It is also worth noting that the Feed-forward Prediction Strategy with the combination of Autoregressive and linear forecasting models (D-QMOO FPS (AR and LX)) has very good performance. This is especially true in the high-lambda regime when the optimum's motion is close to a straight line.

Table 6. Versions of the D-QMOO algorithm for the Moving Peaks experiments.

D-QMOO versions for the Moving Peaks problem	
D-QMOO	Base algorithm. Time changing optimization ability is provided only by the convergence-diversity balance.
D-QMOO FPS (AR)	D-QMOO with Feed-forward Prediction Strategy. The anticipatory population is composed of a single individual only, and a univariate Autoregressive model is used for forecasting.
D-QMOO FPS (AR) LH	D-QMOO with Feed-forward Prediction Strategy. The anticipatory population has a Latin hypercube shape, and a univariate Autoregressive model is used for forecasting.
D-QMOO FPS (AR and LX)	D-QMOO with Feed-forward Prediction Strategy. The anticipatory population is composed of a single individual only. Two forecasting models are used in parallel: a univariate Autoregressive model and a simple linear extrapolation. One anticipatory individual is created for each model.
D-QMOO FW LH	D-QMOO using the technique described in this section. The anticipatory population is distributed around each current optimum in the shape of a Latin hypercube, with each side separately dimensioned by the expected magnitude of motion in that direction.
D-QMOO FW LHR	D-QMOO using the technique described in this section. The anticipatory population is distributed around each current optimum in the shape of a Latin hypercube, with all sides having the same length.

Comparing with other algorithms, all versions of D-QMOO are close to each other and have significantly better performance than the various Genetic Algorithm versions (GA with no memory, GA with memory and GA with memory/search population). The fact that all configurations of D-QMOO, with or without the FPS, are close together and perform well is a demonstration of the effectiveness of the algorithm's diversity control technique which provides good performance throughout the whole range of correlation lambda.

The overall best performer is Branke's Self-Organizing Scouts algorithm (SOS, see Branke et al. 2000) which has slightly better performance than D-QMOO. It should be kept in mind though that the severely multi-modal nature of this problem is well suited to an algorithmic concept like SOS, where the population is dynamically and continuously re-arranged into sub-groups which follow specific optima. The only mechanism used to handle multi-modality in D-QMOO is its static clustering function, which was not designed for time-changing environments. Under this light D-QMOO's comparative performance is good. It would also be interesting to see how the SOS algorithm would perform if it was combined with the FPS.

5.5 Conclusion

The most important element of the FPS, the anticipatory population, was studied in this chapter in terms of its topology in the objective and design space. The first issue approached is which parts of the solution to include in the prediction set in order to cover the Pareto front. A prediction set consisting of the Pareto anchor points and an additional intermediate point is proposed, and found to increase performance in numerical experiments. The second question addressed is how to populate the neighborhood of the forecast in order to cope with forecast error and increase the chances of the anticipatory individuals discovering the true optimum. Anticipatory populations in the shape of a hypercube and a Latin hypercube are proposed. These shapes are dimensioned proportionately to the expected forecast error in order to surround the actual optimum with predictor individuals. The Latin hypercube emerges as the best overall performer, since it offers a satisfactory coverage while retaining an anticipatory population size of three individuals for each forecast even if the problem dimension increases, and hence it is economical in terms of objective function evaluations.

Finally a different technique is described, for the cases when the objective temporal change pattern is unpredictable in its direction of motion. An estimate for the magnitude of motion is used in order to place anticipatory individuals around the current optimum and increase diversity in an efficient way. This technique, and other algorithmic versions of D-QMOO are favorably compared against other algorithms using the Moving Peaks benchmark. The good comparative performance of the various D-QMOO versions in these numerical experiments also demonstrates the effectiveness of the algorithm's diversity control method.

6 Applications of Multi-Objective Evolutionary Optimization in Time-Changing Environments

Optimization problems of a nonstationary nature can be found across a wide range of disciplines. In this chapter two real-world applications of multi-objective evolutionary optimization in time-changing environments are presented. The methods proposed in the previous two chapters are applied and their behavior and performance is discussed.

The practical problems studied here are interestingly diverse in their nature. In the first case, D-QMOO is used to track the optimal closed-loop gains for an industrial controller. In the second, the algorithm is used to discover efficient asset allocations for a financial portfolio optimization problem.

6.1 Discovery of control gains in a time-changing industrial controller design problem

The design of control laws, both in terms of controller architecture and in terms of control gains specification, largely depends on the system characteristics. Most engineering systems are in reality non-linear and time-changing; take for example a supersonic aircraft's flight dynamics, which change significantly throughout the different airspeed regimes it operates.

The simplest form of control law design comes with the simplification of the system into a linearized, time-invariant model around an operating point (see for example Belanger 1995). Instead of a linearized model, non-linear control techniques can be used (Isidori 1995), while the case of time-varying systems is addressed by the sub-discipline of *adaptive control* (see for example Astrom, Wittenmark 1994 and Narendra, Annaswamy 1989).

Evolutionary optimization has been used in various forms in the past for the design of control systems. The special case of time-varying systems warrants the application of time-changing evolutionary optimization. This can be done in a number of different ways; Annunziato et al. for example (Annunziato et al. 2001, Annunziato et al. 2004) use an artificial life-based algorithm to create control signals on-line instead of using a fixed controller architecture. The algorithm 'learns' the system's dynamics as they change in time, and the absence of a fixed control law allows for a large amount of flexibility in the control signals produced. This in turn enables the optimization of the desired performance metrics. On the other hand, Farina, Deb and Amato in their seminal paper on evolutionary multi-objective optimization (Farina, Deb & Amato 2004) proposed the on-line discovery of the optimal gains for a fixed controller architecture using a time-changing evolutionary optimizer. Other heuristic methods have also been used for the optimal design of controllers, such as Particle Swarm Optimization (Ghoshal 2004).

In this chapter, we explore the behavior of D-QMOO when applied to a controller design problem similar to the one proposed by Farina et al. (Farina, Deb & Amato 2004). We treat an industrial process (a waste incinerator) whose dynamics change in time. The goal is to discover the non-dominated front of controller gain combinations for a number of objectives which are usually conflicting.

6.1.1 System description

A block diagram of the incinerator system can be seen in Figure 52. The plant and controller transfer functions are (Farina, Deb & Amato 2004):

$$\begin{cases} S(s) = \dfrac{1.5}{50s^3 + a_2(t)s^2 + a_1(t)s + 1} \\ C(s) = K_d(t)s + K_p(t) + K_i(t)\dfrac{1}{s} = \dfrac{K_d(t)s^2 + K_p(t)s + K_i(t)}{s} \end{cases} \tag{6.1}$$

As we can see the plant has dynamic characteristics that change in time. This time-dependent nature is expressed by the coefficients *$a_1(t)$* and *$a_2(t)$*. Physically this expresses the effect of factors which are changing in time, such as the quality of the incinerated material or the capacity utilization of the incinerator.

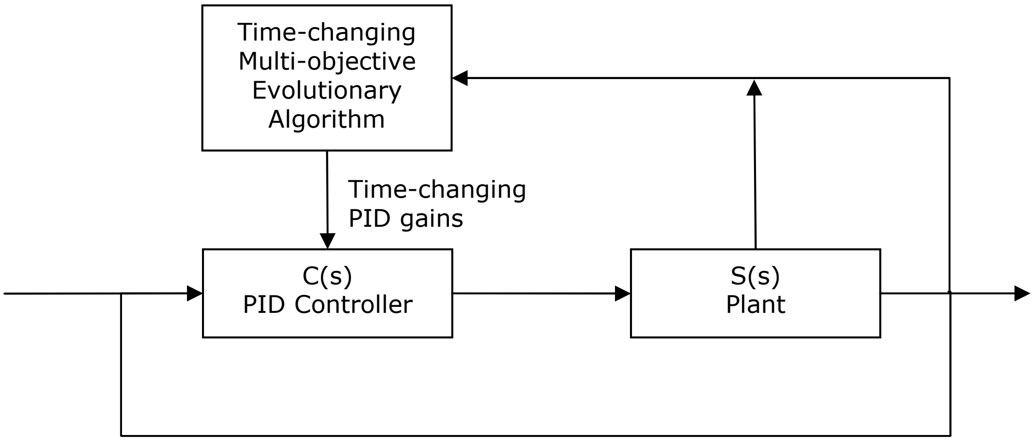


Figure 52. System model with time-changing optimal gain calculation.

A Proportional-Integral-Derivative (PID) compensator controls the system. If the problem is approached in a classical, time-invariant way, then the time-changing coefficients a_1 and a_2 would be approximated with constant values selected in some rational way. For example their time-averaged values could be used, if they are known. The controller would then be designed (i.e., the PID gains would be selected) on the time-invariant model using well-known classical design techniques (see for example Belanger 1995).

Here the problem is approached in its nonstationary form. The transfer function is allowed to change in time, and hence the optimal PID gains may also change each time the plant dynamics change. A time-changing evolutionary optimizer can be used to discover these varying optimal gains, as shown in Figure 52. In order to define optimality, one or more objectives need to be selected. In practice several objectives need to be optimized when the controller is designed. Some of these objectives are conflicting, producing a time-changing multi-objective problem.

This study serves as a conceptual model and demonstration of an envisioned practical application, where the time-changing multi-objective EA is used to specify on-line the control gains which provide a stable optimal controller for a time-varying system.

6.1.2 Problem formulation

There are several different criteria that can be used in controller design. In state-space design (for example in the Linear-Quadratic Regulator problem) a combination of state-induced and control-induced costs are used as objectives. In this context, the dynamic characteristics of the closed-loop system and specifically some of the step-response characteristics are used as optimization criteria.

Some of the step-response characteristics, which can be seen in Figure 53, are:

- *Rise time R*: the time it takes for the response to go from 10% to 90% of its final value.
- *Overshoot O*: the ratio of the maximum value the response attains to its final value.
- *Settling time ST*: the time it takes for the response to settle within 1% percent of its final value.

All three criteria are to be minimized (Farina, Deb & Amato 2004). This application serves as a simple demonstration for the use of a time-changing multi-objective optimizer. This problem can also be treated in a more complex way; for example, additional performance measures can be used (see Pedersen, Yang 2006 for a treatment of the static PID design problem).

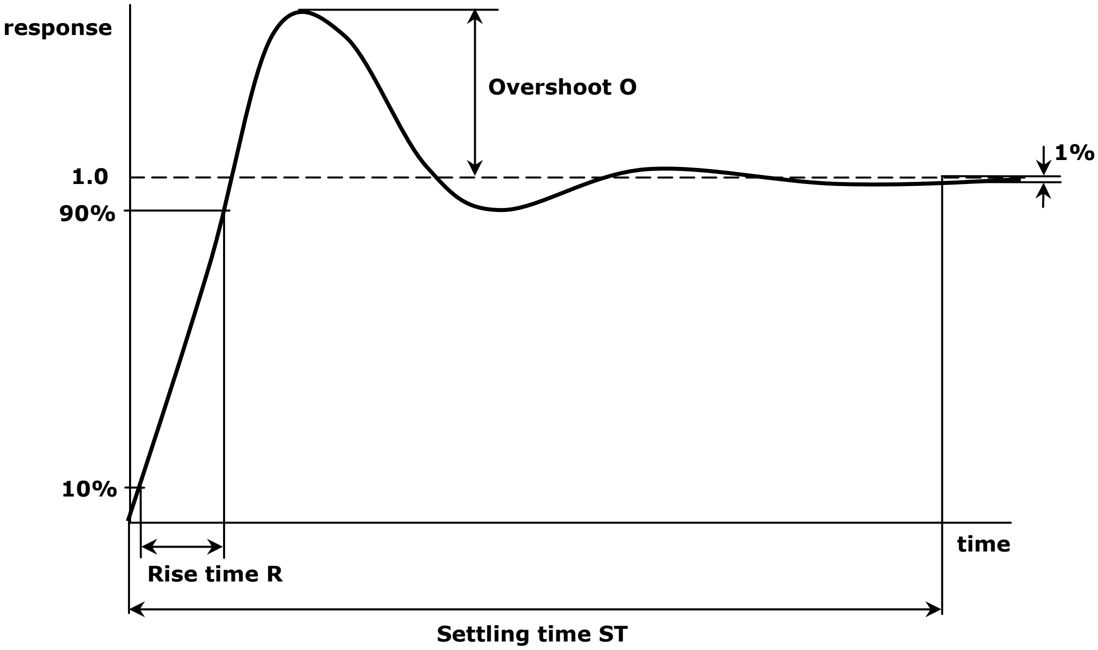


Figure 53. Step response performance measures.

In order to perform numerical experiments we use the two-objective problem for the rise time *R* and the overshoot *O*. This is an interesting problem since these two performance measures are usually conflicting. The basic elements of the optimization problem are presented next.

Time-dependent parameters

Recalling definition (6.1), the time-changing nature of the problem is dictated by the form of the coefficients *a₁(t)* and *a₂(t)*. In this application *a₁(t)* and *a₂(t)* vary in time according to the following relations:

$$\begin{cases} a_1(t) = \overline{A_1} + A_1 f(t) \\ a_2(t) = \overline{A_2} + A_2 f(t) \end{cases} \tag{6.2}$$

with *f(t)*:

$$f(t) = \sin\left(\frac{\pi t}{18}\right) \tag{6.3}$$

Design vector

The time-varying design vector is composed of the PID gains. In this context, the derivative gain is kept constant at a value of $K_d = 8.3317$ while the integral gain K_i and the proportional gain K_p form the design vector:

$$\mathbf{x}(t) = \begin{pmatrix} K_p(t) \\ K_i(t) \\ K_d = 8.3317 \end{pmatrix} \tag{6.4}$$

Constraints

The first constraint for this problem is closed-loop stability since any candidate controller design should result in a stable system. In our formulation we have included a stability constraint in the form of a requirement for the closed-loop poles of the resulting controller-plant combination to have a negative real part (Figure 54). Any solution with a positive real part of a closed-loop pole is considered infeasible. The Superiority of Feasible Points with Initial Feasibility Exploration is used as a constraint handling method, as described in section 3.4.

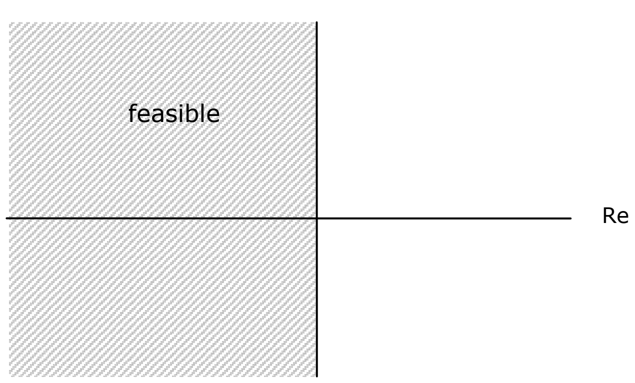


Figure 54. Closed-loop stability constraint.

A performance constraint is also introduced, restraining the overshoot O to a maximum value in order to avoid designing controllers that may be stable, but have a very high overshoot and take a long time to settle.

Optimization problem

Having defined its basic elements, the optimization problem can be seen in Table 7:

Table 7. Time-changing controller design problem definition.

Rise time – Overshoot problem

minimize	$\mathbf{f}(\mathbf{x}(t)) = [\text{Rise time } R(t),\ \text{Overshoot } O(t)]$
subject to:	closed-loop stability constraint performance constraint $O(t) \le \max O$
design vector:	$\mathbf{x} = \begin{pmatrix} K_p(t) \\ K_i(t) \end{pmatrix}$, PID gains (with K_d=8.3317).
$K_p \in [0.5, 15]$, $K_i \in [0.1, 1.0]$	

While solving the problem, the algorithm is called to discover the various heuristic rules that apply for the tuning of PID controllers – for example the fact that the proportional gain K_p reduces rise time – in order to populate the non-dominated front.

6.1.3 Results and discussion

The controller and dynamic system model were implemented in MATLAB and Simulink, and several different algorithmic and experiment configurations were explored. In particular, various options were examined for the front/cruft ratio, the type of forecasting method, the topology of the anticipatory population, the training period of the forecasting model, and the objective change frequency.

A set of characteristic results is presented here, in order to observe the solving behavior of the algorithm and see the effect of using techniques such as anticipation. The algorithmic and experimental parameters used for the results presented here are shown in Table 8.

As a comparative performance metric, a *normalized non-dominated volume* is used in this chapter. This metric is shown graphically in Figure 55, and has been proposed by Zitzler et al. (Zitzler, Laumanns & Thiele 2001, Zitzler, Thiele 1999) and implemented in QMOO by Leyland (Leyland 2002). The actual volume dominated by a Pareto front approximation is the un-hatched part of the control volume defined by the utopia and nadir points. The metric's value is the non-dominated hatched area, normalized by the control volume. Hence between two solutions the one with a smaller metric value is better, since it dominates a larger portion of the control volume. This is an appropriate scalar metric since it incorporates both the distance of the Pareto front approximation from some utopian trade-off surface, and its spread.

Table 8. Optimization parameters.

Time-changing controller design – Optimization parameters	
Maximum number of non-dominated (front) individuals:	50
Maximum number of dominated (cruft) individuals:	30
Nominal front/total population ratio:	50/80 = 0.625
Parent selection – $P_{\text{first rank}}$ (probability that a parent is selected from the front):	0.625 (equal to the front/total population ratio, providing a uniform parent selection probability over the whole population)
Front thinning method:	crowded thinning
Cruft thinning method:	combination of crowded and age thinning, with constant age criterion probability P_{birthday} at 50%
Constraint handling method:	superiority of feasible points with initial feasibility exploration (SFP-IFE)
Forecasting model (when used):	AR univariate model.
Anticipatory population (when used):	Anchor points + CTI point, with hypercube prediction set.
Objective change frequency:	150 objective evaluations per timestep.

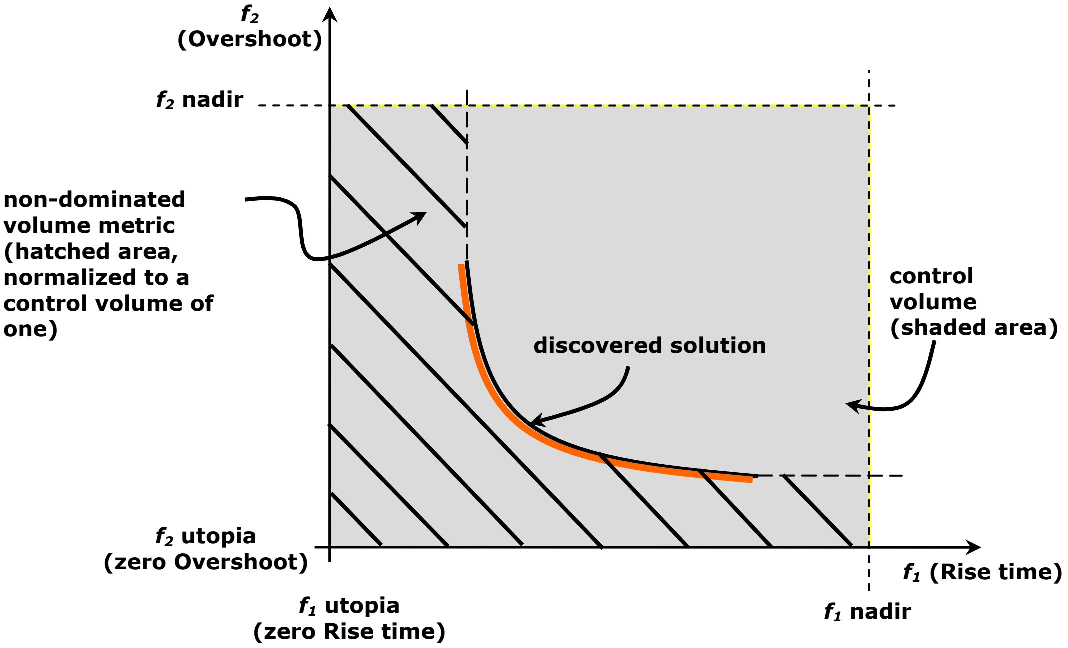


Figure 55. Non-dominated volume performance metric. The hatched area expresses the metric's value. Among two solutions the one with a smaller value is better.

A first observation reveals that the feasible region changes significantly in time. The shape of the feasible region in the variable space is shown for three time instances in Figure 56. It is evident that the problem is not convex in the variable space, and that the shape and size of the feasible region change substantially as time advances. The non-dominated Pareto-approximate controllers are also shown, in the objective space (left column) and in the variable space. Several of these optimal designs lie at the edges of the moving feasible region.

In Figure 57 the non-dominated volume time history is shown for the D-QMOO algorithm, with and without the use of an anticipatory population. In general the algorithm's performance is good, in the sense that it quickly adapts to the changing characteristics of the plant and produces a set of non-dominated designs in the R – O objective space. The anticipatory population's positive effect is not as dramatic as in the test cases of the previous two chapters, but it is especially evident in the periods when the solution changes direction, providing both better and less volatile performance in terms of non-dominated volume.

An interesting range of controllers populating the non-dominated front is discovered. In the design space the minimum overshoot anchor point oscillates to a much smaller extent than the minimum rise time anchor point, which utilizes the full range of the allowable gains as the system was changing in time. In Figure 58 and Figure 59 we can see a series of snapshots of the Pareto front in different time instants. Two numerical experiments are shown, with and without the Feed-forward Prediction Strategy. In Figure 60 we can see the objective and variable space locations of the individuals for the same time instants as in Figure 58. The shape of the front changes between a disjoint form with a gap near its middle (see for example the front at t = 533, Figure 58) and a continuous form (see for example the front at t = 440, Figure 59). Note that the plant dynamics change with a period of 36 timesteps, hence the figures do not cover full periods. The anticipatory population assisted performance more towards the minimum rise time part of the Pareto front, with the minimum *R* and the closest-to-ideal (CTI) anticipatory individuals helping discover better non-dominated solutions. It is evident from the objective and design space plots of Figure 61 that the minimum *O* anchor point moves in the design space much less than the rest of the front.

In Figure 62 and Figure 63 the forecasted and actual locations of the K_P design variable are shown for the minimum *R* and minimum *O* anchor points. Forecasting is better for the minimum *O* anchor point, however its range of motion is much smaller (in the order of 1.1 as opposed to a range of 10 for the minimum *R* anchor point – note the scale on the vertical axis). Hence, although its location is better forecasted, the minimum *O* anticipatory individual's contribution to the performance is much smaller, as we can see from the Pareto plots of Figure 58 and Figure 59.

In Figure 64 and Figure 65 we can see two instants of the non-dominated front with the closed-loop step response shown for some of the solutions, where the trade-off between the minimization of the rise time and the overshoot is evident.

In order to demonstrate the need for the discovery of the time dependent optimal gains, the feasibility of a controller designed for $t = 0$ is shown in Figure 66. This controller was designed by solving the static problem. Note that at $t = 0$, the time dependent factors are at their time-average values. Still this closed-loop system is not stable for a large part of the time, highlighting the need for a continuous re-design of the control gains by the time-changing algorithm.

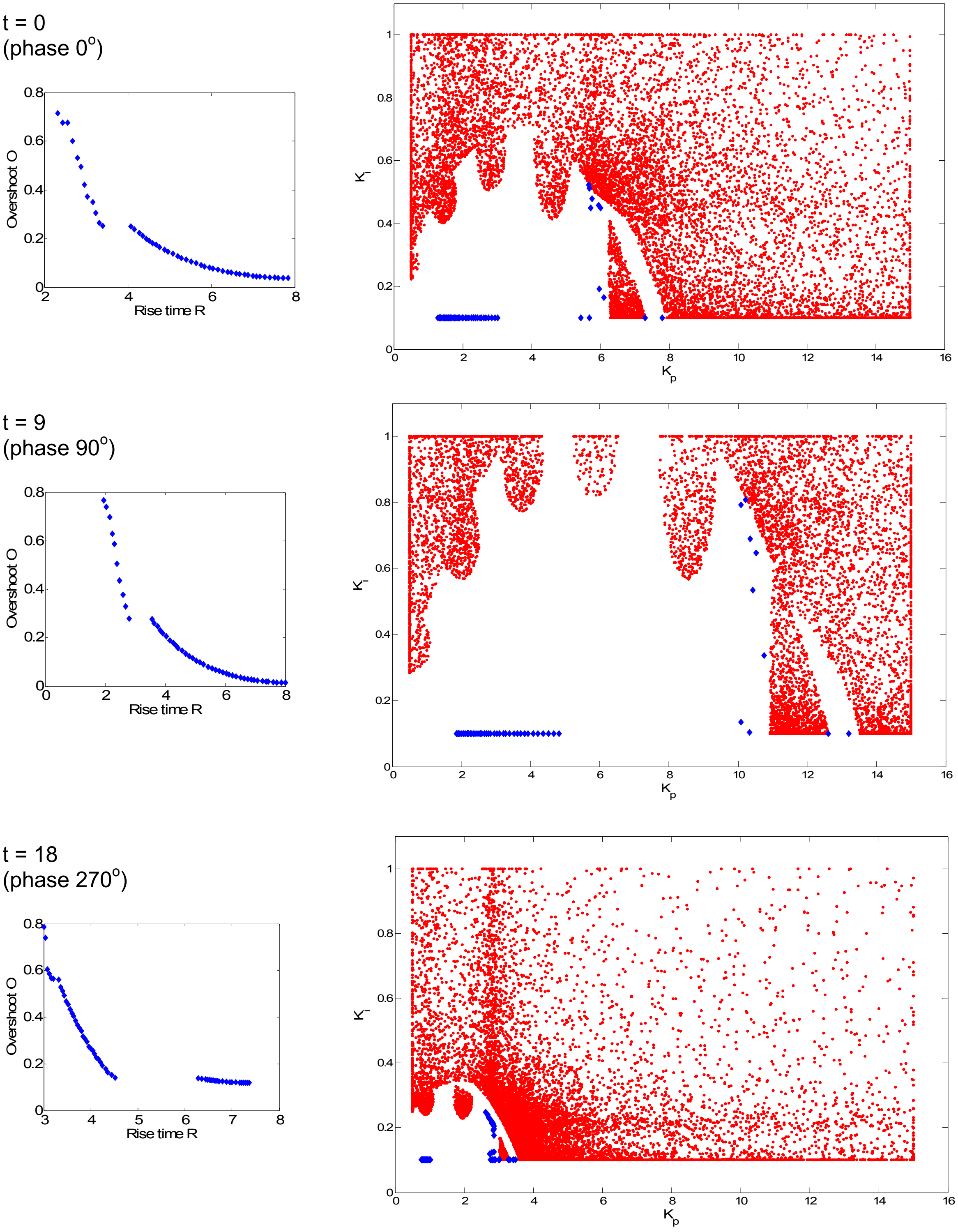


Figure 56. The feasible region at three different timesteps. The problem was run for 50,000 evaluations and all the infeasible individuals discovered were plotted in red (each red point belongs to the infeasible region). The final feasible non-dominated (Pareto) solutions are also plotted as blue diamonds, in the objective and the variable space. The change in the shape and size of the feasible region is evident, and so is the lack of convexity.

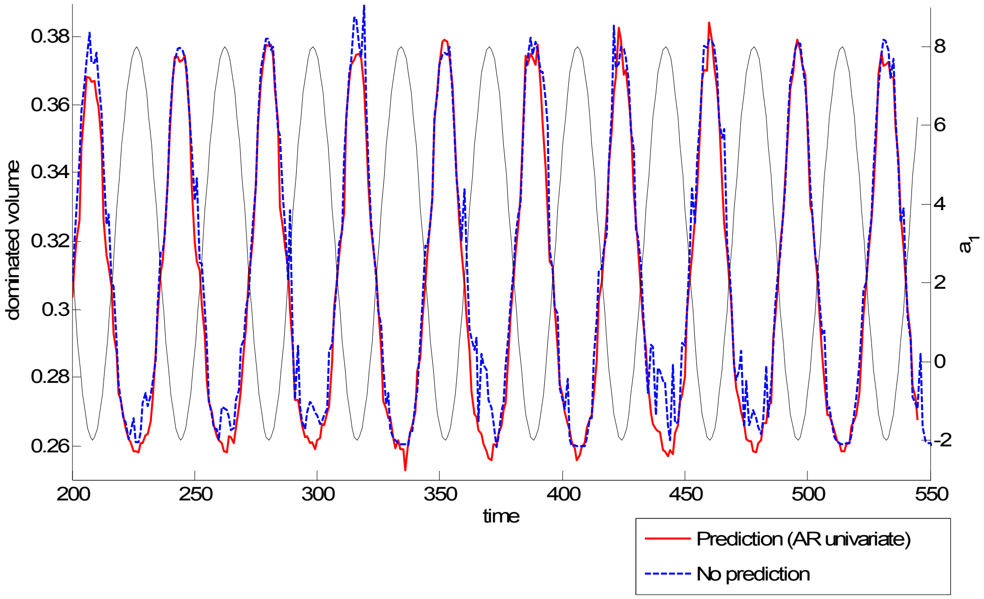


Figure 57. Non-dominated volume. The a_1 factor (from the transfer function in (6.1) and (6.2)) has been overlaid (black line) expressing the plant's time dependence.

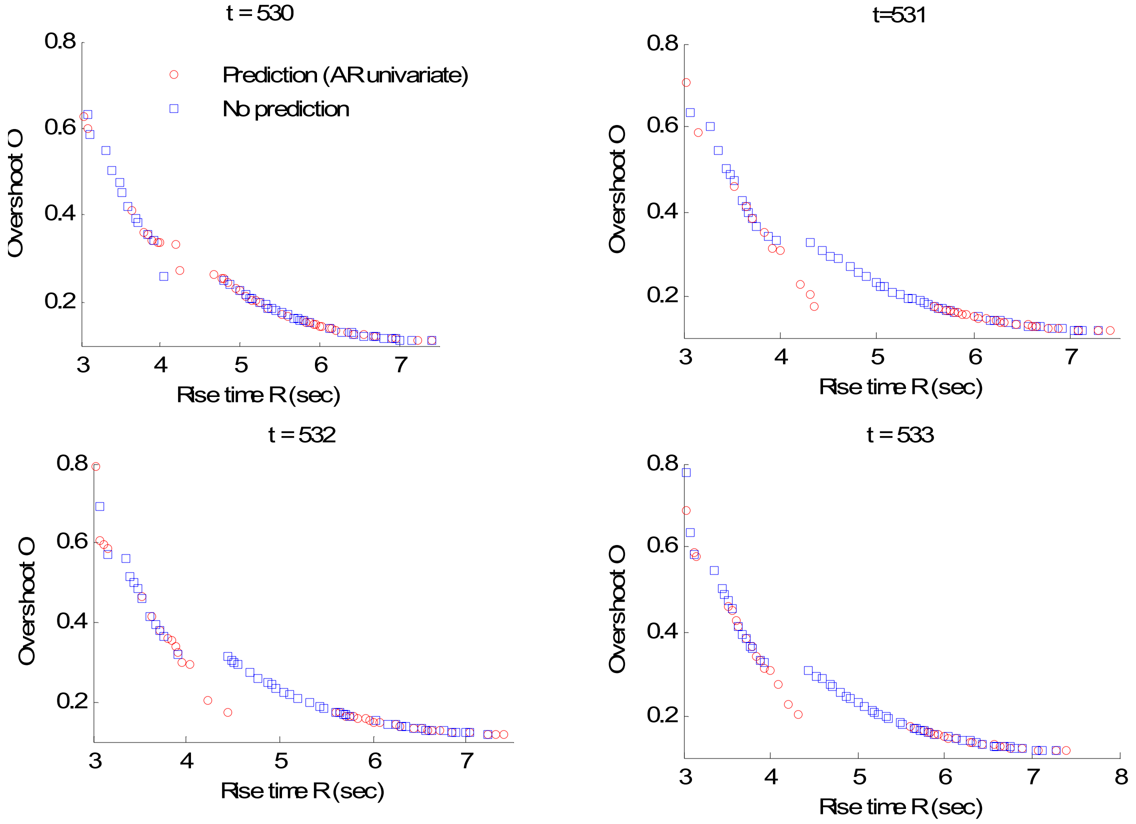


Figure 58. Non-dominated front snapshots (sequential).

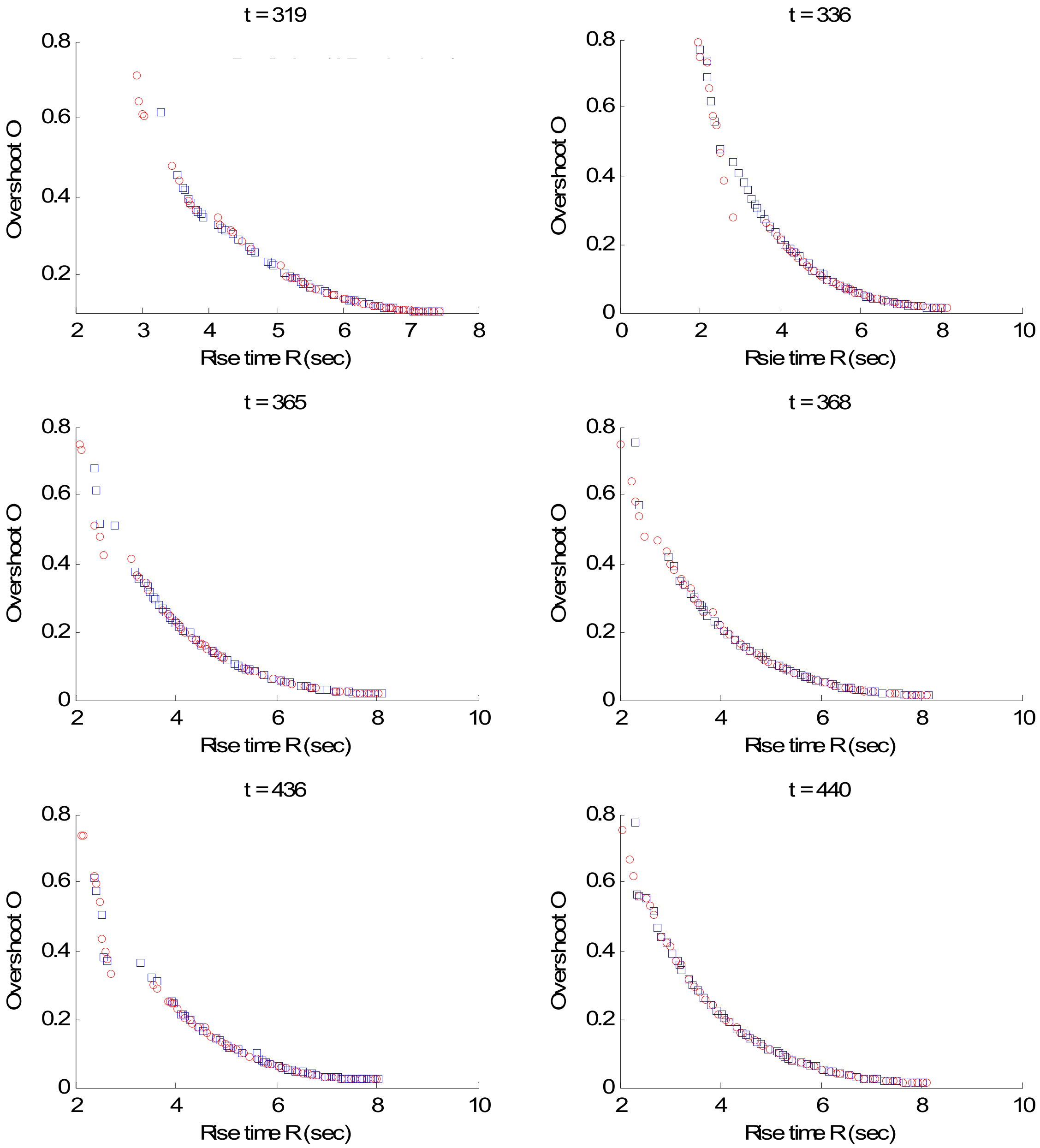


Figure 59. Non-dominated front snapshots.

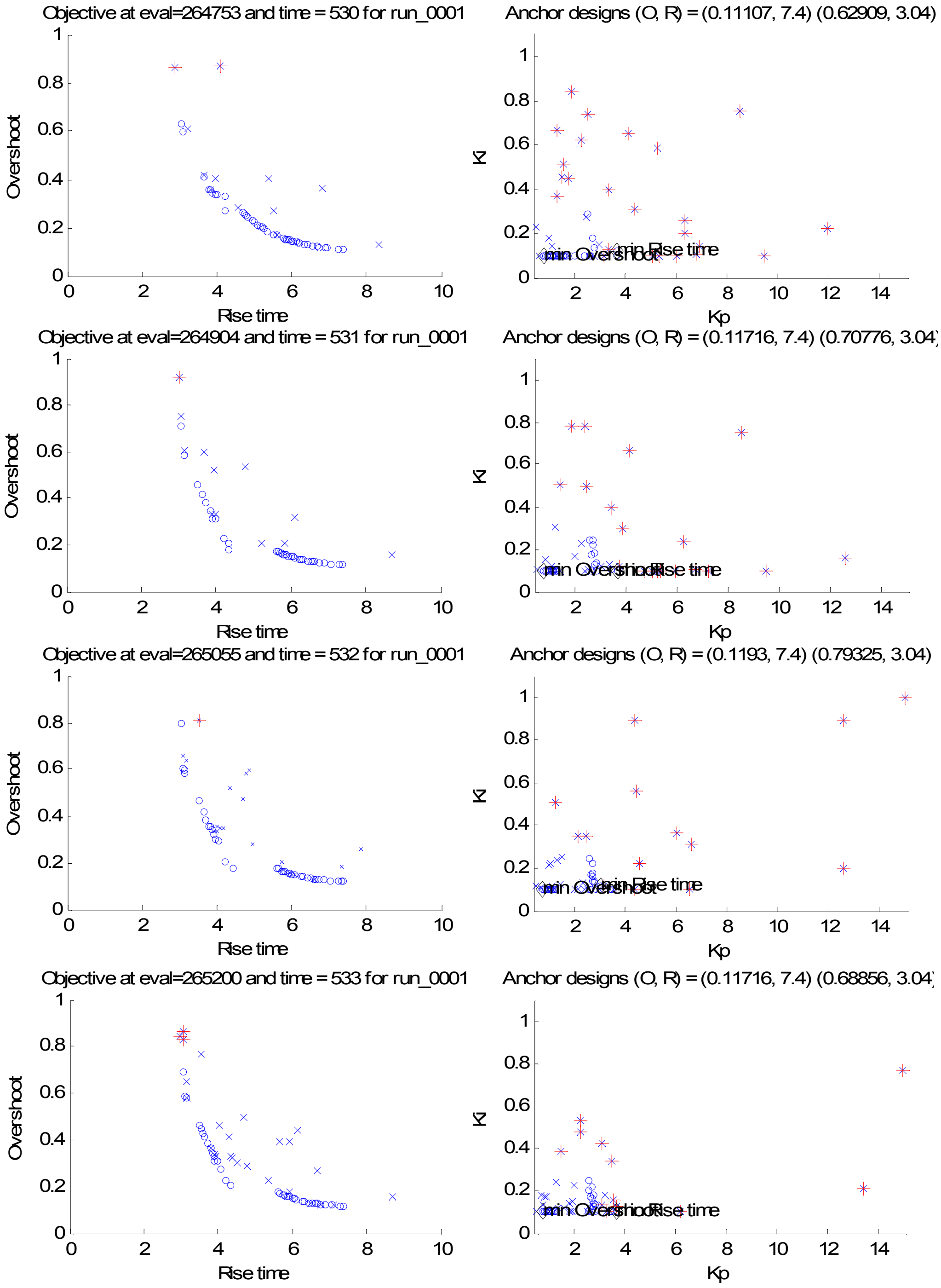


Figure 60. Objective and design space snapshots, using FPS (univariate AR model). Blue circles: non-dominated individuals, blue x's: dominated individuals, red crosses: infeasible individuals. The same timestep sequence as in the comparative plots of Figure 58 is shown (t=530 until t=535).

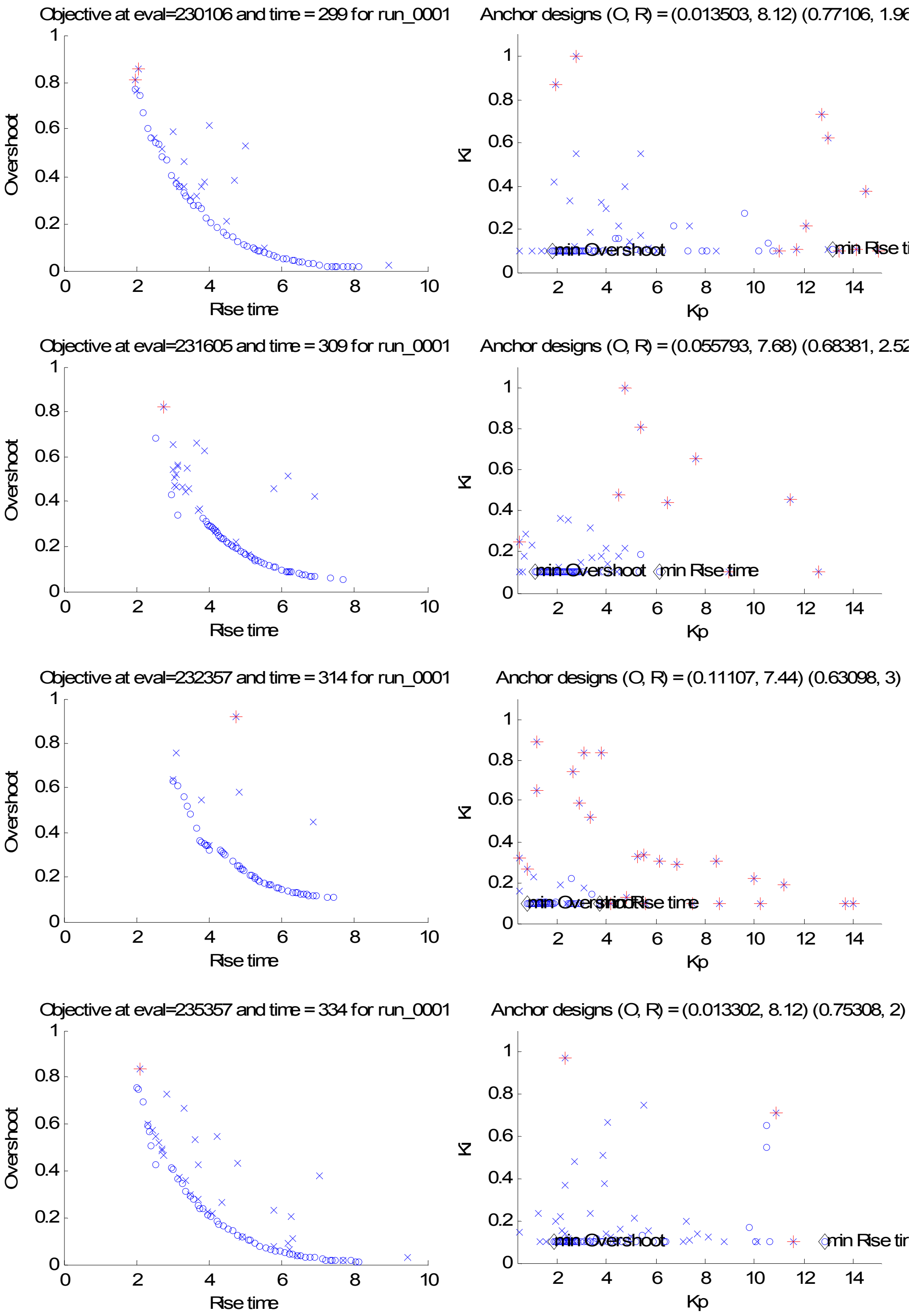


Figure 61. Objective and design space snapshots, using FPS (univariate AR model). Blue circles: non-dominated individuals, blue x's: dominated individuals, red crosses: infeasible individuals.

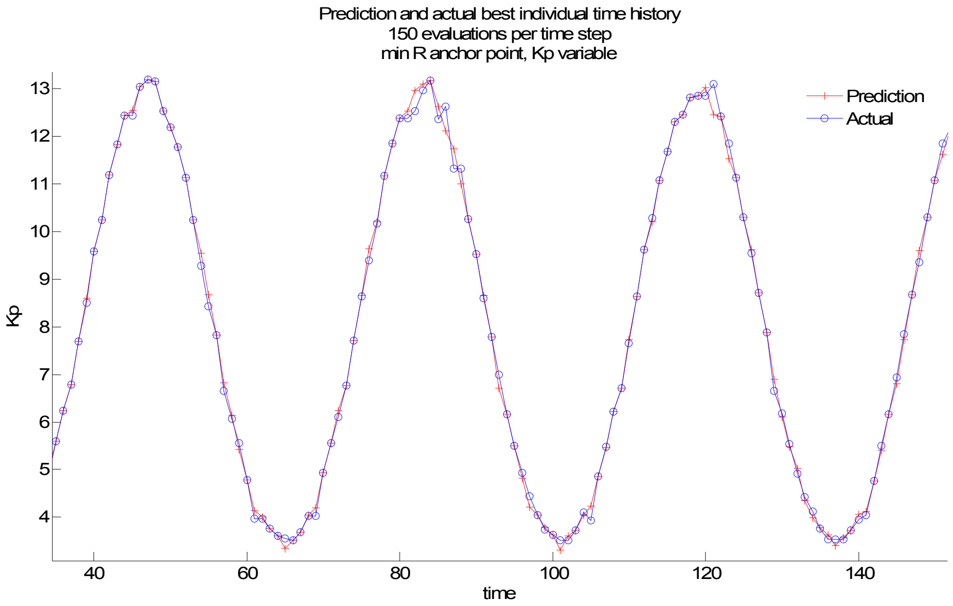


Figure 62. Forecasted and actual (discovered) location time history of the K_P proportional gain design variable for the minimum *R* anchor point.

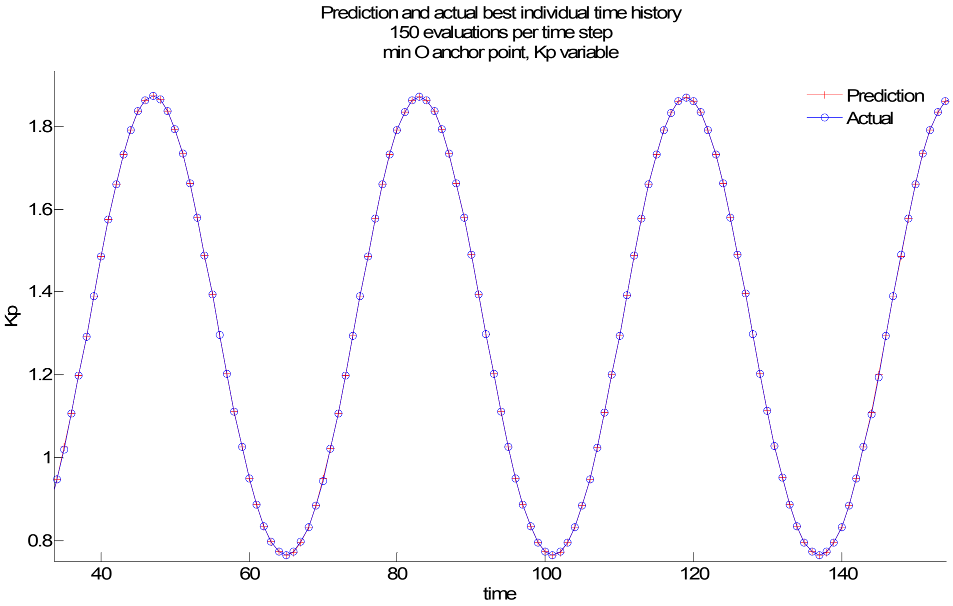


Figure 63. Forecasted and actual (discovered) location time history of the K_P proportional gain design variable for the minimum *O* anchor point.

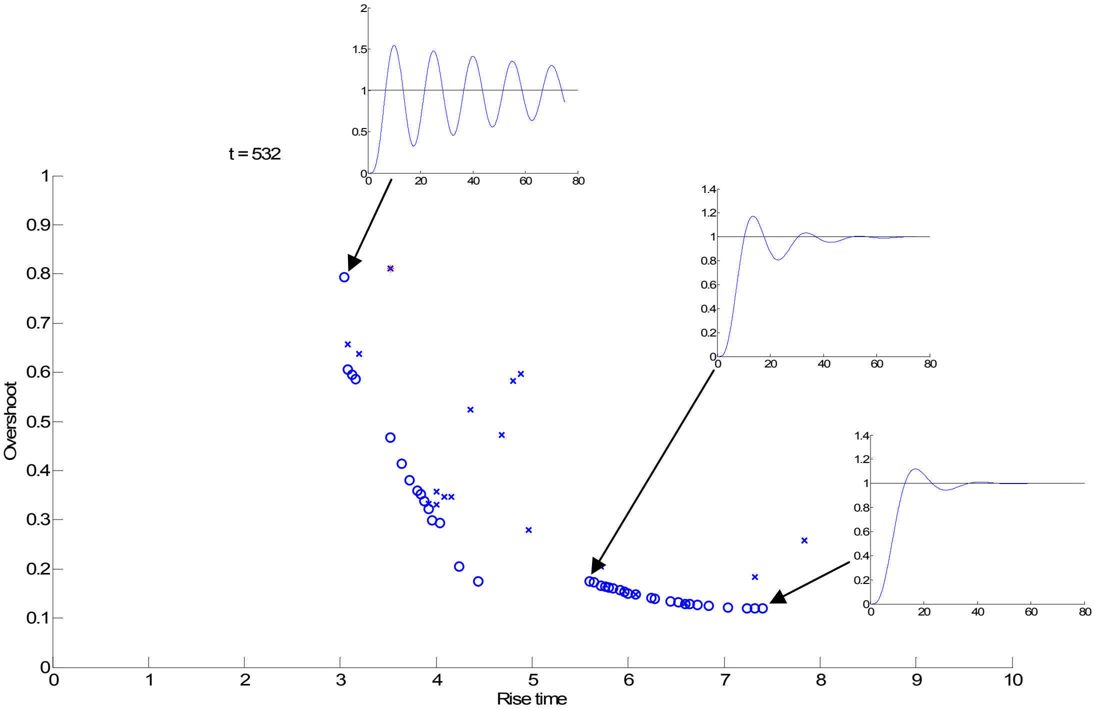


Figure 64. Different non-dominated controller designs along the front (t = 532, 150 evaluations per timestep).

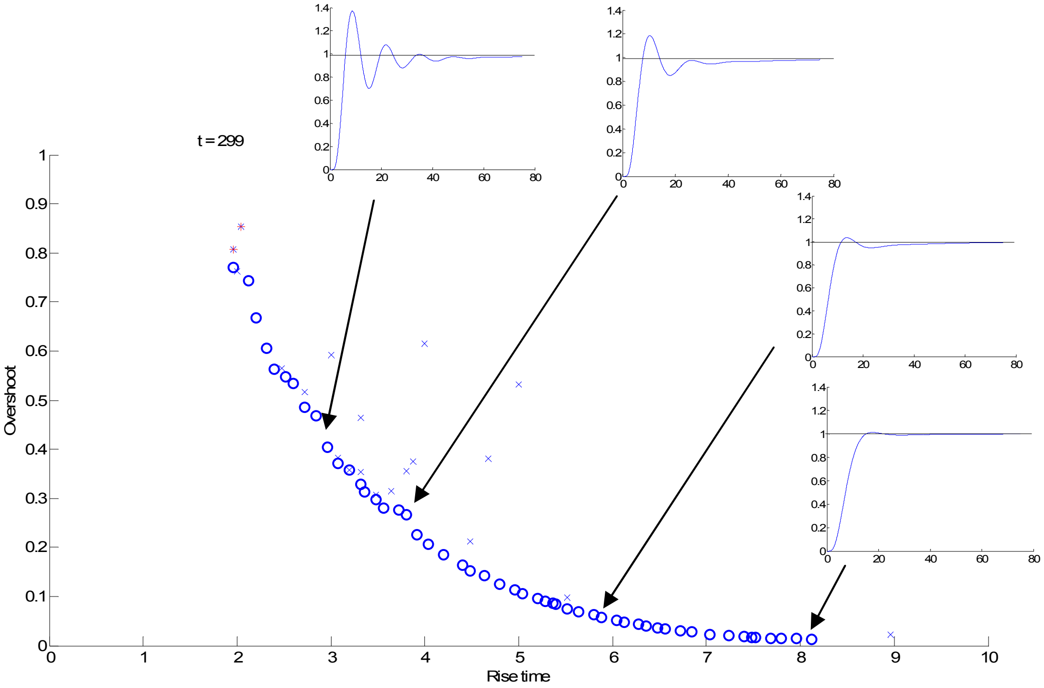


Figure 65. Different non-dominated controller designs along the front (t = 299, 150 evaluations per timestep).

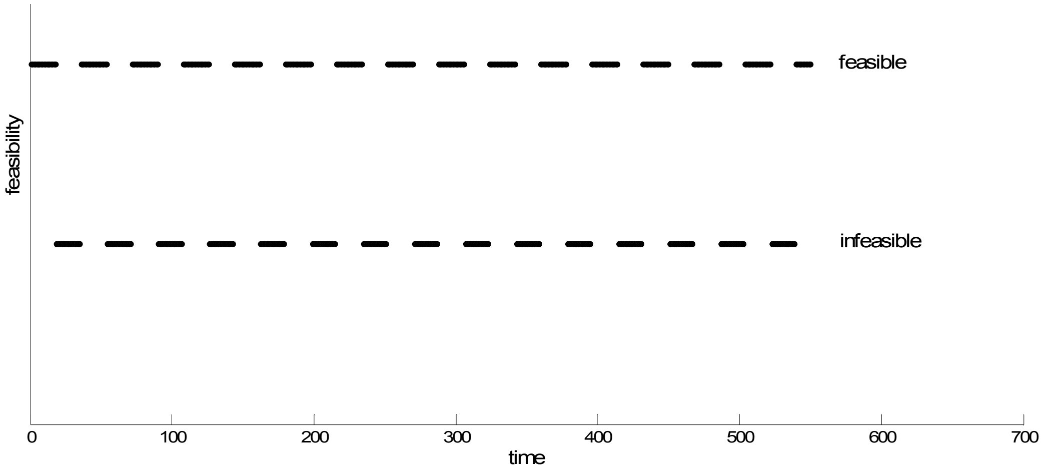


Figure 66. Feasibility of a controller designed for t = 0 by solving the static problem. This design was lying near the middle of the Pareto front at t=0. It is evident that the controller is infeasible about half the time.

6.2 Financial portfolio optimization – discovery of efficient asset allocations in a time-changing environment

An interesting multi-objective optimization problem of significant practical importance arises when one seeks efficient solutions for the asset allocation decision in a financial portfolio. In this section the application of multi-objective evolutionary optimization to time-changing asset allocation problems is described.

This work was done in collaboration with Dr. Frank Schlottmann, and was inspired by Schlottmann's work on the use of static evolutionary algorithms in credit risk portfolios (see for example Schlottmann, Seese 2005). A large part of this research is reported in detail in a separate work by the author (Hatzakis 2007). It is worth however to provide a summary description here since this is an interesting real-world problem which inspired the development of additional techniques for the application of the Feed-forward Prediction Strategy.

Assuming that we have a number of different investment opportunities (for example a number of different stocks we can invest on) and a fixed amount of capital, we wish to find the optimal asset allocation which provides the investor with the maximum profit. The actual return of an allocation is not known until after the decision has been made and the portfolio has been deployed to the market. Hence the maximum desired profit as an objective translates to a maximum expected return with a minimum amount of risk. Since risk and expected return are usually conflicting objectives the solution to the portfolio optimization problem is a Pareto set of non-dominated designs called the *efficient frontier* (Markowitz 2000), as shown in the sketch of Figure 67.

There are various ways to quantify risk and expected return. In this work the arithmetic average from a sample of past returns is taken as the measure of an asset's expected return. Risk is measured either as the sample standard deviation of the past returns, or as an α-quantile, for example as the 1-percentile, of past returns (Value-at-Risk or VaR, see for example Ruppert 2004). Both risk definitions have been treated in detail in (Hatzakis 2007). Here the Value-at-Risk problem will be discussed since it is a non-convex and possibly discontinuous problem (Pflug 2000) which benefits from an evolutionary approach.

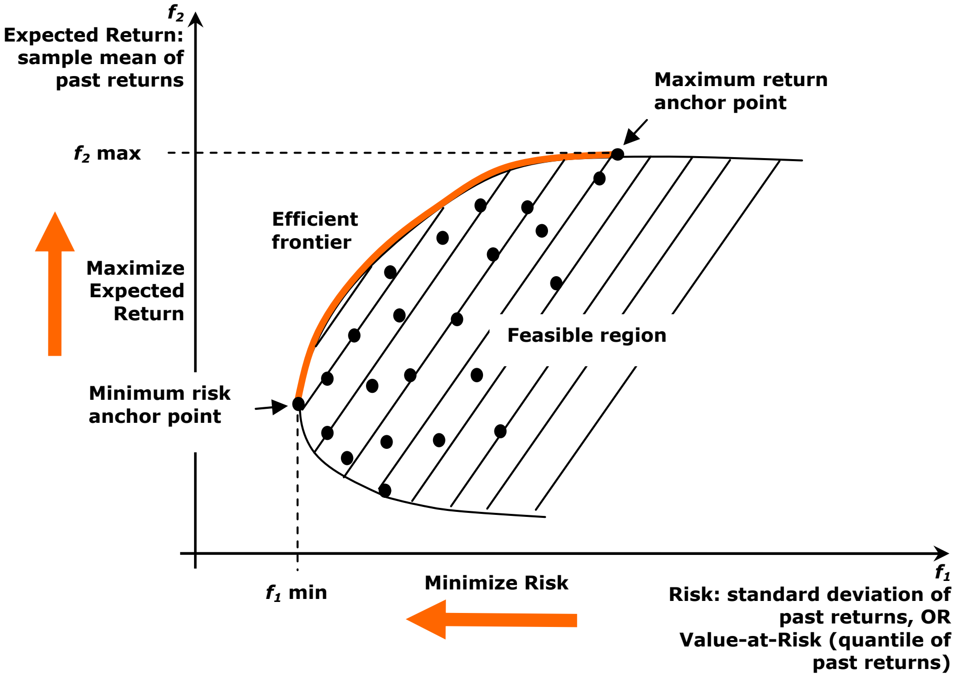

f_2
Expected Return: sample mean of past returns
Maximum return anchor point
f_2 max
Efficient frontier
Maximize Expected Return
Feasible region
Minimum risk anchor point
f_1 min
Minimize Risk
f_1
Risk: standard deviation of past returns, OR Value-at-Risk (quantile of past returns)

Figure 67. Efficient frontier in portfolio optimization (Hatzakis 2007).

The time-changing nature of the problem emerges from the fact that market conditions and asset performance change continuously. Each trading day[1] new information becomes available and is added to the historic sample of returns, changing the risk and return expectations. These in turn change the asset allocation of the optimal portfolios. At the same time, it may be the case that there is not enough computational power available to solve the portfolio optimization problem each day from scratch, given that investors have a trading space of hundreds or even thousands of assets which increase the size of the problem.

Hence the portfolio optimization problem is an application candidate for time-changing multi-objective optimizers such as D-QMOO. The goal of the time-changing algorithm is to find approximations of globally optimal portfolios quickly over time t.

Problem definition

Consider a space of $n \in N$ investment opportunities for an investor. The point in time when the investor makes a decision about the allocation of her capital is denoted by $t \in N$. At time $t > 1$, the history of asset returns up to t is:

$$R(t) = (r_i(k)),\ i = 1,...,n,\ k = 1,...,t\text{ - }1\ . \qquad (6.5)$$

In (6.5), $r_i(k)$ denotes the return of asset i at time k.

A portfolio at time t describes an allocation of the investor's capital to the investment

[1] In this context, the timestep is defined as one trading day. However it can be any other interval of time at which an investor makes an asset allocation decision.

opportunities. It is denoted by a real vector:

$$\mathbf{x}(t) = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \text{, where } \forall i \in \{1, \dots, n\} : x_i \geq 0 \tag{6.6}$$

In (6.6), x_i is the fraction of the available capital allocated to asset *i*.

The expected return for each portfolio is calculated directly using the sample statistics of the past history. This approach is shown graphically in the sketch of Figure 68.

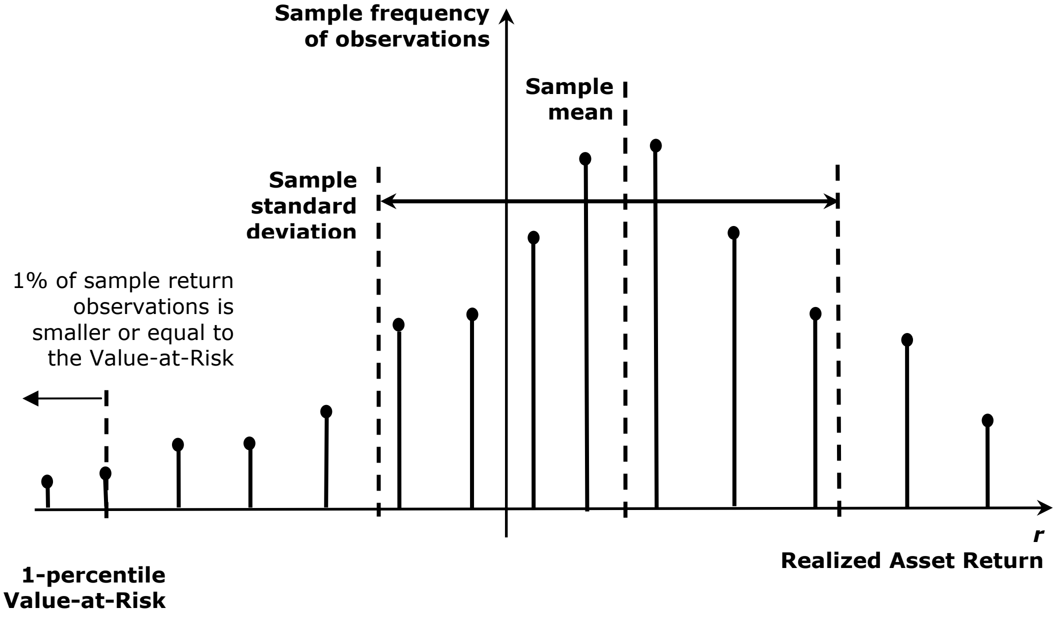

Sample frequency of observations
Sample mean
Sample standard deviation
1% of sample return observations is smaller or equal to the Value-at-Risk
r
Realized Asset Return
1-percentile Value-at-Risk

Figure 68. Statistical measures of risk and return (Hatzakis 2007).

In general, the Value-at-Risk can be calculated in a number of different ways (Beder 1995, Hendricks 1996, Stambaugh 1996). In this work a direct historical simulation is used: the past realizations of a portfolio are calculated, and the $\lceil \alpha t \rceil$ worse return value is selected as the VaR, where q_α is the desired VaR percentile. In practice a rolling time window of 250 trading days (one year) is used, and the one-percentile is taken as the VaR. Hence the $\lceil 2.5 \rceil = 3^{rd}$ worse portfolio return inside the time window gives a portfolio's VaR. This approach is justified on the grounds of accuracy, since it is the most direct method with the least number of assumptions. It is also a standard procedure for risk estimation in financial institutions like banks (Jorion 1997), and accepted by international banking supervision authorities (Basel Committee on Banking Supervision 2001).

It must also be stressed here that although we have a time-changing formulation with several timesteps, we are solving a 'portfolio optimization problem' in the sequential single-period sense as discussed by Markowitz in the third chapter of his book (Markowitz 2000), not in the utility-maximization dynamic programming sense.

The optimization problem for the expected return and Value-at-Risk is shown in Table 9:

Table 9. Portfolio optimization problem definition.

EXPECTED RETURN – VALUE -AT-RISK PROBLEM

Return measure: expected return from the un-weighted sample average

Risk measure: Value-at-Risk, 1-percentile, from historical simulation

minimize $\mathbf{f}(x,R(t),t)=\left[risk(x,R(t),t),-return(x,R(t),t)\right]^T$

where:

$$return(x,R(t),t):=E[R(t)x(t)]=\frac{1}{t-1}\sum_{k=1}^{t-1}\sum_{i=1}^{n}r_i(k)x_i(t)$$

$$risk(x,R(t),t):=VaR(x,R(t),t)=\begin{bmatrix}\text{3}^{\text{rd}}\text{ worse return of portfolio } x\\ \text{in the last 250 days (1 percentile)}\end{bmatrix}$$

subject to:

$l_B \le x_i \le u_B$, for $i=1,...,n$ (if $l_B=0$ and $u_B=1$, no short sales are allowed).

$$\sum_{1}^{\mathrm{n}}x_i=1$$

Heuristics for the creation of anticipatory populations in the portfolio optimization problem

The most important issue encountered when attempting to create anticipatory populations for this problem is the unpredictability of the optimal solution's motion in the variable space. This arises from the inherent lack of predictability that usually characterizes stock prices[2] and the market prices of other assets that are commonly included in an investment portfolio. Since the optimal solution is an asset allocation which directly depends on the behavior of asset prices, its motion is also very hard to predict. An example of this motion can be seen in Figure 69, where the minimum Value-at-Risk solution is shown as a time history of the variable values.

As a result, forecasting models such as the ones used earlier in this work are not very useful in this problem. Univariate or multivariate autoregressive models for example did not create accurate anticipatory populations. Forecast quality was in fact so bad that most of the times the anticipatory population was simply placed at the location of the previous timestep's optimal solution and had no effect on the discovery of the new optimum.

However, the portfolio optimization problem has some interesting qualities that allow the use of different heuristics for the creation of anticipatory populations. Specifically, each of the two anchor points on the Pareto front (see Figure 67) can be quickly approximated using an algebraic

[2] The question whether stock prices are predictable to any extent or not is an open one. For example, the books by Malkiel (Malkiel 2003) and Lo and MacKinlay (Lo, MacKinlay 1999) offer two contradictory views: according to the first, asset prices follow a random walk while according to the second, some form of long-term memory exists. Even by the second school of thought, however, and even in light of the existence of asset price prediction tools (usually proprietary algorithms of financial institutions), it would be impractical to attempt to use asset price prediction for the purposes of the Feed-forward Prediction Strategy.

heuristic in order to obtain anticipatory individuals:

- **Maximum return anchor point.** This anchor point is directly known from the asset returns' history: from the expected return objective definition (see Table 9) it is obvious that in order to maximize the expected return of a portfolio, all capital should be allocated to the asset with the highest expected return. Hence the maximum return solution at each timestep is a 100% allocation to the highest average return asset, and an anticipatory individual can easily be created for this anchor point.
- **Minimum risk (VaR) anchor point.** The minimum risk anchor point is the non-dominated solution with the smallest Value-at-Risk. As we said before this solution cannot be found analytically since VaR is a non-convex, discontinuous measure. However, an approximation for the minimum VaR solution is the minimum standard deviation solution (see for example Figure 70). In order to understand this, consider that if we had assumed a probability distribution for the returns in order to calculate the VaR (Stambaugh 1996), the minimum VaR solution would actually coincide with the minimum standard deviation solution. In our case a direct historical simulation is used, and the second provides an estimator for the first.

  It is relatively straightforward to discover the minimum standard deviation solution by solving a single-objective quadratic minimization problem, for which numerous algorithms exist. This way an anticipatory individual for the minimum risk anchor point can be created.

Using these techniques the population can be seeded with anticipatory individuals at each objective change by collecting the new market data (asset returns) and then performing the numerical tasks of finding the maximum return and minimum standard deviation solutions. These tasks can be executed using very little computational time, orders of magnitude less than the time it takes to go through one timestep with the evolutionary algorithm.

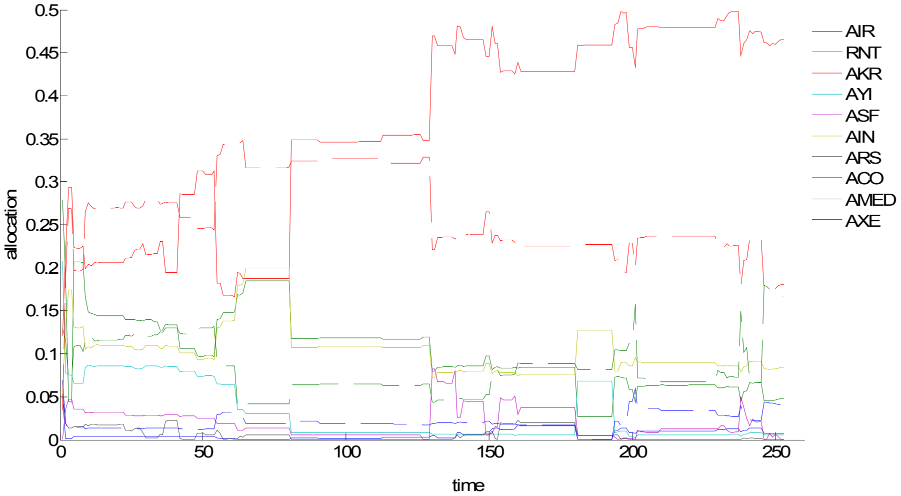


Figure 69. Time history of the minimum VaR anchor point. Each line is the value of a design variable in time (the capital allocation of a specific asset – ticker symbols shown on the legend). We can see that it is difficult to apply a forecasting method for the prediction of this motion.

According to these heuristics, the seed individuals are ‘anticipatory’ only in the sense of computational time, since knowledge from the current timestep is used to create them. However if they are created in an accurate way they can still serve their purpose of helping discover the new POS and increasing the algorithm’s performance. This leads us to the conclusion that in the context of this work the concept of *anticipation* is stronger than the concept of *forecasting*.

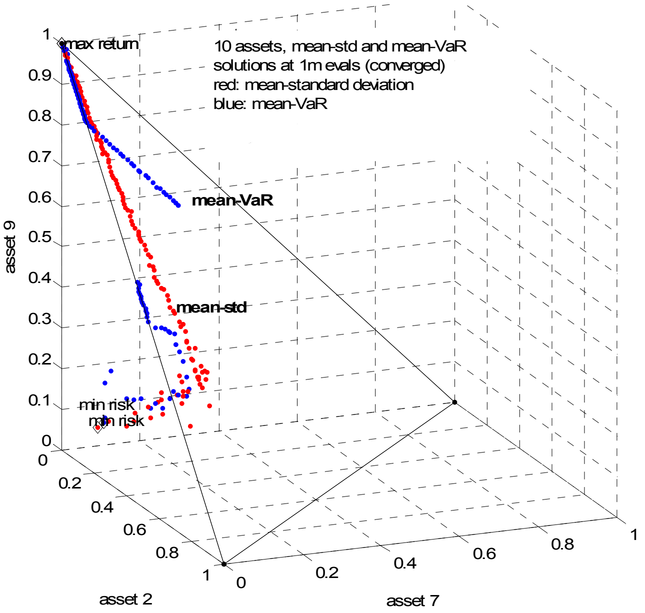


Figure 70. Optimal solution (efficient frontier approximation) in the variable space, plotted for three of the assets. Both the expected return-standard deviation and the expected return-VaR problems are shown. Asset 9 (AMED) has the highest expected return and hence the maximum return solution is centered on it. The difference between the two solutions caused by the different risk measure is evident. However the distance between the minimum standard deviation anchor point and the minimum VaR anchor point is small, and the first can be used as an estimator for the second.

Results and discussion

Sample results from the expected return-VaR problem are presented here, and the behavior of the heuristics introduced earlier is discussed. The sample dataset used is a basket of 10 stocks from the Standard & Poor’s 600 Small Cap Index. The stocks were randomly picked under the constraint that the underlying corporation has a positive profit margin and 10% or more estimated Earnings per Share (EPS) growth. A period of two years is used for the numerical experiments, from January 2nd 2004 until December 30th 2005. The portfolio adjustment period is one day: every day the closing stock prices are collected, the time series is updated and a new portfolio is designed which will deployed and produce an actual return the following day. The optimization starts at the beginning of 2005, halfway through the dataset, since one year of data behind each trading day is needed for the statistics calculation (recall the problem definition in Table 9).

In Figure 71 we can see the non-dominated volume time history for an objective change frequency of 1000 evaluations per timestep, where it is evident how the anticipatory population improves performance. In Figure 72 Pareto front approximations are shown. In this case the maximum return anticipatory individual is used for both runs, while the minimum risk anticipatory individual is only used for one of the runs. This anticipatory individual helps discover a better solution at the minimum risk neighborhood of the front.

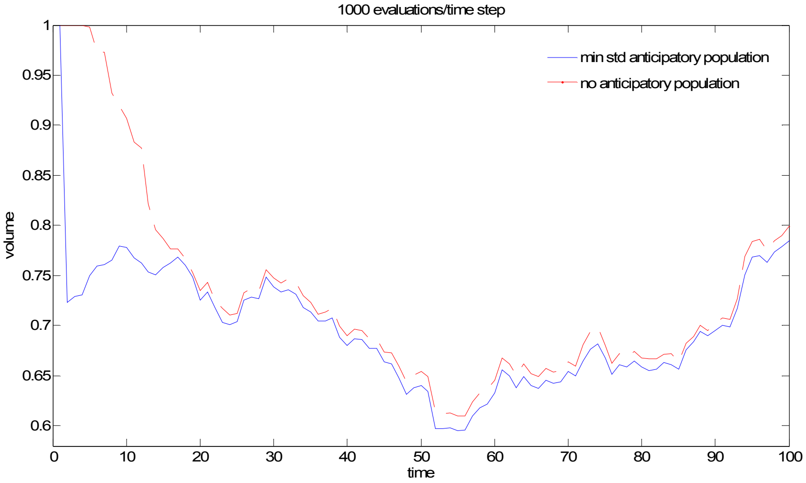


Figure 71. Non-dominated volume time history (smaller value = better performance). Solution of the time-changing portfolio optimization problem with VaR as risk measure, with and without anticipatory populations, at 1000 evaluations per timestep. The positive effect of anticipation is evident.

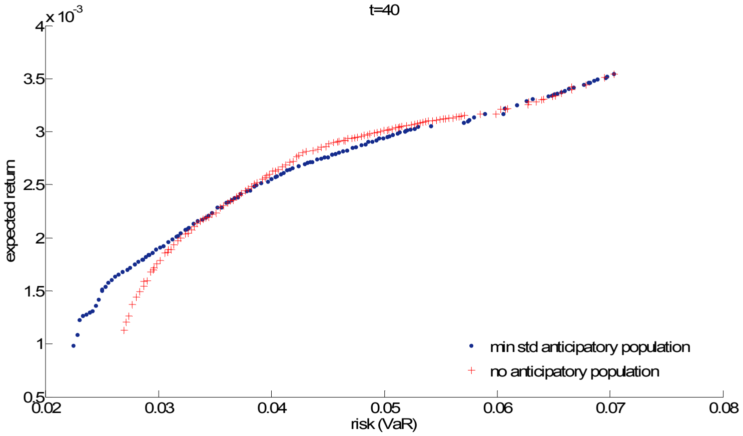


Figure 72. Efficient (Pareto) frontier approximations at various timesteps, with and without minimum risk anticipatory populations. The effect of the minimum risk anchor anticipatory individual is evident.

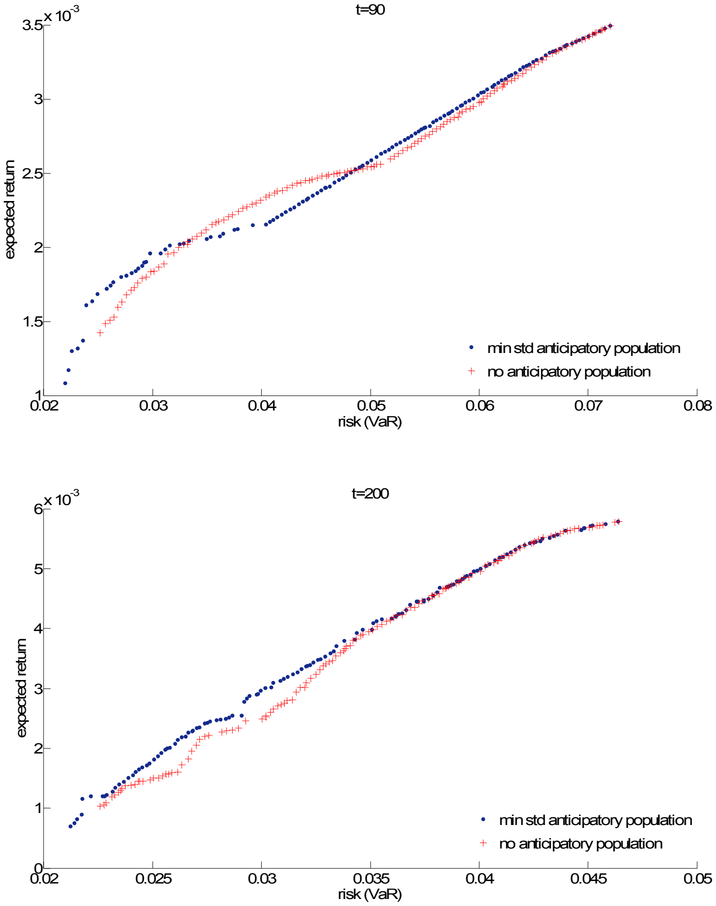


Figure 72 (continued).

6.3 Conclusion

Real-world applications of multi-objective optimization in time-changing environments are discussed in this chapter. An interesting fact is that the two problems studied are quite different in nature, the first dealing with an industrial controller design and the second being a financial portfolio optimization problem. These problems serve as examples of application areas for an algorithm such as D-QMOO.

Several of the techniques proposed in the previous chapters, such as the Feed-forward Prediction Strategy, are applied and their effectiveness is discussed in a practical application context. These techniques prove useful. At the same time however these real-world applications, and especially the portfolio optimization problem which is characterized by unpredictability in the optimal solution's motion, inspire the development of additional heuristics for the creation of anticipatory populations.

7 Conclusions and Future Directions

7.1 Constraint handling

The issue of handling constraints with evolutionary algorithms was explored first in order to provide D-QMOO with a constraint handling ability. This was needed for two reasons: First, most of the problems treated in this work are in some way constrained. Second, D-QMOO aims to provide a widely applicable optimization tool and hence needs to have a constraint handling ability without requiring the creation of customized constraint handling methods each time a new problem is solved. The constraint handling method created for D-QMOO is a hybrid variant of the Superiority of Feasible Points. In its final form the method is called Superiority of Feasible Points with Initial Feasibility Exploration (SFP-IFE). SFP-IFE works in a staged manner: it initially searches for the feasible regions of the design space and after it has discovered them it optimizes for the constrained objectives using a weighted version of the Superiority of Feasible Points. Constraint handling for evolutionary algorithms is by no means universally solved, and this method does not claim to be the best overall performer. However the combination of D-QMOO and SFP-IFE displays three very promising characteristics: Robustness, since it was able to successfully handle every problem, practical or benchmark, on which it was used. Good overall performance, which became apparent with comparative tests against other algorithms. Applicability, since by inception the SFP-IFE can be applied to virtually any problem which possesses a definition of constraint violation.

7.2 Time-changing problems with multiple objectives

An algorithmic architecture for the solution of nonstationary multi-objective problems was proposed in this work, with a combination of two elements at its core: An anticipatory population created with the help of a forecasting method and the time history of the optimum's motion (Feed-forward Prediction Strategy), and a balance between population convergence and diversity. Improving computational efficiency is the basic objective of the Feed-forward Prediction Strategy. The anticipatory population exploits structure in the optimal solution's motion, and helps the algorithm discover the new optimum when the objective changes in time using fewer function evaluations and increasing performance. On the other hand the balance between convergence and diversity ensures that the new optimum will be discovered even if the objective moves in an unpredictable way and the anticipatory population is not placed near the new optimum. This is done by preserving diversity in the population with the use of an exploratory group of individuals.

Initially it was shown how the anticipatory population increases solution performance. Subsequently the topology of the anticipatory population was treated. An anticipatory population consisting of the Pareto front extremities (anchor points) and an intermediate point (CTI point) was proposed in order to cover the Pareto front with anticipatory individuals. This anticipatory population provided adequate coverage when used in benchmark problems, and was also successfully used in some practical applications.

In order to reduce the effect of forecast error, coverage of the forecast neighborhood with more than one anticipatory individual was used. An anticipatory population in the form of a hypercube functioned well in problems of low dimension with four or less design variables. However a Latin hypercube shape emerged as the best overall performer since it consists of only three anticipatory individuals independent of the problem's dimension, and hence scales well in highly dimensional problems. In its final form an anticipatory population can be created, for example, using forecasts for the anchor points and the CTI point of the non-dominated front and populating each of the forecast neighborhoods using a Latin Hypercube topology.

When the landscape's direction of motion is completely unpredictable, a different approach was proposed where information on the expected magnitude of motion is used to populate the area around the current optimum. Conceptually this approach lies between the FPS and the various diversity control techniques used in the past for time-changing problems.

The work in time-changing multi-objective optimization produced two publications: in the first the general concept is described and some first results are given (Hatzakis, Wallace 2006a), and in the second the topology of the anticipatory population is discussed (Hatzakis, Wallace 2006b).

7.3 Applications

One of the best aspects of evolutionary optimization as demonstrated in this work is the width of practical applications these algorithms can handle. In this work three different real-world problems were treated with D-QMOO. A radar telescope array was optimized for the objectives of cost and performance. This is a static problem which helped evaluate D-QMOO's performance as a constrained multi-objective optimization tool. In chapter 6, two time-changing multi-objective problems were explored: the design of an industrial controller for a plant that changes in time, and a financial asset allocation problem under the objectives of risk and expected return. These applications are examples of areas in which algorithms such as D-QMOO can provide solutions. They also encouraged the further refinement of the methods proposed in this work and inspired the development of new techniques.

7.4 Future directions

This work leaves several branches for future investigation.

Creating anticipatory populations that cover the Pareto front

The current approach of an anticipatory population that uses the front's extremities and the intermediate CTI point is simple and has shown to be effective. However, non-dominated fronts can take various shapes and become severely disjoint. In these cases, a more detailed observation of the front's shape is required and a more complex anticipatory population may benefit solution performance. For example, a disjoint front can be separated into its components and an anticipatory population created for each component. The same goes for problems with a large number of objectives (5 or more); although conceptually they are not much different from two-objective problems, in practice they present their own intricacies (see for example Deb 2006) and require a special treatment in terms of the anticipatory population as well.

A conceptually different way of tracking the non-dominated front would be to create a parametric description of the non-dominated set as a shape in the design space and forecast the change of parameters in time, instead of tracking specific points as in the present work.

Using the anticipatory population for diversity control

The anticipatory population almost always increases diversity since the anticipatory individuals are placed at a distance from the existing population. This can lead to a unification between preservation of diversity and anticipation. The anticipatory population would then have as an additional explicit goal the increase of population diversity. This would influence the desired shape of the prediction set, for example by increasing the number of anticipatory individuals or by spreading them over a larger area. This way anticipation and control of diversity are simultaneously handled by the anticipatory population. The algorithm can thus become more efficient by decreasing the number of objective function evaluations it requires. The technique for dealing with unpredictable objective motion proposed in section 5.3, where the expected magnitude of motion is used to create an anticipatory population which is proportionately distanced from the current optimum, is a step towards this concept. However restructuring of the algorithm's architecture is required to ensure that the size of the cruft becomes explicitly associated with the anticipatory population. This way cruft size can be reduced if the anticipatory population provides enough additional diversity, retaining the cruft's independence from the forecast but reducing the required function evaluations.

Forecasting model

The choice of forecasting model strongly depends on the optimization problem, as we saw in the different applications studied. It would be useful to study the tradeoff between using a generally applicable model (such as a linear or second-order extrapolation) which is bound to be less accurate in most cases against a more sophisticated forecasting method which might be very accurate in some cases but fail completely in others. A more detailed study of the dependence between forecasting accuracy and solution performance would also be useful, taking in account the effect of techniques such as the hypercube prediction set presented in chapter 5. Finally, when forecasting models which require training are used (such as autoregressive models), it is worth examining the option of continuously re-training the model at each timestep to ensure that it is up to date with the objective temporal change pattern[1]. This approach is computationally more expensive since model training is usually time consuming but can provide a higher forecasting accuracy.

Switching cost

An interesting issue that emerges from the application of D-QMOO to practical problems is the fact that there is often a cost in adopting a new, different solution to an optimization problem. This cost may or may not be proportional to the distance between successive solutions. It would be interesting to take this switching cost in account in the optimization process and compare it with the cost of retaining an older suboptimal solution, in order to produce a final decision for each timestep. In parallel, if the switching cost has a relatively complex structure it could be included in the problem as an additional objective to be minimized.

Using FPS with different algorithms

The time-changing algorithmic architecture proposed in this work is abstract enough to be potentially used with other algorithms, apart from D-QMOO. It would be interesting to study, for

[1] The Moving Peaks benchmark provides a case where this would be especially useful, when the optimum bounces off the design space limits and changes its direction of motion.

example, the effect from using the FPS with an algorithm such as the Self-Organizing Scouts (Branke et al. 2000).

Applications

There is a range of disciplines where the techniques proposed in this work can be applied. Some stem from the applications examined in this work; namely, evolutionary optimization has many potential application areas in control which reach well beyond the gain-adjustment problem explored in chapter 6. A prominent one would be the use of a time-changing evolutionary algorithm as an 'on-line designer' of a controller for a system that changes in time, such as the artificial life approach by Annunziato et al. (Annunziato et al. 2001, Annunziato et al. 2004). No conceptual restrictions[2] on the type of control would then apply. A time-changing multi-objective evolutionary approach to state space control design would also be interesting.

The field of operations could also provide an interesting application ground for multi-objective time-changing evolutionary optimization, given that EAs have already been successfully applied to dynamic job-shop scheduling and traveling salesman problems (Branke 2002, Farina, Deb & Amato 2004). Scheduling and routing are examples of such applications, many instances of which are of a time-changing nature.

[2] Restrictions which apply, for example, on a PID controller or a linear feedback in the case of state-space design.

Bibliography

Adeli, H. & Cheng, N.-T. 1994, "Augmented Lagrangian Genetic Algorithm for Structural Optimization", *Journal of Aerospace Engineering,* vol. 7, no. 1, pp. 104-118.

Aguirre, A.H., Rionda, B.S., Coello Coello, C. A., Lizarraga Lizarraga, G. & Mezura-Montes, E. 2004, "Handling Constraints using Multiobjective Optimization Concepts", *International Journal for Numerical Methods in Engineering,* vol. 59, no. 15, pp. 1989-2017.

Akaike, H. & Nakagawa, T. 1972, *Statistical Analysis and Control of Dynamic Systems,* KTK Scientific Publishers, Tokyo.

Angeline, P.J. 1996, "Tracking Extrema in Dynamic Environments" in *Advances in Genetic Programming 2*, eds. P.J. Angeline & K.E. Kinnear, MIT Press, Cambridge, MA, pp. 89-109.

Annunziato, M., Bertini, I., Lucchetti, M., Pannicelli, M. & Pizzuti, S. 2004, "The Evolutionary Control Methodology: An Overview" in *Artificial Evolution*, 1st edn, Springer Berlin, Heidelberg, pp. 331-342.

Annunziato, M., Bertini, I., Pannicelli, M. & Pizzuti, S. 2001, "Evolutionary control and optimization: An industrial application for combustion processes", *Proc. EUROGEN* Athens, Greece, pp. 367-372.

Armstrong, J.S. (ed) 2001, *Principles of Forecasting: A Handbook for Researchers and Practitioners.*, Kluwer, Norwell, MA.

Arnold, D.V. & Beyer, H.-G. 2006, "Optimum tracking with Evolution Strategies", *Evolutionary Computation,* vol. 14, no. 3, pp. 291-308.

Astrom, K.J. & Wittenmark, B. 1994, *Adaptive Control,* 2nd edn, Addison-Wesley Longman Publishing Co., Boston.

Bäck, T. 1998, "On the behavior of evolutionary algorithms in dynamic environments", *Evolutionary Computation Proceedings*, pp. 446-451.

Bäck, T. 1996, *Evolutionary Algorithms in Theory and Practice,* Oxford University Press, New York.

Bäck, T. & Schwefel, H.-P. 1993, "An overview of evolutionary algorithms for parameter optimization", *Evolutionary Computation,* vol. 1, no. 1, pp. 1-23.

Bailey, M.W. & VerDuin, W.H. 2000, "FIPER: An Intelligent System for the Optimal Design of Highly Engineered Products", *NIST: Performance Metrics for Intelligent Systems Workshop* Gaithersburg, MD.

Basel Committee on Banking Supervision 2001, *The New Basel Capital Accord* [Homepage of Bank for International Settlements], [Online]. Available: http://www.bis.org/.

Beder, T.S. 1995, "VAR: Seductive but Dangerous", *Financial Analysts Journal,* vol. 51, no. 5, pp. 12-24.

Belanger, P. 1995, *Control Engineering: A Modern Approach,* 1st edn, Saunders College Publishing.

Bosman, P. 2005, "Learning, Anticipation and Time-Deception in Evolutionary Online Dynamic Optimization", *GECCO-2005 Workshop on Evolutionary Algorithms for Dynamic Optimization* Washington, DC.

Bounova, G. 2005, *Graph-theoretical Considerations in the Design of Complex Engineering Systems for Robustness and Scalability*, MIT.

Bounova, G. & de Weck, O. 2005, "Graph-theoretical Considerations in Design of Large Telescope Arrays for Robustness and Scalability", *46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference* Austin, Texas.

Box, G., Jenkins, M. & Reinsel, G. 1994, *Time Series Analysis - Forecasting and Control,* Prentice Hall, New Jersey, NJ.

Branke, J. 2006, *The Moving Peaks Benchmark* [Homepage of University of Karlsruhe], [Online]. Available: http://www.aifb.uni-karlsruhe.de/~jbr/MovPeaks/.

Branke, J. 2001, "Evolutionary Approaches to Dynamic Optimization Problems – Updated Survey", *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pp. 27-30.

Branke, J. 1999a, *Evolutionary Approaches to Dynamic Optimization Problems – A Survey*, Institute AIFB, University of Karlsruhe.

Branke, J. 1999b, "Memory Enhanced Evolutionary Algorithms for Changing Optimization Problems", *In Congress on Evolutionary Computation CEC99*, IEEE, pp. 1875-1882.

Branke, J., Kaussler, T., Schmidt, C. & Schmeck, H. 2000, "A Multi-Population Approach to Dynamic Optimization Problems", *Adaptive Computing in Design and Manufacturing 2000,* pp. 299-308.

Branke, J., Salihoğlu, E. & Uyar, Ş 2005, "Towards an analysis of dynamic environments", *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (Washington DC, USA, June 25 - 29, 2005)*, ed. H. Beyer, ACM Press, New York, NY, pp. 1433-1440.

Branke, J. & Schmeck, H. 2003, "Designing Evolutionary Algorithms for Dynamic Optimization Problems" in *Advances in Evolutionary Computing* Springer, pp. 239-261.

Branke, J. 2002, *Evolutionary Optimization in Dynamic Environments,* 1st edn, Kluwer Academic Publishers, Boston, MA.

Bui, L., Branke, J. & Abbass, H. 2004, *Multiobjective Optimization for Dynamic Environments*, Canberra.

Chipperfield, A.J. & Fleming, P.J. 1995, "The MATLAB Genetic Algorithm Toolbox", *IEEE Colloquium on Applied Control Techniques Using MATLAB,* , pp. 10/1-10/4.

Cobb, H. 1990, *An Investigation into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms Having Continuous, Time-Dependent Nonstationary Environments*.

Cobb, H. & Grefenstette, J.J. 1993, "Genetic Algorithms for Tracking Changing Environments", *Proceedings of the Fifth International Conference on Genetic Algorithms* , ed. S. Forrest, Morgan Kaufmann, San Francisco, CA, pp. 523-530.

Coello Coello, C. A. 2002, *Theoretical and numerical constraint handling techniques used with evolutionary algorithms: A survey of the state of the art*, EVOCINV.

Coello Coello, C. A. 2000a, "Constraint Handling using an Evolutionary Multiobjective Optimization Technique", *Civil Engineering and Environmental Systems,* vol. 17, pp. 319-346.

Coello Coello, C. A. 2000b, "Treating Constraints as Objectives for Single-Objective Evolutionary Optimization", *Engineering Optimization,* vol. 32, no. 3, pp. 275-308.

Coello Coello, C. A. & Mezura-Montes, E. 2002, "Handling Constraints in Genetic Algorithms using Dominance-based Tournaments", *Proceedings of the Fifth International Conference on Adaptive Computing Design and Manufacture (ACDM 2002)* Springer-Verlag, pp. 273-284.

Coello Coello, C. & Lamont, G. 2004, "Applications of Multi-Objective Evolutionary Algorithms" in World Scientific, pp. 3-5.

Cohanim, B.E., Hewitt, J.N. & de Weck, O. 2004, "The Design of Radio Telescope Array Configurations using Multiobjective Optimization: Imaging Performance versus Cable Length", *The Astrophysical Journal Supplement Series,* vol. 154, pp. 705-719.

Davidor, Y. 1991, "A Genetic Algorithm Applied to Robot Trajectory Generation" in *Handbook of Genetic Algorithms*, ed. L. Davis, Van Nostrand Reinhold, New York, NY, pp. 144-165.

Day, A.H. & Doctors, L.J. 1997, "Design of fast ships for minimal motions", *Proceedings of the Seventh International Offshore and Polar Engineering Conference*, pp. 677-683.

De Jong, K. 2006, *Evolutionary Computation: A Unified Approach,* MIT Press, Cambridge, MA.

De Jong, K. & Spears, W. 1993, "On the state of evolutionary computation", *Proceedings of the Fifth International Conference on Genetic Algorithms and their Applications*, ed. S. Forrest, Morgan Kaufmann, pp. 618-623.

Deb, K. 2000, "An Efficient Constraint Handling Method for Genetic Algorithms", *Computer Methods in Applied Mechanics and Engineering,* vol. 186, no. 2/4, pp. 311–338.

Deb, K., Pratap, A., Agarwal, S. & Meyarivan, T. 2000, *A Fast and Elitist Multi-Objective Genetic Algorithm: NSGA-II.* Technical Report No. 2000001. Kanpur Genetic Algorithms Laboratory (KanGAL), Kanpur, India.

Deb, K., Pratap, A. & Meyarivan, T. 2002, *Constrained Test Problems for Multi-Objective Evolutionary Optimization*, Kanpur Genetic Algorithms Laboratory (KanGAL), Kanpur, India.

Deb, K. 2006, "Towards Estimating Nadir Objective Vector Using Evolutionary Approaches", *GECCO 2006* Seattle, WA, pp. 643-650.

Deb, K. 2004, *Single- and Multi-Objective Optimization using Evolutionary Computation*. Technical Report No. 2004002. Kanpur Genetic Algorithms Laboratory (KanGAL), Kanpur, India.

Deb, K. 2001, *Multi-Objective Optimization using Evolutionary Algorithms,* Wiley.

Deb, K. & Agarwal, S. 1995, "Simulated binary crossover for continuous search space", *Complex Systems,* vol. 9, no. 2, pp. 115-148.

Deb, K., Rao, U. B. N. & Karthik, S. 2006, *Dynamic Multi-Objective Optimization and Decision-Making Using Modified NSGA-II: A Case Study on Hydro-Thermal Power Scheduling*, KanGAL, Kanpur, India.

Eschelman, L.J. & Schaffer, J.D. 1993, "Real-coded genetic algorithms and interval schemata" in *Foundations of Genetic Algorithms 2* Morgan Kaufmann, San Francisco, CA, pp. 187-202.

Farina, M., Deb, K. & Amato, P. 2004, "Dynamic Multiobjective Optimization Problems: Test Cases, Approximations and Applications", *IEEE Transactions on Evolutionary Computation,* vol. 8, no. 5.

Fogel, D. & Michalewicz, Z. 2002, *How to solve it: Modern Heuristics,* Springer-Verlag.

Fonseca, C.M. & Fleming, P.J. 1993, "Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization", *Proceedings of the Fifth International Conference on Genetic Algorithms*, ed. S. Forrest, Morgan Kaufmann Publishers, San Mateo, California, pp. 416-423.

Ghoshal, S.P. 2004, "Optimizations of PID gains by particle swarm optimizations in fuzzy based automatic generation control.", *Electric Power Systems Research,* , no. 72, pp. 203-212.

Goldberg, D.E. 1989, *Genetic Algorithms in Search, Optimization, and Machine Learning,* 1st edn, Addison-Wesley, Reading, MA.

Goldberg, D.E. & Smith, R.E. 1987, "Nonstationary function optimization using genetic algorithm with dominance and diploidy.", *Proceedings of the Second international Conference on Genetic Algorithms on Genetic Algorithms and their Application* , ed. J.J. Grefenstette, Lawrence Erlbaum Associates, Mahwah, NJ, pp. 59-68.

Grefenstette, J. 1992, "Genetic Algorithms for Changing Environments" in *Parallel Problem Solving from Nature 2*, eds. R. Manner & B. Manderick, North Holland, Amsterdam, pp. 137-144.

Gruninger, T. & Wallace, D. 1996, *Multimodal Optimization using Genetic Algorithms: An investigation of a new crowding variation and the relationship between parental similarity and the effect of crossover*. Technical Report 96.02, MIT CADLab.

Hamilton, J. 1994, *Time Series Analysis,* Princeton University Press, Princeton, NJ.

Harris, R. & Sollis, R. 2003, *Applied Time Series Modeling and Forecasting,* Wiley.

Hatzakis, I. 2007, *Multi-Objective Evolutionary Methods for Time-Changing Portfolio Optimization Problems*, Master of Science in Ocean Systems Management, Massachusetts Institute of Technology, Cambridge, MA.

Hatzakis, I. & Wallace, D. 2006a, "Dynamic Multi-Objective Optimization with Evolutionary Algorithms: A Forward-Looking Approach", *2006 Genetic and Evolutionary Computation Conference,* Seattle, WA.

Hatzakis, I. & Wallace, D. 2006b, "Topology of Anticipatory Populations for Evolutionary Dynamic Multi-Objective Optimization", *11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference* AIAA, Portsmouth, VA.

Hendricks, D. 1996, "Evaluation of Value-at-Risk Models Using Historical Data", *FRBNY Economic Policy Review,* no. April, pp. 39-70.

Hernandez Aguire, A., Botello Rionda, S., Coello Coello, C. A., Lizarraga Lizarraga, G. & Mezura-Montes, E. 2003, "Handling Constraints using Multiobjective Optimization Concepts", *Int. J. Num. Meth. Eng.,* pp. 1-6.

Homaifar, A., Qi, C. & Lai, S. 1994, "Constrained Optimization via Genetic Algorithms", *Simulation,* vol. 62, no. 4, pp. 242-253.

Horn, J., Nafpliotis, N. & Goldberg, D.E. 1994, "A Niched-Pareto Genetic Algorithm for Multiobjective Optimization", *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence* IEEE Service Center, Piscataway, New Jersey, pp. 82-87.

Isidori, A. 1995, *Nonlinear Control Systems,* 1st edn, Springer, London.

Jin, Y. & Branke, J. 2005, "Evolutionary Optimization in Uncertain Environments — A Survey", *IEEE Trans. on Evolutionary Computation,* vol. 9, no. 3, pp. 303-317.

Jorion, P. 1997, *Value at Risk,* 1st edn, Academic Press, San Diego.

Koziel, S. & Michalewicz, Z. 1999, "Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization", *Evolutionary Computation,* vol. 7, no. 1, pp. 19-44.

Leyland, G. 2002, *Multi-Objective Optimization Applied to Industrial Energy Problems*, PhD Thesis, EPFL.

Lo, A. & MacKinlay, C. 1999, *A Non-Random Walk Down Wall Street,* Princeton University Press, Princeton, NJ.

Lütkepohl, H. 1994, *Introduction to Multiple Time Series Analysis,* Springer-Verlag.

Malkiel, B. 2003, *A Random Walk Down Wall Street,* W. W. Norton & Company, New York.

Markowitz, H. 2000, *Mean-Variance Analysis in Portfolio Choice and Capital Markets,* Fabozzi, New Hope, PA.

McKay, M.D., Beckman, R.J. & Conover, W.J. 1979, "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output From a Computer Code", *Technometrics,* vol. 21, no. 2.

Mezura-Montes, E. & Coello Coello, C.A. 2002, "A Numerical Comparison of some Multiobjective-Based Techniques to Handle Constraints in Genetic Algorithms", *Technical Report EVOCINV-03-2002.*

Michalewicz, Z. 1995, "Genetic algorithms, numerical optimization and constraints", *Proceedings of the Sixth International Conference on Genetic Algorithms,* ed. L.J. Eschelman, Morgan Kaufmann, San Mateo, CA, pp. 151-158.

Michalewicz, Z. & Janikow, C. 1991, "Handling constraints in genetic algorithms", *Proceedings of the Fourth International Conference on Genetic Algorithms*, eds. R.K. Belew & L.B. Booker, Morgan Kaufmann, pp. 151-157.

Morrison, R.W. 2004, *Designing Evolutionary Algorithms For Dynamic Environments,* Springer.

Narendra, K.S. & Annaswamy, A.M. 1989, *Stable Adaptive Systems,* 1st edn, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Obayashi, S. & Sasaki, D. 2004, "Multiobjective Aerodynamic Design and Visualization of Supersonic Wings by using Adaptive Range Multiobjective Genetic Algorithms" in *Applications of Multi-Objective Evolutionary Algorithms*, eds. Coello Coello, C. A. & G.B. Lamont, World Scientific, pp. 295-315.

Paredis, J. 1994, "Co-evolutionary Constraint Satisfaction", *Proceedings of the Third Conference on Parallel Problem Solving from Nature* Springer-Verlag, New York, pp. 46-55.

Pedersen, G. K. M. & Yang, Z. 2006, "Multi-Objective PID-Controller Tuning for a Magnetic Levitation System using NSGA-II", *GECCO 2006* Seattle, WA, pp. 1737-1744.

Pflug, G. 2000, "Some remarks on the Value-at-Risk and the Conditional Value-at-Risk" in *Probabilistic constrained optimization*, ed. S. Uryasev, Kluwer, Dordrecht, pp. 272-281.

Powell, D. & Skolnik, M.M. 1993, "Using genetic algorithms in engineering design optimization with non-linear constraints", *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA-93)*, ed. S. Forrest, Morgan Kaufmann Publishers, San Mateo, California, pp. 424-431.

Ramsey, C.L. & Grefenstette, J. 1993, "Case-based initialization of genetic algorithms", *Proc. Int. Conf. Genetic Algorithms*, ed. S. Forrest, , pp. 84–91.

Reeves, C. & Rowe, J. 2003, *Genetic Algorithms – Principles and Perspectives,* Kluwer Academic Publishers.

Ronnewinkel, C., Wilke, C.O. & Martinetz, T. 2001, "Genetic algorithms in time-dependent environments" in *Theoretical Aspects of Evolutionary Computing* Springer-Verlag, London, pp. 261-285.

Rudolph, G. 1998, "Evolutionary search for minimal elements in partially ordered finite sets" in *Evolutionary Programming VII*, eds. V.W. Porto, N. Saravanan, D. Waagen & A.E. Eiben, Springer, Berlin, pp. 345–353.

Runarsson, T.P. & Yao, X. 2000, "Stochastic Ranking for Constrained Evolutionary Optimization", *IEEE Trans. on Evolutionary Computation,* vol. 4, no. 3, pp. 284-294.

Ruppert, D. 2004, *Statistics and Finance: an introduction.* Springer, New York.

Schlottmann, F. & Seese, D. 2005, "Financial applications of multi-objective evolutionary algorithms: Recent developments and future research." in *Handbook on Applications of Multi-Objective Evolutionary Algorithms*, eds. Coello Coello, C. A. & G. Lamont, World Scientific, Singapore.

Schneider, T. & Neumaier, A. 2001, "ARFIT – A Matlab package for the estimation of parameters and eigenmodes of multivariate autoregressive models", *ACM Transactions on Mathematical Software,* vol. 27, no. 1, pp. 58-65.

Senin, N., Wallace, D.R. & Borland, N. 2003, "Distributed Object-Based Modeling in a Design Simulation Marketplace", *Journal of Mechanical Design,* vol. 125, no. 2, pp. 2-13.

Shaffer, J.D. 1985, "Multiple Objective Optimization with Vector Evaluated Genetic Algorithms", *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, ed. L. Erlbaum, , pp. 93-100.

Simoes, A. & Costa, E. 2002, "Using Genetic Algorithms to Deal with Dynamic Environments: A Comparative Study of Several Approaches Based on Promoting Diversity", *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'02)*, ed. W.B. Langdon, Morgan Kaufmann, New York.

Spears, W. 1995, "Adapting Crossover in Evolutionary Algorithms", *Proceedings of the 4th Annual Conference on Evolutionary Programming*, eds. J.R. McDonnell, R.G. Reynolds & D. Fogel, MIT Press, Cambridge, MA.

Stambaugh, F. 1996, "Risk and Value-at-Risk", *European Management Journal,* vol. 14, no. 6, pp. 612-621.

Surry, P. & Radcliffe, N. 1997, "The COMOGA method: Constrained Optimization by Multiobjective Genetic Algorithms", *Control and Cybernetics,* vol. 26, no. 3, pp. 391-412.

Suzuki, J. 1995, "A Markov Chain Analysis on Simple Genetic Algorithms", *IEEE Transactions on Systems, Man and Cybernetics,* vol. 25, no. 4, pp. 655-659.

Tanaka, M. 1995, "GA-based decision support system for multicriteria optimization", *Proceedings of the International Conference on Systems, Man and Cybernetics - 2*, pp. 1556-1561.

Toffolo, A. & Benini, E. 2003, "Genetic Diversity as an Objective in Multi-Objective Evolutionary Algorithms", *Evolutionary Computation,* vol. 11, no. 2, pp. 151-167.

Tong, S. & Powell, D. 2003, "Genetic Algorithms: A Fundamental Component of an Optimization Toolkit for Improved Engineering Designs", *Genetic and Evolutionary Computation (GECCO) 2003* Springer, Berlin, pp. 2347-2359.

Vavak, F. & Fogarty, T.C. 1996, "Comparison of Steady State and Generational Genetic Algorithms for Use in Nonstationary Environments.", *Proceedings of the Society for the Study of Artificial Intelligence and Simulation of Behavior; workshop on Evolutionary Computation '96,* University of Sussex, pp. 301-307.

Vavak, F., Fogarty, T.C. & Jukes, K. 1997, "Learning the local search range for genetic optimization in nonstationary environments", *Proc. 4th IEEE Conf. Evolutionary Computation,* IEEE Press, Piscataway, NJ, pp. 355-360.

Vavak, F., Fogarty, T.C. & Jukes, K. 1996, "A genetic algorithm with variable range of local search for tracking changing environments", *Proc. 4th Conf. Parallel Problem Solving from Nature*, eds. H.-. Voigt, W. Ebeling, I. Rechenberg & H.-P. Schwefel, Springer, Berlin, pp. 376-385.

Weicker, K. & Weicker, N. 1999, "On evolution strategy optimization in dynamic environments", *Proc. Congr. Evol. Comput.*, pp. 2039–2046.

Woyak, S. & Malone, B. 1999, "ModelCenter and the Analysis Server: Tools for Building Distributed Applications on the Internet", *SUMMER COMPUTER SIMULATION CONFERENCE* Phoenix.

Wronski, J. 2005, *A design tool architecture for the rapid evaluation of product design tradeoffs in an Internet-based system modeling environment*, Master of Science in Mechanical Engineering, MIT.

Xiao, J., Michalewicz, Z., Zhang, L. & Trojanowski, K. 1997, "Adaptive Evolutionary Planner/Navigator for Mobile Robots", *IEEE Transactions on Evolutionary Computation,* vol. 1, no. 1, pp. 18-28.

Yamasaki, K. 2001, "Dynamic Pareto Optimum GA against the changing environments", *Proc. Genetic Evolutionary Computation Conf. Workshop Program*, pp. 47-50.

Zitzler, E., Laumanns, M. & Thiele, L. 2001, *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*, Swiss Federal Institute of Technology (ETH), Zurich.

Zitzler, E. & Thiele, L. 1999, "Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach", *IEEE Transactions on Evolutionary Computation,* vol. 3, no. 4, pp. 257-271.

Appendix

Constrained benchmark problems

Problems *g02* through *g13* can be found in (Koziel, Michalewicz 1999, Michalewicz 1995).

g02

maximize $$f(\mathbf{x}) = \left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1} n \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^{\,2}}} \right|$$

subject to $$g_1(\mathbf{x}) = 0.75 - \prod_{i=1} n x_i \le 0$$

$$g_2(\mathbf{x}) = \sum_{i=1}^{n} x_i - 7.5n \le 0$$

where n = 20 and $0 \le x_i \le 10\,(i = 1,...,n)$. The global maximum is unknown; the best reported solution in literature is in (Runarsson, Yao 2000), *f(***x***)* = 0.803619, with the constraint g_1 close to being active (g_1= - 10^{-8}).

g05

minimize $$f(\mathbf{x}) = 3x_1 + 0.000001x_1^3 + 2x_2 + (0.000002/3)x_2^3$$

subject to $$g_1(\mathbf{x}) = -x_4 + x_3 - 0.55 \le 0$$

$$g_2(\mathbf{x}) = -x_3 + x_4 - 0.55 \le 0$$

$$h_3(\mathbf{x}) = 1000\sin(-x_3 - 0.25) + 1000\sin(-x_4 - 0.25) + 894.8 - x_1 = 0$$

$$h_4(\mathbf{x}) = 1000\sin(x_3 - 0.25) + 1000\sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0$$

$$h_5(\mathbf{x}) = 1000\sin(x_4 - 0.25) + 1000\sin(x_4 - x_3 - 0.25) + 1294.8 = 0$$

where $0 \le x_1 \le 1200,\ 0 \le x_2 \le 1200,\ \text{-}0.55 \le \mathrm{x}_3 \le 0.55,$ and $\text{-}0.55 \le \mathrm{x}_4 \le 0.55$. The best known solution is $\mathbf{x}^* = (679.9453,\ 1026.067,\ 0.1188764,\ \text{-}0.3962336)$ where $f^* = 5126.4981$.

g06

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) = (x_1 - 10)^3 + (x_2 - 20)^3 \\ \text{subject to} & g_1(\mathbf{x}) = -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \le 0 \\ & g_2(\mathbf{x}) = (x_1 - 6)^2 + (x_2 - 5)^2 - 82.81 \le 0 \end{array}$$

where $13 \le x_1 \le 100$ and $0 \le x_2 \le 100$. The optimum solution is $\mathbf{x}^* = (14.095, 0.84296)$ where $f^* = \text{-}6961.81388$.

g10

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) = x_1 + x_2 + x_3 \\ \text{subject to} & g_1(\mathbf{x}) = -1 + 0.0025(x_4 + x_6) \le 0 \\ & g_2(\mathbf{x}) = -1 + 0.0025(x_5 + x_7 - x_4) \le 0 \\ & g_3(\mathbf{x}) = -1 + 0.01(x_{87} - x_5) \le 0 \\ & g_4(\mathbf{x}) = -x_1x_6 + 833.33252x_4 + 100x_1 - 83333.333 \le 0 \\ & g_5(\mathbf{x}) = -x_2x_7 + 1250x_5 + x_2x_4 - 1250x_4 \le 0 \\ & g_6(\mathbf{x}) = -x_3x_8 + 12500000 + x_3x_5 - 2500x_5 \le 0 \end{array}$$

where $100 \le x_1 \le 10000,\ 1000 \le x_i \le 10000\ (i = 2,3)$, and $10 \le x_i \le 1000\ (i = 4,...,8)$. The optimum solution is $\mathbf{x}^* = (579.3167,\ 1359.943,\ 5110.071,\ 182.0174,\ 295.5985,\ 217.9799,\ 286.4162,\ 395.5979)$ where $f^* = 7049.3307$.

g11

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) = x_1^2 + (x_2 - 1)^2 \\ \text{subject to} & h(\mathbf{x}) = x_2 - x_1^2 = 0 \end{array}$$

where $\text{-}1 \le x_1 \le 1$ and $\text{-}1 \le x_2 \le 1$. The optimum solution is $\mathbf{x}^* = \left(\pm 1/\sqrt{2}, 1/2\right)$ where $f^* = 0.75$.

g13

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) = e^{x_1x_2x_3x_4x_5} \\ \text{subject to} & h_1(\mathbf{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0 \\ & h_2(\mathbf{x}) = x_2x_3 - 5x_4x_5 = 0 \\ & h_3(\mathbf{x}) = x_1^3 + x_2^3 + 1 = 0 \end{array}$$

where $-2.3 \le x_i \le 2.3$ $(i = 1,2)$ and $-3.2 \le x_i \le 3.2$ $(i = 3,4,5)$. The optimum solution is $\mathbf{x}^* =$ (-1.717143, 1.595709, 1.827247, -0.7636413, -0.763645) where $f^* = 0.0539498$.

TNK

This problem can be found in (Tanaka 1995).

Minimize $\mathbf{f}(\mathbf{x}) = \left(f_1(\mathbf{x}), f_2(\mathbf{x})\right)$

where $f_1(\mathbf{x}) = x_1$

$f_2(\mathbf{x}) = x_2$

subject to $g_1(\mathbf{x}) = x_1^2 + x_2^2 - 1 - 0.1\cos(16\arctan\dfrac{x_1}{x_2}) \ge 0$

$g_2(\mathbf{x}) = (x_1 - 0.5)^2 + (x_2 - 0.5)^2 \le 0.5$

where $0 \le x_1 \le \pi$ and $0 \le x_2 \le \pi$. The feasible region boundary (optimum solution) as a Pareto front is plotted on the results graph as a continuous line.

Time-changing benchmark problems

Moving Peaks

This problem can be found in (Branke 2006). The moving peaks test function with *n* dimensions and *m* peaks is:

minimize $F(\mathbf{x},t) = \max\left\{B(\mathbf{x},t), \max\limits_{k=1,\dots,m}\left\{P\left(\mathbf{x}, h_k(t), w_k(t), \mathbf{p}_k(t)\right)\right\}\right\}$

where $0 \le x_j \le 100,\ j = 1,\dots,n.$

B is a time-invariant basis landscape and *P* is the function defining a peak shape with each of the *m* peaks having its own height *h*, width *w* and location **p**.

Every Δe evaluations a change in the landscape happens, and each peak changes shape and location in the following way:

$$\begin{aligned} h_k(t) &= h_k(t-1) + height\_severity \cdot \sigma \\ w_k(t) &= w_k(t-1) + width\_severity \cdot \sigma \\ \mathbf{p}_k(t) &= \mathbf{p}_k(t-1) + \boldsymbol{\upsilon}_k(t) \end{aligned}$$

where $\sigma \in N(0,1)$.

The shift vector $\mathbf{v}_k$ is a linear combination of a random vector $\mathbf{r}$ and the previous shift $\mathbf{v}_k(t-1)$, normalized to length s:

$$\mathbf{v}_k(t) = \frac{s}{\left|\mathbf{r} + \mathbf{v}_k(t-1)\right|}\left((1-\lambda)\mathbf{r} + \lambda \mathbf{v}_k(t-1)\right)$$

The coefficient λ (lambda) expresses the correlation in a peak's direction of motion between the previous and the current timestep. The random vector $\mathbf{r}$ is created by drawing random numbers for each dimension and normalizing its length to s.