TUNING EVOLUTIONARY SEARCH FOR CLOSED-LOOP OPTIMIZATION

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Engineering and Physical Sciences

2012

By
Richard Allmendinger
School of Computer Science

Contents

Word Count: 56459

List of Tables

List of Figures

List of Algorithms

Abstract

Closed-loop optimization deals with problems in which candidate solutions are evaluated by conducting experiments, e.g. physical or biochemical experiments. Although this form of optimization is becoming more popular across the sciences, it may be subject to rather unexplored resourcing issues, as any experiment may require resources in order to be conducted. In this thesis we are concerned with understanding how evolutionary search is affected by three particular resourcing issues — ephemeral resource constraints (ERCs), changes of variables, and lethal environments — and the development of search strategies to combat these issues.

The thesis makes three broad contributions. First, we motivate and formally define the resourcing issues considered. Here, concrete examples in a range of applications are given. Secondly, we theoretically and empirically investigate the effect of the resourcing issues considered on evolutionary search. This investigation reveals that resourcing issues affect optimization in general, and that clear patterns emerge relating specific properties of the different resourcing issues to performance effects. Thirdly, we develop and analyze various search strategies augmented on an evolutionary algorithm (EA) for coping with resourcing issues. To cope specifically with ERCs, we develop several static constraint-handling strategies, and investigate the application of reinforcement learning techniques to learn when to switch between these static strategies during an optimization process. We also develop several online resource-purchasing strategies to cope with ERCs that leave the arrangement of resources to the hands of the optimizer. For problems subject to changes of variables relating to the resources, we find that knowing which variables are changed provides an optimizer with valuable information, which we exploit using a novel dynamic strategy. Finally, for lethal environments, where visiting parts of the search space can cause the permanent loss of resources, we observe that a standard EA's population may be reduced in size rapidly, complicating the search for innovative solutions. To cope with such scenarios, we consider some non-standard EA setups that are able to innovate genetically whilst simultaneously mitigating risks to the evolving population.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

First and foremost I would like to thank my PhD supervisor, Joshua Knowles, for his valuable moral support, constructive criticism, help and feedback throughout the progress of this work, and for being a good friend too.

I am also very grateful to Hans-Paul Schwefel, Alan Reynold, David Corne, and Warwick Dunn for answering many questions about their experience with resource constraints.

Thanks to Arjun Chandra and Julia Handl for reading portions of my thesis and making useful suggestions.

Thanks also to Mauricio, Michalis, Kevin, Adam, Ahmad, Peter, Ruofei, Nicolo, Manuela, Stefan, Freddy, and Richard for making the office more fun and enjoyable!

To Neil Lawrence for being a fair opponent in internal swimming and cycling competitions, interesting conversations about sports, and the BBQs.

Warm thanks to all the people I have met and lived with, and made my time in Manchester more enjoyable, particularly Eyad, Liz, Claudia, Hagen, Abdul, Chris, Dan, Steve, Zoe, Lotti, Freddy, and Karla.

Finally, many thanks to my family, without whom I would never have been able to do what I did.

Nomenclature

σ_t	Set of problem-specific and time-evolving parameters
E_t	Set of evaluable solutions at time step t
k	Length of activation period
$t_{\text{ctf}}^{\text{start}}$	Time step at which the constraint time frame of an ERC starts
$t_{\text{ctf}}^{\text{end}}$	Time step at which the constraint time frame of an ERC is finished
T	Total optimization time
H	Constraint schema
$l(H)$	Length of a constraint schema
$o(H)$	Order of a constraint schema
V	Duration of an epoch
$commRelaxERC(t_{\text{ctf}}^{\text{start}}, t_{\text{ctf}}^{\text{end}}, V, H)$	Representation of a commitment relaxation ERC
P	Period length
$perERC(t_{\text{ctf}}^{\text{start}}, t_{\text{ctf}}^{\text{end}}, k, P, H)$	Representation of a periodic ERC
p	Activation probability
$randERC(t_{\text{ctf}}^{\text{start}}, t_{\text{ctf}}^{\text{end}}, k, o(H), p)$	Representation of a random ERC
$H_{\#}$	High-level constraint schema
$\#SC$	Number of storage cells
TL	Time lag
RN	Reuse number
SL	Shelf life
c_{order}	Cost per submitted composite order
$c_{\text{time\_step}}$	Cost per time step
C	Budget
$commCompERC(H_{\#}, \#SC, TL, RN, SL)$	Representation of a commitment composite ERC

Chapter 1

Introduction

In the late 60s, Hans-Paul Schwefel reported an ingenious set of experiments designed to optimize the shape of a flashing nozzle (Schwefel, 1968; Klockgether and Schwefel, 1970). Figure 1.1 illustrates the setup employed. The aim was to set aside traditional design principles — and the equations of fluid dynamics — and turn the design process instead over to a series of trial and error experiments akin to *natural evolution*. Schwefel was using an early form of evolutionary algorithm (EA) and evaluating designs, not through simulation, but by conducting real (physical) experiments. Although expensive (i.e. time consuming and/or resource expensive), this setup is effective because experiments replace the need for having available, or designing, sufficient mathematical models of the problem being solved.

This thesis will consider problems featuring experimental setups of very similar character to that of Schwefel's, which are nowadays commonly referred to as *closed-loop optimization problems*. The distinct difference between these problems and standard optimization problems is that while candidate solutions (genotypes) to a problem (e.g. set of parameter values specifying nozzle shapes) are *planned on a computer*, their phenotypes (e.g. an actual flashing nozzle) are *realized or prototyped ex-silico* (e.g. relying on a physical experiment of some sort). In general, the process of measuring the fitness (or quality) of the phenotype involves conducting a physical experiment too, as it was also the case in Schwefel's setup. Over the years, many scientific and technological problems — including in areas like quantum control, analytical biochemistry, robotics, electronics design, medicine, food science, neuroscience, and other sciences (an overview of closed-loop applications will be given in Section 2.2) — have been tackled using an experimental

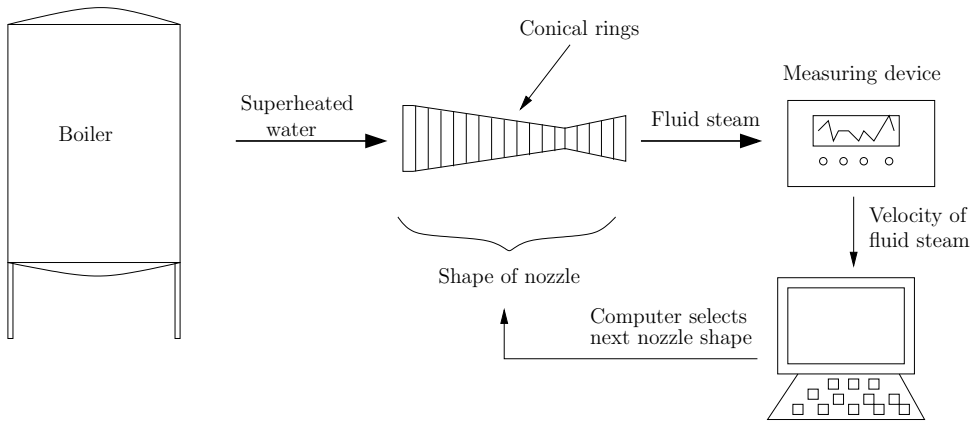

Figure 1.1: Schematic of the experimental setup as used by Schwefel (1968) in the shape optimization of a flashing nozzle. The nozzle consisted of a series of conical brass rings, each having its own diameter. The quality or fitness of the nozzle was measured by injecting superheated water into one side of the nozzle using a pressurized boiler, and measuring the velocity of the fluid steam at the other side of the nozzle. Based on this quality measure, a computer then selected the next nozzle shape for testing.

procedure similar to that of Schwefel's.

However, the reliance on physical experiments comes along with a variety of challenges to be faced by an experimentalist/optimizer. Schwefel's application features several challenges that are also common to other problems in this domain, such as noisy or imprecise measurements, uncontrolled environmental factors, and expensive fitness evaluations; an overview of common challenges in closed-loop optimization will be provided in Section 2.3. To cope with these challenges, Schwefel and other pioneering researchers in closed-loop optimization (Box, 1957; Schwefel, 1968; Klockgether and Schwefel, 1970; Rechenberg, 1973; Schwefel, 1975; Rechenberg, 2000) have opted for methods based on simulated evolution, such as EAs.[1] EAs have several advantages over other optimizers when it comes to closed-loop optimization (see Section 2.6 for examples) but one important practical reason for their usage is that they employ a set of solutions rather than a single one. This property is convenient when experimental equipment is capable of doing many experiments in parallel.

There seems to be an increasing interest in applying EAs to closed-loop optimization (Calzolari et al., 2008; Wong et al., 2008; Chen et al., 2009b; Knowles, 2009; Shir et al., 2009; Caschera et al., 2010), due perhaps to the increasing popularity of EAs as optimization methods across the sciences, the greater availability

[1]When an EA is used, closed-loop optimization is sometimes referred to as *evolutionary experimentation* or *experimental evolution*.

of automation of experiments in areas like physics, chemistry and the biosciences, and a push towards more interdisciplinary working. The tutorials by Shir and Bäck (2009) and by Bäck et al. (2010), and recent keynote talks at international conferences such as GECCO (Bedau, 2010) and PPSN (Michalewicz, 2010) are further evidence of a burgeoning interest.

Whilst there is growing evidence that closed-loop optimization works, there are various unexplored resourcing issues an experimentalist may face during an iterative experimental loop. The aim of this thesis is to understand how evolutionary search is affected by three particular *resourcing issues* in a closed-loop optimization scenario — ephemeral resource constraints (ERCs), changes of variables, and lethal environments — and the development of search strategies to combat these issues. The resourcing issue around ERCs was encountered by Knowles in several closed-loop optimization settings (see O'Hagan et al. (2005, 2007); Knowles (2009); Jarvis et al. (2010)), while changing variables were encountered in our own collaborative work (Small et al., 2011).[2] The third issue, lethal environments, is a challenge that may occur when equipment is potentially destructible as in software testing of autonomous robots or nanotechnology testing (we have not encountered this particular issue ourselves but believe that similar scenarios will arise in future).

The primary resourcing issue we are focusing on is related to the temporary non-availability of resources required in the evaluation process of solutions. This situation may cause solutions that are perfectly *feasible* candidate solutions to the problem to be temporarily *non-realizable* and thus not available for fitness assessment; we will refer to such solutions as (temporarily) *non-evaluable*. We refer to the constraints specifying which solutions are not evaluable at a given time as *ephemeral resource constraints* (ERCs), and any optimization problem that involves ERCs as an *ephemeral resource-constrained optimization problem* (ERCOP).

Although the resourcing issue around ERCs has not been raised much in the literature to date, at least in the terms similar to those we use in this thesis, precedents do exist. We have spoken to several people who do, or have done closed-loop work, and they have recounted how they dealt with the issue. For example, from discussions with Schwefel (April 2009 - January 2010), we know that he faced

[2]Note that ERCs are not discussed specifically in the work of O'Hagan et al. (2005, 2007); Jarvis et al. (2010), but they existed and were circumvented using waiting or other resource-wasteful strategies.

ERCs in his flashing nozzle work. In fact, not knowing in advance which shape would be realized, Schwefel sometimes ran out of brass rings of certain diameters. In that case, additional brass rings were ordered, produced and delivered, causing the optimization process to wait until these resources arrived; that is, here ERCs prevented the genotype-phenotype mapping. In modern closed-loop problems, such as combinatorial drug discovery and optimization of analytical instrument configurations (Weuster-Botz and Wandrey, 1995; Davies et al., 2000; King et al., 2004; O'Hagan et al., 2007; Knowles, 2009), there are several aspects that may further complicate the process of ensuring resource availability. For instance, the storage space for resources may be limited, there may be time lags between ordering resources and receiving them, and resources may be associated with shelf-lives or may be viable for usage only a limited number of times.

In another problem encountered by Schwefel, the fitness of a single solution was measured by running a time-consuming simulation on a computer (while there were no resources required in the genotype-phenotype mapping). During some simulations, the process ended prematurely (i.e. an execution error or exception occurred) and no fitness was returned. From a discussion with Reynolds and Corne (September 2010) we know that they faced a similar problem recently; execution errors were also encountered by other researchers including Booker et al. (1999); Büche et al. (2002).

Motivated by a closed-loop problem involving the identification of effective drug combinations drawn from a non-static drug library, the next resourcing issue considered in this thesis is related to problems featuring dynamically changing variables. The effect of changing variables is that we introduce a change in the fitness landscape we are optimizing over, which may cause the global optimum to shift; notice the difference to ERCOPs, which feature a static fitness landscape but dynamic constraints. Problems that are subject to changing variables can be seen as traditional dynamic optimization problems (Branke, 2001). However, in the scenario considered in this thesis an optimizer knows and may even be allowed to control when and which variables (drugs) are replaced. Compared to standard dynamic optimization problems, this property yields two important advantages: (i) changes in the fitness landscape do not need to be detected explicitly before a potentially shifting optimum can be tracked, and (ii) tracking of optima is simplified because there is a correlation between the changed variables and the parts of the landscape that undergo a change. While these two aspects provide

clearly some useful extra knowledge, the challenge remaining is how to exploit this knowledge efficiently to accelerate tracking of optima.

The final resourcing issue we are considering is related to optimization in lethal environments. The general setup is that we wish to evolve (i.e. improve) a finite population of entities, but when a lethal solution (i.e. one that damages equipment) is evaluated, it is immediately removed from the population and the population size is reduced by one. This models certain closed-loop evolution scenarios that may be encountered when the hardware on which individuals are tested are reconfigurable, destructible and non-replaceable. A motivational example of an optimization problem subject to lethal environments will be given in Section 6.2.1. In contrast to the other two resourcing issues (ERCs and changes of variables) lethal environments feature a static fitness landscape and feasible region; the dynamic component is associated with the fact that the population size may reduce over the course of the optimization.

1.1 Objectives and contributions

Our goal is to gain a better understanding of why and how the three resourcing issues mentioned above affect the application of evolutionary search for closed-loop optimization, and consequently devise search strategies for combating these issues.

We claim the thesis makes the following contributions:

- A general problem definition of ERCOPs, including the development of various types of ERCs. As our primary focus is on the investigation of the effects of ERCs; a large body of the thesis is devoted to advancing this subject (Chapter 4).

- An initial theoretical analysis of the effect of ERCs on simple EAs using Markov chains. The major aim of this analysis is to understand whether and how ERCs should affect configuration choices of an EA; the configuration choices considered relate to different selection and reproduction schemes commonly used within EAs (Chapter 4).

- Various static constraint-handling strategies, including repairing, waiting, and penalty strategies, for coping with ERCs. The empirical study of these strategies provides evidence that knowing about the type of ERC may be

sufficient to select an effective constraint-handling strategy for efficiently coping with it *a priori*, even when knowledge of the fitness landscape is limited (Chapter 5).

- Empirical investigation of two learning-based approaches for coping with ERCs. Both approaches aim at learning when to switch between various static constraint-handling strategies during the optimization process. However, while one approach learns this task *offline* using a reinforcement learning method, the other learns it *online* using a multi-armed bandit algorithm (Chapter 5).

- Various online resource-purchasing strategies to cope with ERCs that leave the arrangement of resources to the hands of the optimizer. We consider a specific scenario motivated by a real-world ERC (Chapter 5).

- A novel technique, which we call *fair mutation*, for dealing with problems featuring changing variables. To track potentially shifting optima upon a variable change, fair mutation exploits the correlation existing between the changed variables and the parts of the landscape undergoing a change (Chapter 6).

- Guidelines for tuning simple EA configuration parameters (degree-of-elitism, population size, selection pressure, mutation mode and strength, and crossover rate) for coping with lethal environments. The guidelines are validated on different fitness landscape topologies, helping us understand the underlying challenging features of landscapes in lethal environments (Chapter 6).

The thesis contributions in a wider scope are:

- A review of the field of closed-loop optimization including general concepts, applications, challenges, and optimization algorithms employed to address these challenges (Chapter 2).

- A taxonomy for describing ERCOPs and the different ERCs types we introduce (Chapter 4).

- A variety of detailed suggestions on directions of future research that we believe are necessary and fruitful in combating resourcing issues in closed-loop optimization (Chapter 7).

1.2 Outline of the thesis

In Chapter 2, we review literature specifically relating to closed-loop optimization. After defining general concepts of closed-loop optimization, we review a variety of closed-loop applications we found in the literature across the sciences and engineering. Following this review we summarize the main challenges common to closed-loop applications, and finally give an overview of methods used to tackle closed-loop problems. In particular, we introduce the design of experiments discipline, response surface methods, and methods based on simulated evolution and EAs. This thesis favours the application of EAs to closed-loop problems. The reasons for this choice will be described in detail in Section 2.6.

In Chapter 3, we review the application of EAs to problem classes that are related to closed-loop problems with resourcing issues, including noisy, constrained, dynamic, and online optimization problems. Prior to this review we give a more detailed description of the basic principles and concepts of evolutionary search, and outline techniques employed for: (i) theoretically analyzing and explaining the working of EAs, and (ii) tuning the performance of EAs. The final part of this chapter is devoted to the field of reinforcement learning (RL), which offers a variety of tools for tuning EAs. We introduce the fundamental ideas of RL and review several studies that looked at extending EAs with RL concepts to improve performance. This concludes the review of literature; the other chapters present original research.

In Chapter 4, we give a general problem definition of an ERCOP and define several types of ERCs derived from the experimental work of Knowles (2009). The second part of this chapter presents an initial theoretical analysis, using Markov chains, of the effect of one of the ERC types on common selection and reproduction schemes used within an EA. The theoretical analysis concludes that an order relation-based selection operator, such as tournament selection, is more robust to simple ERCs than a fitness proportionate-based selection operator. A perhaps more surprising result is that, while an EA with a non-elitist generational reproduction scheme converges more quickly to some optimal population state than with a non-elitist steady state scheme during unconstrained optimization periods, the opposite is the case during activation periods of an ERC. This result implies that ERCs should be accounted for when tuning EAs for ERCOPs. An empirical study of ERCOPs follows in the subsequent chapter.

In Chapter 5, we develop and empirically analyze various constraint-handling

strategies for coping with the ERCs introduced in Chapter 4. First we develop several general static constraint-handling strategies including repairing, penalizing and waiting strategies. Some of these strategies are inspired by existing strategies used for coping with standard (static) constraints, and some others are realizations of the tested approaches of Schwefel, Reynolds and Corne, described in the preface of this chapter. The empirical analysis reveals that different strategies should be favoured depending on the properties of an ERC, and gives evidence that it is possible to select a suitable strategy for an ERCOP offline, if the ERCs are known in advance. Inspired by this observation, we investigate whether an EA that learns offline (using a reinforcement learning approach) when to switch between the static strategies during the optimization process can do better than the static strategies themselves. We consider also a strategy that learns the same switching task online using a multi-armed bandit algorithm. Finally, we investigate strategies for coping with an ERC type that requires an optimizer to purchase resources online and pre-emptively so that they arrive in time to be used in the evaluation process of solutions. This ERC imposes three additional limitations related to the storage space, the shelf life and reuse number of the resources purchased. We show that this challenging ERC affects EA performance significantly but that certain online purchasing strategies that we propose and investigate here can be effective against it.

In Chapter 6, we consider the resourcing issues related to changing variables in the first part of the chapter, and lethal environments in the latter. A change in the variables causes *parts* of the fitness landscape to change in unpredictable ways; the fact that we know which variables are changed means that tracking of a potentially shifting optimum is slightly easier than in a standard dynamic optimization scenario. Based on this, we propose a novel strategy, which we call fair mutation, and compare it against standard techniques in dynamic optimization. For coping with lethal environments, we do not propose novel strategies per se but look at how this issue can be dealt with best by tuning some of the basic EA configuration parameters: degree-of elitism, population size, selection pressure, mutation mode and strength, and crossover rate. We also vary aspects of the fitness landscape we are optimizing to observe which landscape topologies pose a particular challenge when optimizing in a lethal environment.

In Chapter 7, we conclude the thesis, summarizing what we have learnt about

the effects of resourcing issues on evolutionary search when applied within closed-loop optimization, and how EAs can be extended to cope with these issues. In addition we summarize directions for further research.

1.2.1 Publications resulting from the thesis

Referred journal papers

B. G. Small, B. W. McColl, R. Allmendinger, J. Pahle, G. López-Castejón, N. J. Rothwell, J. Knowles, P. Mendes, D. Brough, and D. B. Kell (2011). Efficient discovery of anti-inflammatory small molecule combinations using evolutionary computing. *Nature Chemical Biology*, 7:902–908, 2011.

Referred conference papers

R. Allmendinger and J. Knowles (2011). Evolutionary search in lethal environments. In *Proceedings of the Evolutionary Computation Theory and Applications Conference*, to appear.

R. Allmendinger and J. Knowles (2011). Policy learning in resource-constrained optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM Press, pages 1971-1978.

R. Allmendinger and J. Knowles (2010). Evolutionary optimization on problems subject to changes of variables. In *Proceedings of Parallel Problem Solving from Nature – PPSN XI*. Springer LNCS 6239, pages 151-160.

R. Allmendinger and J. Knowles (2010). On-line purchasing strategies for an evolutionary algorithm performing resource-constrained optimization. In *Proceedings of Parallel Problem Solving from Nature – PPSN XI*. Springer LNCS 6239, pages 161-170.

Under review

R. Allmendinger and J. Knowles (2011). Policies for dealing with ephemeral resource constraints in closed-loop optimization. *IEEE Transactions on Evolutionary Computation*, under review.

Chapter 2

Closed-Loop Optimization

In the previous chapter, we introduced the class of closed-loop optimization problems in which solutions are evaluated through physical experimentation rather than simulation. The objective of this chapter is to give a more comprehensive introduction to the field of closed-loop optimization and its historic development. We begin this introduction with a more detailed description of the general concepts of closed-loop optimization in Section 2.1. An overview of applications of closed-loop optimization from various fields of science and engineering is provided in Section 2.2, followed by a summary of common challenges faced by experimentalists and algorithmic designers in these applications (Section 2.3). Finally, we give a historical review of approaches used to cope with closed-loop optimization problems. These include approaches based on design of experiments (Section 2.4), the response surface fitting (Section 2.5), and simulated evolution (Section 2.6).

2.1 Concepts of closed-loop optimization

Optimization is concerned with finding an answer $\vec{x}$ from a set of alternatives X that is best with respect to some criterion f. A more technical definition of an optimization problem of generic form is

$$\begin{aligned} &\text{maximize } \; y = f(\vec{x}) \\ &\text{subject to } \; \vec{x} \in X, \end{aligned} \tag{2.1}$$

where $\vec{x} = (x_1, ..., x_l)$ is a *solution vector* (also referred to as solution or candidate solution) and its l components for which values must be found are called

decision variables. The variables may be integer, real, or a mixture. The *feasible search space* X is the set of solutions over which the search is performed. In a standard constrained optimization problem, the set X is defined by a set of equality and non-equality constraints. The static *objective function* (also known as *fitness function*) $f : X \mapsto Y$ represents a mapping from X into the *objective space* $Y \subset \mathbb{R}$.

In many standard optimization problems, the function f is available in algebraic form allowing an optimizer to evaluate solutions by simply computing f. However, for some problems the mapping between the decision variables and their effects on the fitness of a solution is cost prohibitive to define or simply unknown; in this case we also say that the function f is *black box*. For instance, when optimizing combinations of drugs, the drugs interact in non-linear and complex ways that are difficult to predict and express in terms of a model. The approach taken to deal with such problems is to evaluate solutions by a process of physical experimentation or alternatively expensive computer simulations (Schütz and Schwefel, 2000). This is the type of optimization problem we consider in this thesis, and they are commonly referred as *closed-loop* or *experimental problems*. The term closed-loop, first used by Box (1957), suggests here that the setup in such problems establishes an interactive loop between an optimizer and the experimental platform, potentially including a (human) experimentalist.

Figure 2.1 illustrates a closed-loop setup commonly used in experimental optimization; Schwefel employed a very similar setup in his flashing nozzle problem (see Figure 1.1). To make the description of this loop clearer we will distinguish between two concepts: the genotype or structure of a solution (e.g. nozzle shape), and the phenotype or actual solution that can be evaluated (e.g. an actual nozzle). The genotype of a solution $\vec{x}$ is planned on a computer using an optimization algorithm. Then, the process of evaluating this genotype involves two steps: (i) realizing or prototyping the phenotype of $\vec{x}$, and (ii) measuring the (inherently noisy) fitness $f(\vec{x})$ of the phenotype. Both steps may require experiments to be conducted (although a strict separation between the two steps is not always possible). Upon obtaining the fitness value of a solution, this value is fed back to the computer and considered in the planning of the next solution. This closed-loop is repeated until some termination criterion is met. The real-world nature of experimental problems means that the termination criterion may be as simple as reaching a pre-determined fixed number of experiments conducted, but

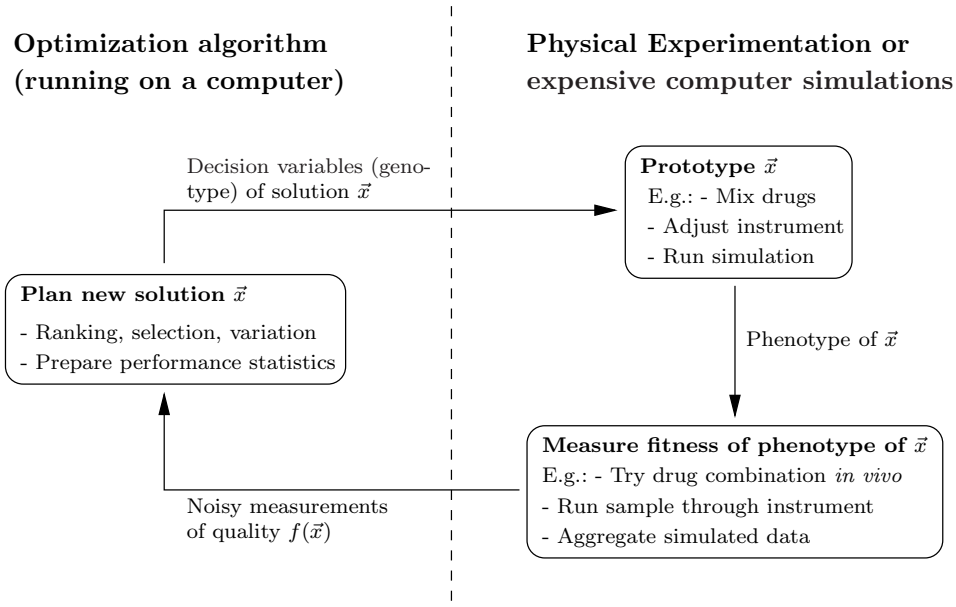

Figure 2.1: Schematic of closed-loop optimization. The genotype of a candidate solution $\vec{x}$ is generated on the computer but its phenotype is experimentally prototyped. The quality or fitness $f(\vec{x})$ of a solution may be obtained experimentally too and thus may be subject to measurement errors (noise).

time and budgetary limitations may enforce premature termination too.

It is common that an experimental loop involves human intervention. For example, biologists may be involved in the fitness measurement of drug cocktails, while optimizing the configuration of an instrument may rely on engineers for replacing and adjusting of instrument parts. Typically, human intervention is needed when dealing with large-scale optimization problems, such as optimization of industrial processes, or problems involving a complex and meticulous setup, such as applications in biochemistry. An alternative way for experimenting is to employ a fully automated closed-loop setup. Here, a host computer is connected to all the hardware possibly needed in the experimental loop, allowing it to autonomously plan, prepare and launch experiments, process their outcomes and subsequently use them to plan new experiments if needed.[1] Whilst this is a convenient way of experimenting, it may incur additional costs and effort for setting up the loop. The next section takes a closer look at experimental setups employed in applications from different areas of science and engineering.

[1]Other researchers, such as Hornung et al. (2001), refer to a fully automated closed-loop as a *self-learning closed-loop*.

Table 2.1: Domains of closed-loop applications.

Application domain	*Literature*
Shape design optimization	Rechenberg (1964, 1973, 2000); Schwefel (1968, 1975); Klockgether and Schwefel (1970); Olhofer et al. (2011)
Evolvable hardware	Thompson (1997, 1996a,b); Thompson et al. (1996); Thompson and Layzell (1999)
Evolving controllers for robots	Nolfi et al. (1994); Floreano and Mondada (1994); Matarić and Cliff (1996); Harvey et al. (1996); Husbands and Meyer (1998); Nolfi and Floreano (2004); Nelson et al. (2009)
Embodied evolution	Ficici et al. (1999); Watson et al. (1999, 2002); Pollack et al. (1999, 2001); Lipson and Pollack (2000); Zykov et al. (2005)
Experimental quantum control	Judson and Rabitz (1992); Baumert et al. (1997); Bardeen et al. (1997); Hornung et al. (2001); Zeidler et al. (2001); Pearson et al. (2001); Shir et al. (2007, 2009); Shir (2008); Roslund et al. (2009)
Adaptive optics	Sherman et al. (2002); Lubeigt et al. (2002); Marsh et al. (2003); Wright et al. (2005b,a)
Fermentation optimization	Rincon et al. (1993); Montserrat et al. (1993); Poorna and Kulkarni (1995); Weuster-Botz and Wandrey (1995); Weuster-Botz et al. (1995); Weuster-Botz (2000); Davies et al. (2000)
Functional genomics	Zytkow et al. (1990); King et al. (2004); Byrne (2007)
Optimization of analytical instrument configurations	Vaidyanathan et al. (2003); O'Hagan et al. (2005, 2007); Wallace et al. (2007); Jarvis et al. (2010)
Drug discovery	Singh et al. (1996); Calzolari et al. (2008); Wong et al. (2008); Caschera et al. (2010); Small et al. (2011)
Optimizing taste and aroma of food	Herdy (1997); Ferrier and Block (2001); Yang and Vickers (2004); Ferreira et al. (2007); Knowles (2009); Veeramachaneni et al. (2010)
Optimization of running industrial processes	Box (1957); Hunter and Kittrell (1966)

2.2 Applications of closed-loop optimization

This section gives a historical overview of applications in closed-loop optimization. The main applications are also summarized in Table 2.1.

In the 60s and 70s, closed-loop optimization problems were mainly concerned with **shape design problems** for fluid dynamics. Schwefel's work on shape optimization of a flashing nozzle, which we described in the previous chapter, is an example of this type. Another example is the work of Rechenberg (1973) on optimizing the shape of a pipe bend so as to obtain minimal flow losses. The bending was held by 6 manually adjustable shafts, and an optimization algorithm running on a computer provided the shaft setting (genotype) to be tested. The shafts were then adjusted accordingly (i.e. the phenotype was constructed) by

an experimentalist. In a more modern version of the experiment, this task was carried out by an industrial robot. Similar to Schwefel's application, the form of a bending was assessed by injecting pressured air into one side of the pipe using a kettle, and measuring the pressure of the air at the other side of the pipe. There are many further examples of shape design problems; for detailed descriptions please refer to (Rechenberg, 1964, 1973; Schwefel, 1975; Rechenberg, 2000).

Due to the increasing computational power and the constant development of innovative simulation software, problems related to fluid dynamics can nowadays often be optimized *in silico*. This technological progress affected the prominence of closed-loop optimization. However, in the last 20 years or so, closed-loop optimization regained its popularity thanks to applications in areas like evolvable hardware, evolution of controllers for robots, and applications in biochemistry.

The general objective in the field of **evolvable hardware** (Greenwood and Tyrrell, 2006) is to improve the setup of an electric circuit using an approach based on simulated evolution. The work of Thompson and his collaborators (Thompson, 1996a,b; Thompson et al., 1996; Thompson and Layzell, 1999) was seminal to this field. One of his studies (Thompson, 1996a) was related to the configuration of a FPGA consisting of 10×10 reconfigurable cells. The genotype of a particular configuration was encoded onto a bit string of length 1800, with 8 configurable bits allocated to each cell (the bits are related to the Boolean function a cell can perform). A computer, and on it implemented an optimizer, was connected to the chip allowing for automatic reading in and updating of the current configuration (see Figure 2.2). The experimental setup was such that no configuration could damage the chip nor had legality constraints to be checked. The aim was to design a configuration that is able to discriminate between two power levels of square waves presented at the input using a tone generator by accordingly changing the output voltage (which was visually inspected by an oscilloscope). The quality of a configuration was measured by calculating the mismatch between outputs observed over a random sequence of bursts of the two input waves. As an example where such a circuit could be used, Thompson (1996a) mentions the demodulation of frequency-modulated binary data received over a telephone line. Later, Thompson (1997) extended this work by considering difficult implementation constraints such as fault-tolerance. The interested reader is referred to Yao and Higuchi (1999) for a general review of challenges in the interesting and still very active field of evolvable hardware.

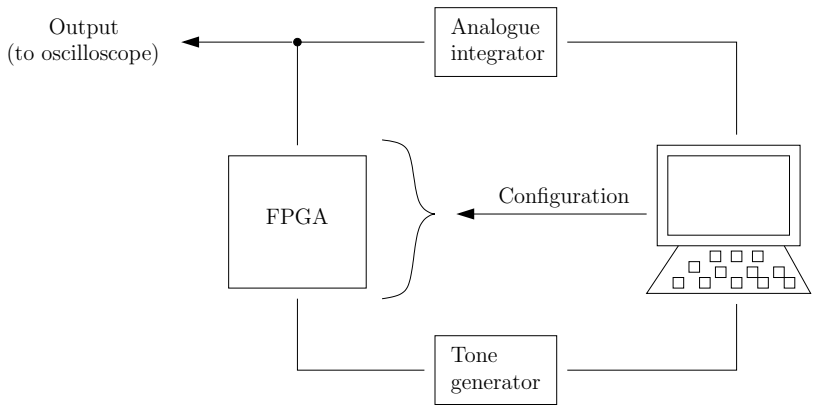

Figure 2.2: Experimental arrangement as used by Thompson (1996a) in the configuration optimization of an FPGA chip. A computer, and on it implemented an optimizer, was connected to the chip allowing for automatic reading in and updating of the current configuration. A tone generator drove the circuit's input, and an oscilloscope was employed to visually inspect the voltage of the circuit's output.

Compared to the pioneering design problems tackled by Schwefel and Rechenberg, we notice that the experimental loop employed in Thompson's FPGA work is fully automated and works without human intervention. A fully automated closed-loop (although sometimes supported by simulation experiments conducted beforehand) has also been adopted in the **evolution of controllers for robots** (Nolfi et al., 1994; Floreano and Mondada, 1994; Matarić and Cliff, 1996; Harvey et al., 1996; Husbands and Meyer, 1998; Nolfi and Floreano, 2004; Nelson et al., 2009). The general objective in this application is to improve the structure (e.g. weights and thresholds of activation functions) of a neural network (Bishop, 1995). The neural network is then used to transform the sensory inputs describing the environment into motor outputs of the robot. The quality of a controller is measured by assessing the behavior of the robot when applied to a particular task. For example, when the task is to reach the centre of a room from a fixed starting point, then a control system can be considered better the smaller the average distance between the robot and the centre is. As the fitness is expressed in terms of a robot's behaviour, the design of an appropriate fitness function becomes increasingly more challenging with the complexity of tasks to be performed by the robot (Harvey et al., 1996).

Some researchers believe that the future in evolving controllers for robots lies in a fully automated closed-loop procedure referred to as **embodied evolution** (Ficici et al., 1999; Watson et al., 2002): In this procedure, a population

of physical robots autonomously evolves in the task environment by reproducing with one another. A robot evaluates its own performance and broadcasts it along with information related to its controller setup to other robots in the population. These can then decide whether they want to use this information to update their own controller setup. Despite the difficulty of realizing embodied evolution in real-world (e.g. due to power and communication constraints), preliminary and promising results have been published e.g. by Pollack et al. (1999); Watson et al. (1999); Lipson and Pollack (2000); Pollack et al. (2001); Zykov et al. (2005).

A fully automated closed-loop that is rather inexpensive to execute is employed in **experimental quantum control**. The aim in this application is to tailor laser light fields by means of spectral amplitude and phase variables in order to control things like the position, orientation, velocity, and quantum states of the sample of interest (Zeidler et al., 2001; Pearson et al., 2001; Roslund et al., 2009). The ultimate objective can be, for example, establishing a laser light field that yields maximal molecular alignment (Shir et al., 2007, 2009). The convenient property of quantum control experiments is their short duration, which is of order of 1 second (Shir et al., 2009). This property allows for testing of different optimization algorithms and setups to rather low cost; Figure 2.3 shows a setup commonly employed in experimental quantum control. Additional information related to applications in experimental quantum control including physical background can be found in (Judson and Rabitz, 1992; Baumert et al., 1997; Bardeen et al., 1997; Hornung et al., 2001; Shir, 2008).

Inexpensive experiments within a fully automated closed-loop setup can also be conducted in **adaptive optics**. The general aim in this field is to design microscopy techniques that achieve optically sectioned images with high resolution of the biological sample under investigation. However, when imaging at depth within a biological sample, the resolution may suffer due to optical aberrations. Active optical elements, such as deformable mirrors, aim at counteracting this effect by altering the wavefront of the light source. Using a fully automated closed-loop approach, Wright et al. (2005a) showed that the resolution, or, more precisely, the brightness of the image of a sample can be optimized by appropriately configuring the shape of the deformable mirrors. In this particular application, the shape of the mirrors was controlled by 37 electrostatic actuators resulting in a continuous 37-dimensional search space. To test a mirror shape, a test sample was used instead of a real one in order to allow for accurate controlling of the

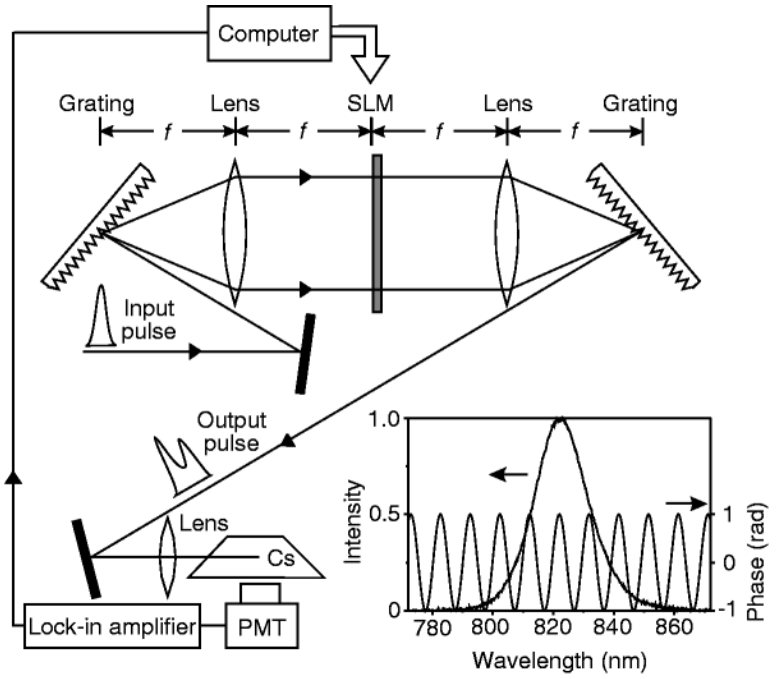

Figure 2.3: Experimental arrangement of an application in quantum control. A dynamic pulse shaper is used to generate modulated laser pulses. Experimental feedback signals are then collected from the liquid-phase emission spectroscopy, the Cs gas cell, and filtered optically and detected by the amplifier. The computer is used for reading in the processed signals and for updating the modulator. Figure taken from Meshulach and Silberberg (1998).

degree of aberration. Wright et al. report that an entire optimization run takes on average between 1 and 12 minutes, depending on the algorithm. Researchers in adaptive optics relying on a similar closed-loop setup include e.g. Sherman et al. (2002); Lubeigt et al. (2002); Marsh et al. (2003); Wright et al. (2005b).

When dealing with large-scale applications where experimentation involves complicated and/or extensive preparation and testing steps, it can be difficult and costly to set up a fully automated closed-loop. Applications in **fermentation optimization** as considered by Davies et al. (2000) belong to this class of closed-loop problems. The goal in the work of Davies et al. was to design silage additive combinations that yield herbage with high nutritional values and good storage properties. Before the herbage could be analyzed, it had to be grown for 4 weeks, cut, additively treated and packed at the start of the ensilage process, and after 2 days again unpacked. The evaluation of an entire generation, which was limited by logistical constraints to 50 additive treatments, took two weeks

(as two plots of grass were available); and, the number of generations was limited due to the constraints of herbage production over the normal growth season. To account for the apparent risk of measurement errors coming from biological, experimental and machine variability, each generation included replicates of control silage (the same herbage but without any additives) and replicate testing of silage additive combinations was applied too. An interesting aspect of the work is that costs were associated with additive concentrations in order to discourage the search for silages containing additives of high concentrations (which would be difficult to realize on a farm scale). Similar closed-loop approaches to fermentation optimization have been considered by Weuster-Botz and Wandrey (1995); Weuster-Botz et al. (1995); Weuster-Botz (2000); while these studies use optimization algorithms based on simulated evaluation, the work of Rincon et al. (1993); Montserrat et al. (1993); Poorna and Kulkarni (1995) employs response surface methods (see Section 2.5).

Closed-loop optimization is often the only way of experimenting in biochemistry applications, such as combinatorial drug discovery and optimization of analytical instrument configurations. A seminal closed-loop study in this field includes the robot scientist work of King et al. (2004). In this work a system has been developed (see Figure 2.4) to autonomously solve an **inference problem in the field of functional genomics** by a process of originating hypotheses to explain experimental data, devising and running experiments to test these hypotheses using a laboratory robot, interpreting the results to falsify hypotheses inconsistent with the data, and originating new hypotheses. A fully automated closed-loop for tackling inference problems has already been employed by Zytkow et al. (1990), though on a limited scale. In addition to achieving a high inference accuracy, the robot scientist accounts also for variable experimental costs. This gives rise to the problem of how to select an informative batch of experiments that can be tested with little expense; Byrne (2007) has opted for a multi-objective optimization approach to tackle this problem.

Modern analytical instruments such as mass spectrometers involve a large number of configuration parameters, and testing them systematically in order to optimize some objective is often impossible because of the large number of possible combinations. Consider the **instrumentation optimization** work of O'Hagan et al. (2005, 2007) aimed at detecting as many metabolites in a biological sample as possible using mass spectrometry. This application involved 15 integer

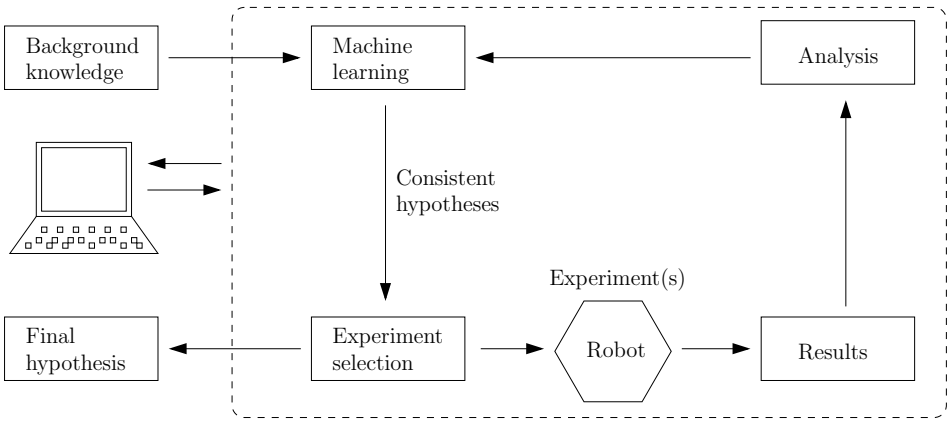

Figure 2.4: Experimental arrangement as used in the robot scientist study of King et al. (2004). An inference problem is solved using a computer-controlled closed-loop setup: hypotheses are originated to explain experimental data; experiments are devised and executed by a robot to investigate these hypotheses; results are interpreted to falsify hypotheses inconsistent with the data, and new hypotheses are originated.

variables or instrument parameters that made up a search space of 7.32×10^9 different instrument configurations. A typical closed-loop cycle in analytical experimentation involves to set up the instrument (phenotype) according to some configuration (genotype), and then to evaluate the quality of the configuration by running a biological sample through the instrument and measuring properties of the instrument's output signal, such as the number of peaks detected, signal-to-noise ratio of the peaks, and run time. O'Hagan et al. (2005, 2007) fully automated this cycle, while the studies presented by Vaidyanathan et al. (2003); Wallace et al. (2007); Jarvis et al. (2010) involved human intervention, with the human being involved in tasks like changing instrumentation settings.

Combinations of pharmaceuticals (also referred to as drugs, reagents, or compounds) interact in complex ways that typically cannot be predicted. Hence, when optimizing combinations of drugs and/or concentrations of these drugs according to some criterion, such as the expression of pro-inflammatory cytokines (Small et al., 2011), an experimental approach needs to be adopted. In a common **drug discovery application**, a human experimentalist arranges a library of promising or relevant drugs to be considered by an optimizer running on a computer. The library size may range from 10 drugs (Wong et al., 2008) to a few dozen drugs (Small et al., 2011), and sometimes be as large as 100 drugs (Calzolari et al., 2008). Drug mixtures (genotypes) devised by the optimizer are then mixed together (phenotyped) commonly using a laboratory robot, and their potency is

measured using analytical tests. However, this process may involve several complex and manually performed steps related to the storage and maintenance of individual drugs, setup of analytical tests, and the actual testing. A glance at the method section of the above cited studies including the work of Singh et al. (1996); Caschera et al. (2010) will confirm the sophistication of experimental setups in drug discovery applications.

Recent studies indicate that closed-loop optimization has also been used by the food industry for **improving taste and aroma of food**. Knowles (2009) reports on the optimization of a cocoa roasting process employed by a chocolate manufacturer. The precise objective in this application is to optimize the design of a roast curve so as to create cocoa of a specific desired flavour. A genotype defines the properties of a roast curve including the temperature with which cocoa nibs (the center of the beans) are roasted, and the placement and duration of water injections during the roasting. Upon physically realizing a roast curve, the quality of the resulting cocoa is subjectively assessed by a human taster. Obvious challenges in this application are the variability in the quality of the raw cocoa nibs, and the subjective fitness assessment, which greatly depends on the experience of the human taster. Unfortunately, the optimization of taste and aroma is often a business secret, limiting the number of studies publicly available. Nevertheless, we have found applications related to the closed-loop optimization of blends of coffee (Herdy, 1997), the blending of wine (Ferrier and Block, 2001), and the taste of cheddar (Yang and Vickers, 2004). More accessible is research related to the optimization of procedures for the isolation of taste in a food sample (Ferreira et al., 2007). Of interest is the recent study of Veeramachaneni et al. (2010), which mentions the application of closed-loop optimization in the design of food flavors. More precisely, a small set of mixtures of ingredients are subjectively evaluated by a panel of consumers using a closed-loop approach. A model (surrogate) of the consumer data is then generated and used to optimally determine a set of flavors liked by some target group of panellists.

Further and perhaps more isolated applications of closed-loop optimization can be found, for example, in the field of **combinatorial chemistry** (Schlögl, 1998; Gobin et al., 2007; Wolf et al., 2000), **ceramic ink-jet printing** (Evans et al., 2001), **combustion process optimization** (Büche et al., 2002; Paschereit et al., 2003; Hansen et al., 2009), **directed evolution** (Arnold, 1998; Knight et al., 2009; Platt et al., 2009), and **deep brain stimulation therapy** (Feng

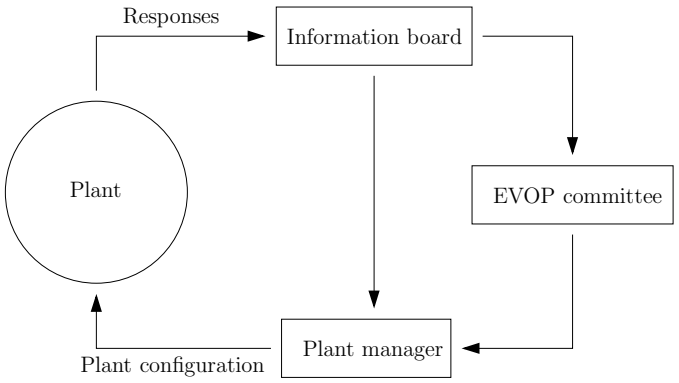

Figure 2.5: Schematic of the closed loop as employed with evolution operation (EVOP) (Box, 1957) to optimize a running manufacturing process. The EVOP committee contains specialists on EVOP that serve as advisors but the ultimate responsibility for running the scheme rests with the plant manager. She sets up the plant configuration to be tested. The resulting responses are then used to update the information board on which future decisions are made.

et al., 2007).[2] Detailed case studies of selected applications in instrument optimization, drug discovery, directed evolution, and cocoa roasting have also been presented by Knowles (2009).

The above closed-loop applications have in common that the search for an optimal solution is terminated after some fixed period of time (e.g. due to time and/or budgetary limitations) or once a solution of high quality has been found. Typically, the aim is then to manufacture the solution found (e.g. a drug combination, nozzle, software, electric circuit, etc.) many times and sell it to the public or other companies; perhaps improving it further at some future point in time when the quality of raw material and technological equipment has advanced.

However, closed-loop setups have also been adopted in the continuous improvement of a running industrial process, similar to the cocoa roasting process mentioned above but on a larger scale. The goal here is to optimize some cumulative performance of the process over an infinite or finite time horizon, rather than finding a single optimal (and ultimate) solution, and this needs to be done in a controlled and restricted fashion to reduce the risk of producing off-quality material. To achieve this goal, Box (1957) suggested an approach termed evolution operation (EVOP) (see Figure 2.5). The objective of EVOP is to continuously improve a running industrial process by applying small changes (mutations) to

[2]Although the work of Feng et al. (2007) describes a closed-loop optimization framework (which the authors hope to realize in future) for achieving deep brain stimulation therapy of Parkinsonian motor symptoms, for validation purposes a computational model is used.

it. Rather than taking random mutation steps, EVOP is guided by a committee including human experimentalists and specialists, such as statisticians and experts on EVOP. Since its introduction in 1957, EVOP has been applied in numerous sectors including chemical manufacturers, automotive industry, natural gas producers, food industry, and canning industry. Thanks to EVOP, savings were made, for example, due to reduced usage of raw material, reduced time cycles in batch operations, and equipment modifications. In addition to savings and the resulting profits, EVOP contributed also to the understanding of certain processes, and pointed engineers to the importance of specific control variables. An extensive review of EVOP applications and benefits of EVOP has been published by Hunter and Kittrell (1966).

Nowadays, the movement of continuously improving a process including products and services with the aim to reduce waste and time is referred to as lean manufacturing or lean management (Shah and Ward, 2003). Tools for realizing lean manufacturing include EVOP but also techniques such as just-in-time production (Sugimori et al., 1977), design of experiments (Montgomery, 1976) and Taguchi methods (Taguchi, 1989; Peace, 1993).

2.3 Challenges of closed-loop optimization

From the description of some of the applications listed above it is apparent that closed-loop problems feature challenges that may not be prevalent in standard problems tackled by computer simulation. We will briefly recall the main challenges here; for a more detailed description please refer to Knowles (2009).

Arguably the most obvious challenge of closed-loop optimization is that *evaluations of solutions are generally expensive*. This concerns both the time and costs needed to conduct experiments. In the presence of time and/or budgetary limitations this factor can restrict the number of evaluations available to several tens or hundreds only.

Experiments may be of different durations and associated with non-homogeneous experimental costs. This scenario has been encountered e.g. in the robot scientist work (King et al., 2004) and in fermentation optimization (Davies et al., 2000). In the presence of a limit budget, this challenge can cause an optimizer to account for both fitness gradients and differential costs of evaluations.

Designing a suitable fitness function may be difficult. This challenge exists

in both computational and physical experimentation, but the complexity of real world systems means often that fitness function design is more delicate and laborious in practice (Mondada and Floreano, 1995; Matarić and Cliff, 1996; Knowles, 2009). As an example, consider the design of a fitness function that evaluates the behavior of a robot in achieving a certain task. The difficulty of designing a suitable fitness function tends to increase with the complexity of the task to be fulfilled (Harvey et al., 1996).

Experiments are inherently noisy, and may be subject to *uncertainty and uncontrolled factors*. Noise is a challenge that is much studied, and several methods have been proposed to account for it. In Section 3.2 we will introduce some of these methods in the context of evolutionary optimization.

Experimental setup and environmental factors may dictate settings of the optimization algorithm. Recall that this situation was present, for instance, in the fermentation optimization work of Davies et al. (2000). A population-based optimizer was employed in this application. Its population size was limited by logistical constraints to 50 additive treatments, and the number of generations the optimizer could be employed for was also limited due to the constraints of herbage production over the normal growth season. In biochemistry applications, such as in drug discovery (Small et al., 2011), the use of standard laboratory tools like microwell plates may dictate the population size too.

Experimentation may be subject to encoding issues and constraints. It is common for closed-loop problems to have an encoding with components being both discrete (or binary) and real values. Constraints may restrict the values certain variables may take but also complicate the optimization process in other ways (as investigated in this thesis in Chapters 4 to 6). For example, in the evolution of robot controllers, the lifetime of the robot battery and the robot itself may slow down the optimization procedure due to recharging, maintenance and repair duties (Matarić and Cliff, 1996). Constraints may also be challenging to identify and define as reported by Knowles (2009).

2.4 Design of experiments

Traditionally, closed-loop problems have been dealt with using methods from the design of experiments (DoE) (also known as experimental design (ED)) (Montgomery, 1976; Mason et al., 1989; Box et al., 2005) discipline. These are statistical

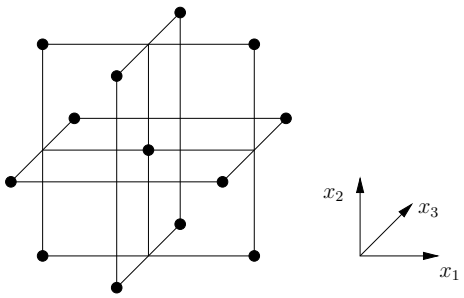

Figure 2.6: Plot showing factor combinations to be tested with a Box-Behnken experimental design for a system with $l = 3$ factors with each factor having three levels. The factor combinations are specifically chosen to allow the efficient estimation of the coefficients of a quadratic model as defined in Equation (2.2).

methods traditionally concerned with explaining a system or process in terms of a model using as few informative experiments as possible.[3] This model can then be used to study different properties of a system, such as the most important factors of the system or the performance of the system in the presence of noise. Figure 2.6 shows an example of a particular experimental design. For exhaustive reviews of experimental designs, including conditions when their application is recommended, and instructions to construct them, please refer to the books (ibid.).

The three basic principles employed by many DoE techniques are *replication, randomization, and blocking* (also known as *local control*) (Preece, 1990; Box et al., 2005). Replication (i.e. repeating of the entire experiment or a portion of it under two or more sets of conditions) is a way for estimating and accounting for noise, while randomization and blocking are used to deal with variabilities caused by unavoidable sources, or nuisance factors (e.g. ambient temperature or the time of the day the experiment was conducted). Randomization refers to the process by which factor combinations for testing are allocated to individual experiments in a random or unbiased approach. Blocking is the process by which the total set of trials is partitioned into subsets (blocks) that are as homogeneous as possible.

The DoE discipline has also produced methods for seeking optimal factor combinations: factorial, univariate, steepest-ascent, and random methods (Brooks, 1958, 1959). The univariate and steepest ascent method are sequential procedures, while the other two methods generate an entire set of trials for testing

[3]The terminology in the field of DoE differs slightly from the one in optimization. Most notably, decision variables are referred to as factors, their values as levels, fitness as response, and an experiment testing a particular factor combination (candidate solution) is called a trial.

beforehand. The random method tests factor combinations selected at random in the factor space. Compared to factorial and univariate methods, which test levels of all factors in some systematic approach, random methods can be applied to systems involving a larger number of factors. Compared to steepest-ascent methods, random methods are also less likely to find only local optimal factor combinations. In any of these methods, there are two options by which an optimal factor combination can be estimated. The first option is to simply select the factor combination with the highest response from the set of all trials made. The other option is to construct a response surface by fitting a polynomial surface to the responses of the trials made, and then estimating the optimal factor combination by finding the highest point of this surface. We will describe this methodology in more detail in the next section.

2.5 Response surface methods

Traditionally, a response surface is fitted using polynomial models (in the factors) for the responses y (Box and Wilson, 1951). As an example, consider the quadratic model

$$y = b_0 + \sum_{i=1}^{l} b_i x_i + \sum_{i=1}^{l-1} \sum_{j=i+1}^{l} b_{ij} x_i x_j + \sum_{i=1}^{l} b_{ii} x_i^2 \ + \epsilon \ . \tag{2.2}$$

This model is expressed in terms of main effects of factor levels $(x_1, ..., x_l)$, their interactions or joint effects with other factor levels $(x_1x_2, ..., x_{l-1}x_l)$, and their quadratic components $(x_1^2, ..., x_l^2)$. To account for noise and experimental errors, a random error ϵ (assumed to be uncorrelated and distributed with mean 0 and constant variance) may be considered in the model as well. The normal procedure is to fit a surface to the available data using the least square method; Figure 2.7 shows an example of a response surface. Following this, the validity of the fitted surface may be investigated using an analysis of variance (ANOVA), whereby the observed variability of the responses (measured by sums of squared deviations from mean response) is partitioned into independent, or orthogonal, components from identifiable sources. A comprehensive introduction to least squares, and the application of ANOVA in the context of response surfaces can be found in Chapter 10 of (Box et al., 2005).

Response surface methods (RSMs) employing the above described approach

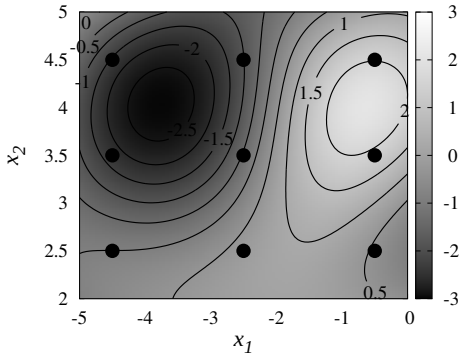

Figure 2.7: A response surface plot for a system with $l = 2$ factors. The surface is fitted to the nine data points representing different experiments.

generate a response surface that tends to be non-interpolating because they minimize only the sum of squared deviations from some pre-determined (polynomial) function. Such a surface may be misleading for seeking maxima as it may not sufficiently approximate the shape of the actual underlying model (Jones, 2001). This drawback encouraged many researchers to develop RSMs that approximate the actual model more accurately, ideally, also in fewer trials. Modern RSMs tend to generate a surface that interpolates the observed data points using a linear combination of 'basis functions' (one 'basis function' term is centred around one data point), such as thin-plate splines and kriging (Matheron, 1963; Sacks et al., 1989).

RSMs can be further classified into two-stage and one-stage (or sequential) DoE methods (Robbins, 1952b). Two-stage approaches fit a surface to observed data in stage one, estimating any required parameters (related to the surface). In the second stage, these parameters are assumed to be correct and the surface is used to select the next data point for evaluation. The drawback of this approach is that the search can be misguided if the estimated surface from stage one is incorrect. Sequential DoE methods aim at overcoming this issue in that they skip the first stage of fitting a surface to observed data. Instead, hypotheses about the location of the optimum (its variables $\vec{x}$ and fitness $f(\vec{x})$) are made and evaluated by determining properties of the best-fitting response surface that passes through the observed data *and* the point $(\vec{x}, f(\vec{x}))$ (Jones, 2001). However, this approach comes at a cost of high computational demands, particularly when kriging is used (because a matrix of estimated correlations between responses needs to be inverted). For a more comprehensive review of one- and two-stage

RSMs please refer to Jones (2001); Myers et al. (2009).

An alternative approach to RSMs are methods based on simulated evolution, which we will discuss in the next section.

2.6 Simulated evolution in closed-loop optimization

Proposals to consider ideas from nature, in particular, the concept of *natural selection* and *variation*, for closed-loop optimization were already made in the late 1950s. Brooks (1958) proposed a *creeping random method* that extends the DoE-based random method introduced elsewhere (see Section 2.4) with the concept of natural selection. The experimental design technique of Box (1957), EVOP (see Section 2.2), employed the concepts of natural selection and variation too, although the variation steps are proposed by some committee rather than being random. Remember, however, that the objective of EVOP is to optimize a cumulative score over an infinite or finite time horizon rather than to find a single optimal (and ultimate) solution.

The work of Bremermann (1962); Fraser (1957); Friedberg (1958); Friedberg et al. (1959) and others has further popularized approaches based on simulated evolution and initiated the field of *evolutionary computation*. In particular, this pioneering work influenced the development of the three related approaches, genetic algorithms (GAs) (Holland, 1975; De Jong, 1975), evolution strategies (ESs) (Rechenberg, 1973; Schwefel, 1975), and evolutionary programming (EP) (Fogel, 1962, 1964), nowadays commonly referred to as *evolutionary algorithms* (EAs). All three approaches are population-based search techniques that generate solutions based on a process of simulated evolution. For a more comprehensive historical overview of the development of EAs please refer to (Fogel, 1994; Bäck et al., 1997; Fogel, 1998).

Whilst certain DoE methods, such as the creeping random method and EVOP, and EAs may share similar working principles, there are differences in the objectives pursued by the two algorithm types: DoE is typically applied to low-dimensional search spaces with the aim of obtaining statistically robust results using as few experiments as possible. EAs, on the other hand, are approaches

commonly used for finding a single optimal solution, using typically many evaluation steps, to problems featuring high-dimensional search spaces e.g. combinatorial optimization problems such as the TSP. Closed-loop optimization problems often fall in the intersection of the two scenarios and thus require ideas from both areas. In particular, we believe that closed-loop evolution methods should draw on DoE in areas like noise-handling, replication, and blocking (e.g. the pipe bending setup of Rechenberg (2000) used blocking). To reduce the number of expensive evaluations, ideas from sequential DoE and RSMs can also be incorporated into evolutionary search, where they are known variously as surrogate models, meta-models or fitness approximation methods; this step has already been realized in recent years (see e.g. Jin et al. (2002); Ong et al. (2004); Jin (2005) for surveys on this subject). To our knowledge, however, the DoE field has not so far considered resourcing issues as a perturbing influence on conducting the most informative experiments, as we do here.

There are also several aspects that make EAs suitable candidates for closed-loop optimization: Evolutionary search relies solely on the fitness of previously evaluated solutions and not on information related to the problem instance being solved, making EAs suitable for problems featuring a black-box function, such as closed-loop problems. In fact, evolution does not even need to be given specific fitness values but searching for behavioral novelty may be sufficient to solve a problem (Lehman and Stanley, 2011). EAs can also be hybridized easily if prior knowledge about the problem at hand, gradient information, or other methods are available (Blum and Roli, 2003). The ability to employ a population of solutions allows EAs to conveniently adjust the population size to meet a given experimental setup, and, moreover, to cope with problems potentially featuring multiple objectives (Deb, 2001), complex (non-linear and dynamic) constraints (Michalewicz and Schoenauer, 1996; Coello, 2002; Nguyen and Yao, 2009a), and a dynamically changing fitness landscape (Branke, 2001).

2.7 Chapter summary

In this chapter we have reviewed the field of closed-loop optimization, which is characterized by the property that solutions are evaluated by conducting experiments (as opposed to evaluating a closed-form mathematical expression). We first introduced the general concepts of closed-loop optimization. Then, we reviewed a

variety of closed-loop applications across the sciences and engineering, and summarized the main challenges of closed-loop optimization. Finally, we provided an overview of techniques employed in closed-loop optimization including approaches based on design of experiments, response surfaces and simulated evolution. We favour the application of simulated evaluation approaches and, in particular, EAs, to closed-loop problems and focus on them in this thesis. The reasons for this choice have been stated in the previous section. The next chapter will introduce EAs in more detail, and review techniques employed within EAs for coping with many of the challenges common to closed-loop optimization, such as noise, constraints, and dynamically changing environments. These challenges are related to the resourcing issues considered in this thesis, and understanding how they are dealt with is important and beneficial for the development of our own strategies.

Chapter 3

Evolutionary Optimization

In the previous chapter we introduced the field of closed-loop optimization and motivated the choice for using evolutionary algorithms (EAs) as optimizers in this domain. The next section introduces some of the basic principles of evolutionary search and reviews various approaches for tuning and controlling parameters of an EA. The application domain of EAs includes several problem categories related to closed-loop problems and the particular resourcing issues (ephemeral resource constraints, changing variables, and lethal environments) considered in this thesis: the main categories are noisy, constrained, dynamic, and online optimization problems. Section 3.2 to 3.5 review how EAs have been tuned to tackle these problem categories (these sections will also highlight the relationship between the resourcing issues considered in this thesis and the different problem categories, but for a more detailed description of the resourcing issues themselves please refer to the subsequent chapters). Tuning of EAs has also been realized with techniques from machine learning, such as reinforcement learning techniques. Section 3.6 gives a brief introduction to the RL field, and reviews methods for extending EAs with RL to improve performance.

3.1 Evolutionary Algorithms

From Section 2.6 we know that evolutionary algorithms (EAs) (Fogel, 1962; Rechenberg, 1973; Schwefel, 1975; Holland, 1975; Goldberg, 1989) are population-based techniques that tackle optimization problems based on models of *biological evolution*. We can think of biological evolution as a process where fit individuals of a population are passed on from one generation to the next upon undergoing

Table 3.1: Nomenclature used in evolutionary algorithms

Term	*Meaning*
Individual	Solution
Population	Set of individuals
Fitness	In the context of EAs, the term fitness is used to refer an individual's quality, while in biology it refers to the reproductive success of an individual in passing its genetic information to the next generation
Objective function	Function that computes the fitness of an individual
Chromosome/Genome	Encoded individual; often represented as a string of parameters
Gene	Basic unit of an individual; in optimization, a gene can be seen as a decision variable
Allele	Value of a gene
Locus	Position of a gene in a string; this term is often used interchangeably with the term gene
Genotype	Genetic structure that represents an individual
Phenotype	Individual that can be evaluated
Crossover/Recombination	Process by which two or more individuals exchange some of their genetic information; alternatively, the term(s) refer to the operator that performs this process
Mutation	Process by which an individual's genetic information undergoes random changes; alternatively, the term refers to the operator that performs this process
Variation	Crossover and/or mutation
Parent	Individual used to generate new individuals (offspring)
Offspring/Child	Individual resulting after applying crossover and/or mutation to parents
(Parental) selection	Process by which parents of the current population are selected for crossover and/or mutation
Generation	One iteration through parental selection, variation and environmental selection, to arrive at an updated population
Environmental selection	Process by which individuals from the combined pool of offspring and/or current population are selected to form the new population

variation in form of mutation and/or crossover. Due to the relationship to biology, many biological terms have been adopted to describe EAs; these terms and their meanings are summarized in Table 3.1.

Let us describe the working scheme of a basic EA (see Algorithm 3.1) in more detail; modifications of this scheme will be introduced in the remainder of this section. This thesis considers EAs featuring a very similar structure to that shown in Algorithm 3.1. In the algorithm, μ denotes the size of the current population of solutions *Pop*, also referred to as parent population. Upon initializing *Pop* with,

Algorithm 3.1 A basic generational evolutionary algorithm

Require: μ (parent population size), λ (offspring population size), f (objective function)
1: $g = 0$ (generation counter), $Pop = \emptyset$ (parent population), $OffPop = \emptyset$ (offspring population)
2: initialize Pop
3: evaluate all solutions in Pop using f
4: **while** termination condition not satisfied **do**
5: **repeat**
6: select parents from Pop
7: generate offspring by applying variation operators to selected parents
8: evaluate offspring using f and add them to $OffPop$
9: **until** λ offspring are created
10: form new population Pop by selecting solutions from Pop and $OffPop$
11: g++
12: return best solution of Pop

for example, μ randomly generated individuals, an offspring population $OffPop$ of size λ is incrementally generated. This is done by first selecting parents for reproduction from Pop, and then applying variation operators to them in order to produce offspring solutions; commonly used variation operators are crossover and/or mutation. Before offspring individuals are added to $OffPop$, they are evaluated using the objective function f. Once the offspring population is complete, a new population Pop is formed by selecting fit solutions from the combined pool of solutions from Pop and $OffPop$. This step is referred to as *environmental selection*, and may involve selecting the fittest μ solutions from Pop and $OffPop$. This variant is referred to as $(\mu + \lambda)$ reproduction, and the special case where $\lambda = 1$ is also referred to as steady-state reproduction. The concept where the fittest individuals found so far are allowed to survive to the next generation is known as elitism (De Jong, 1975). Alternatively, if $\lambda \geq \mu$, a new population Pop can be formed by selecting the fittest λ offspring individuals from $OffPop$; this scheme is referred to as (μ, λ) reproduction, and the special case where $\lambda = \mu$ is also referred to as generational reproduction. The notations $(\mu + \lambda)$ and (μ, λ) were originally used within evolution strategies (ESs) (Rechenberg, 1973; Schwefel, 1975).

The variety of EAs is very large. In addition to different reproduction schemes, there are different schemes for performing parental selection, recombination, and mutation. We will give examples of common EA setups. Two commonly used selection operators are fitness proportional selection (also known as roulette-wheel selection) and tournament selection. The former operator selects a parent with a probability that is proportional to its fitness. The latter operator first chooses a set of TS solutions (TS is referred to as the tournament size) at random from the

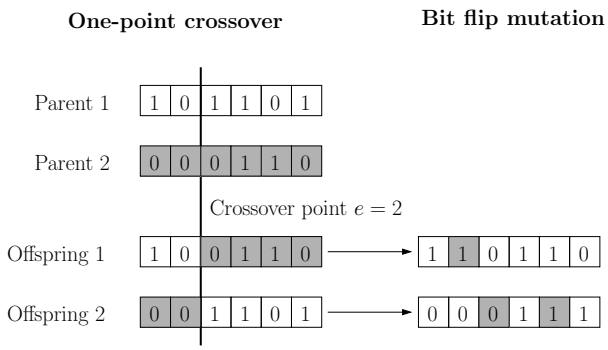

Figure 3.1: Visualization of one-point crossover (left) and bit flip mutation (right). One-point crossover selects a single crossover point e (here $e = 2$) at random, and then swaps all genes beyond that point in either parent's chromosome between the two parents. Bit flip mutation flips each gene in an offspring's chromosome independently with some probability p_m.

current population, compares these solutions, and then selects the best one as a parent. Note, in a standard EA, this selection step is carried out twice to obtain two parents. Typically, with probability p_c the two parents are then recombined by, for example, swapping each of their genes with a probability of 0.5; this form of crossover is known as uniform crossover (Syswerda, 1989). Alternatively, parents can be recombined using one-point crossover, in which case a random crossover point $e \in \{1, ..., l-1\}$ is chosen and all genes of the chromosome after that point are swapped; analogously, one can define a two-point or, more generally, a multi-point crossover operator. The idea of mutation is to introduce new genetic material into the population by altering an individual's chromosome in some random fashion. Commonly used mutation operators are ones that flip each gene in a chromosome independently with some probability p_m, or flip a fixed number of randomly selected genes; Figure 3.1 visualizes the interplay between (one-point) crossover and (bit flip) mutation. Different variation operators may be applied depending on the encoding of solutions. For a more comprehensive introduction to EAs including an overview of different selection and variation operators please refer to e.g. Bäck (1996); Bäck et al. (1997); Eiben and Smith (2003); Reeves and Rowe (2003).

3.1.1 Analyzing and explaining evolutionary algorithms

In the above, we have taken a rather operational view to introduce EAs. However, we did not answer the question of how EAs work under the hood or how new, on

average, fitter individuals are evolved. There are several alternative viewpoints on this subject and we introduce some of them in the following; for a more comprehensive overview please refer to e.g. Reeves and Rowe (2003).

The *schema theorem* of Holland (1975) was the first attempt to explain EAs, in particular genetic algorithms (GAs), from a theoretical point of view. This theorem is based on the concept of schemata, which are subsets of solutions that share some common properties. For instance, consider solution vectors to be binary strings of length $l = 5$. Then, the schema $H = (*1**0)$ describes the set of all solutions with value $x_2 = 1$ at gene 2 and value $x_5 = 0$ at gene 5; the $*$ is a wildcard symbol, meaning that a gene can have any possible value (thus in the binary case either value 0 or 1). The two properties of a schema are its *order* $o(H)$ and its *length* $l(H)$, representing the number of defined genes and the distance between the first and last defined gene position, respectively (Reeves and Rowe, 2003). Thus, for the above example we have $o(H) = 2$ and $l(H) = 3$.

The schema theorem relies on the assumption of *implicit* (or *intrinsic*) *parallelism* — the idea that information on many schemata can be processed in parallel. Based on this assumption, Holland argues that there are two competing forces in the GA: the force of selection increases the representation of fit schemata in line with their fitness, while the destructive force of crossover and mutation tends to break schemata up, in particular schemata of long and high order. Technically, we can define the theorem as

$$E[N(H, t+1)] \geq \left\{1 - p_c \frac{l(H)}{l-1} - p_m o(H)\right\} r(H,t) N(H,t), \tag{3.1}$$

where $E[.]$ is the expectation, $N(H,t)$ is the number of instances of schema H at time t, p_c and p_m are the probabilities of applying crossover and mutation, respectively, and $r(H,t)$ the fitness ratio expressing the average fitness of schema H relative to the average fitness of the population at time t. The theorem says that the expected number of instances of a schema H at time $t+1$ is greater than at time t if the schema is fitter than average, and its order and length are short. The third idea related to the theorem is that the surviving schemata, which tend to be fit schemata of short and low order, are recombined into larger schemata (*building blocks*) of potentially higher fitness; Goldberg (1989) calls this idea the *building block hypothesis*. Although Holland's schema theorem has been sometimes criticized, often because it was overinterpreted (Radcliffe, 1997), fundamental ideas

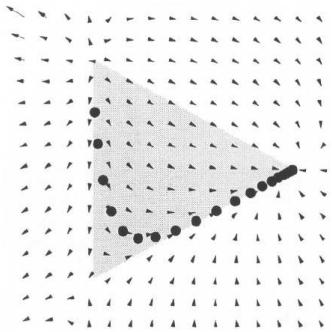

Figure 3.2: The simplex represents the space of populations containing three individuals. The flow indicates the movement of populations under the effect of selection using a dynamical systems approach. The trajectory represents the movement of a single population. Figure taken from Reeves and Rowe (2003).

have been proved to be valid. Some clarification of the schema theorem has been given by the work of Altenberg (1995). He uses *Price's theorem* (Price, 1970) to show that macroscopic properties of the population, such as the population average fitness, can be derived by accounting for properties related to the variation and selection operators employed, and the encoding of solutions.

An alternative perspective on EAs is to view transitions between populations as a *stochastic process* (Nix and Vose, 1992; Davis and Principe, 1993). In the simplest case, the population at time $t + 1$ depends only on the population at time t. This kind of stochastic process is referred to as *Markov process*, and the sequence of population transitions forms a *Markov chain*. With this approach, a transition matrix needs to be calculated to model the likelihood of an EA moving from a population at time t to any other possible population at time $t+1$ after applying the stochastic effects of selection, crossover and/or mutation. Markov chains are a convenient tool for analyzing EAs, which we will consider in Section 4.3 to investigate the impact of ephemeral resource constraints (ERCs) on evolutionary search; this section will also give a more detailed introduction to Markov chains.

A perspective on EAs related to Markov chains is to model EAs in terms of a *dynamical system* (Whitley, 1993; Vose, 1995, 1999). In essence, dynamical systems model the population as a state (expressed in terms of a probability vector) whose trajectory through the state space (see Figure 3.2) evolves according to deterministic, mathematical equations (defined by the effects of selection, crossover and mutation). The trajectory of the population is exact in the infinite population limit. However, as the number of equations to be solved depends on

the population size μ, in practice, this approach becomes impractical for large μ. Techniques that approximate the working of an EA in terms of large-scale or average properties of the system modelled, such as a *statistical mechanics approach* (Prügel-Bennett and Shapiro, 1994; Rattray, 1996), can help circumventing this complexity problem. Although, strictly speaking, this approach yields approximate results only (Reeves and Rowe, 2003), predictions of EA dynamics tend to be accurate, and this is also true for complex problems (Shapiro, 2001).

Various other perspective on the working principles of EAs can be found in the literature. For instance, underlying working principles of an EA can be viewed from an *experimental design perspective* (Reeves and Wright, 1995, 1996). The perspective taken by Mühlenbein (1991) is that the schema theorem cannot be used to explain the search strategy of a GA. Instead, he emphasizes the importance of constraining selection and mutation by a *spatial population structure*, and allow some or all solutions in the population to improve their fitness by *local hill-climbing* (Mühlenbein, 1992); these concepts have been realized by Mühlenbein et al. (1991) in the parallel GA. A perhaps more applied perspective is related to how EAs *balance exploration and exploitation* necessary for effective optimization. It is assumed that conventional EAs perform a highly explorative search at the beginning of the optimization process, which, as the optimization progresses, becomes a more exploitative one focused on the best solutions found so far. When balancing exploration against exploitation it is crucial to *maintain a diverse population*, i.e. solutions with different properties, when performing an explorative search (please refer to Section 3.4 for an overview of diversity-maintaining techniques). However, there are several aspects that may complicate the process of maintaining diversity in the population. Two common aspects are hitchhiking and genetic drift, which we describe next.

Hitchhiking (Mitchell, 1996) describes the scenario where newly discovered high-order schemata spread quickly through the population, slowing down the discovery of schemata in other positions; especially those that are close to the defined positions of highly fit schemata. *Genetic drift* (De Jong, 1975), on the other hand, refers to the statistical drift of gene frequencies in a population due to random sampling effects. It occurs because of the finite size of a population and the stochastic choices made in selection, population updates and genetic operations, whereby, due to accidents of sampling among the survived individuals, highly fit individuals may be lost from the population or the population may

suffer from a loss of variety concerning some alleles. One effect of drift is that a population might be pushed down a hill and finds itself in a less fit valley. Drift in combination with selection enables a population to move uphill (through the effect of selection) as well as downhill (through the effect of drift), and, of course, whether the population will find its way back to the same hill is not guaranteed. These concepts have been suggested by Wright (1932) and form part of his *shifting balance theory* of evolution. Geographic isolation of individuals and speciation are other parts of his model.

The choice of optimal parameter settings of EAs has also been approached from several theoretical viewpoints. For example, Goldberg et al. (1992) suggest that a linear dependence of the population size on the string length is appropriate. Empirical results of e.g. Grefenstette (1986), on the other hand, have indicated that a population size of 30 is sufficient for many applications. The choice of the mutation rate p_m has also been the focus of many investigations. The theoretical analysis of Mühlenbein (1992) suggests that a setting of $p_m = 1/l$ (with l being the genotype length) is generally adequate. Similar theoretical analyses have been conducted for the choice of the recombination rate and the selection pressure, which refers to the bias towards selecting fit individuals. The general conclusion is that the mutation and recombination rate should increase with the selection pressure (Bäck, 1996; Ochoa et al., 2000).

Recent theoretical studies on EAs focus rather on estimating the running time complexity of an algorithm as a function of the input space size l of a problem (see e.g. the work of Doerr et al. (2011); Auger and Doerr (2011); Oliveto and Witt (2011)). Such analyses can be performed using, for example, a *drift analysis* (Hajek, 1982; He and Yao, 2001).

Despite the drawbacks that some analytical methods might have, theoretical investigations of EAs contribute to the understanding of evolutionary optimization techniques. This in turn, helps us with the tasks of selecting and designing good strategy variants and operators. In other words, an improved understanding of evolutionary search allows for more effective tuning of EAs, a subject we consider next.

3.1.2 Tuning evolutionary algorithms

The objective of tuning an EA is to enhance its search performance. This process involves to select things like suitable operators and their parameter settings, a

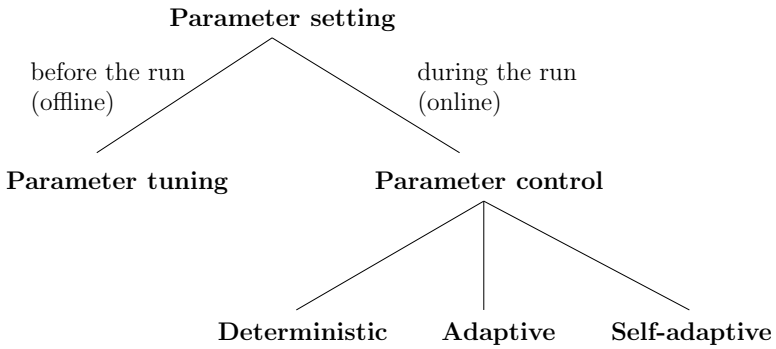

Figure 3.3: A classification of different parameter setting methodologies for EAs according to Eiben et al. (1999). A similar classification can be made for other design choices involved in EAs, such as operator selection, population size, etc.

representation of individuals, the population size and other EA parameters. We will now review a number of techniques that facilitate this tuning process.

Before tuning of an EA can be commenced, however, the goal of the experiment needs to be specified, such as designing a more robust, faster and/or cheaper algorithm. This helps to identify the hypotheses to test, appropriate test problems, and the performance measures to use. Let us assume that all these aspects have been determined, and we are now faced with the design and tuning of an efficient EA. We can adopt two fundamental methodologies to realize this tuning step: (i) tuning *offline* before running an algorithm on the actual problem at hand (this method is referred to as *parameter tuning*), and (ii) adapting parameter values *online* during an algorithmic run (this method is referred to as *parameter control*). Parameter control is sometimes further divided into *deterministic*, *adaptive*, and *self-adaptive* approaches (Eiben et al., 1999); this classification is also shown in Figure 3.3. In the following we will introduce commonly-used techniques to each method but a more complete overview has been given e.g. by Lobo et al. (2007); Fialho (2010); Eiben and Smit (2011).

A naïve technique for *parameter tuning* is to try out many different parameter settings by hand and then select the best setup (De Jong, 1975). Such an *ad hoc* approach, however, can be time-consuming and requires an *a priori* definition of certain aspects, such as the number of different configurations to be tested and the number of runs to be performed for each configuration. *Statistical racing methods* (Maron and Moore, 1997; Birattari, 2005) aim at eliminating some of these drawbacks. The idea here is to empirically evaluate a set of candidate configurations and successively discard bad ones as soon as sufficient statistical

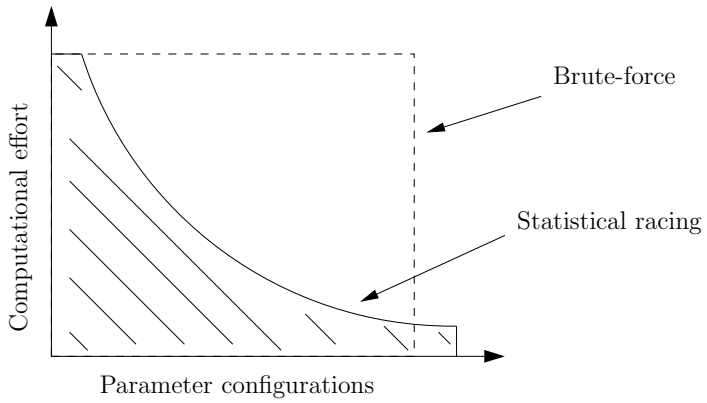

Figure 3.4: Comparison of the computational effort needed by a statistical racing method (dashed region) and a brute-force approach (rectangular area) to find an optimal parameter setting. While brute-force spends an equal amount of computation for all parameter configurations, a statistical racing method discards poor configurations as soon as sufficient statistical evidence is gathered against them. Figure reproduced from Birattari (2005).

evidence is gathered against them (Birattari et al., 2002); a visualization of this procedure is given in Figure 3.4. Another branch of techniques for parameter tuning is inspired by the experimental design discipline introduced in Section 2.4. Popular techniques adopted from this discipline include space-filling designs (Myers and Hancock, 2001) and response surface methods (Coy et al., 2001; François and Lavergne, 2001; Czarn et al., 2004; Bartz-Beielstein et al., 2004); in this context, a response surface represents the performance of an algorithm as a function of different parameter configurations. An approach that, similar to response surface methods, aims at tackling the optimization of expensive functions has been proposed by Corne et al. (2003); Rowe et al. (2006). The authors suggest to model a landscape as a finite state machine, inferred from preliminary sampling runs, and then run offline different EAs on this model to chose the most efficient one. An early drawback of this approach was its limitation to algorithms employing mutation only, but more recent versions of the technique also model crossover (Rowe et al., 2010), allowing both mutation and crossover to be tuned (in addition to population size and selection pressure). However, before (offline) tuning can be started, one needs to conduct a certain number of (expensive) evaluations on the real-world landscape to establish a sufficiently accurate model of the landscape. Finally, some researchers, such as Grefenstette (1986); Nannen and Eiben (2006); Branke and Elomari (2011), have formulated parameter tuning as an optimization problem and employed an EA as a meta-algorithm to tackle

this problem. The use of multi-objective EAs is particularly convenient when the aim is to find configurations that are optimal with respect to several performance measures (Dréo, 2009; Smit et al., 2010).

A general property of methods for parameter tuning is that the parameter configuration found (offline) is kept fixed during the optimization process of the actual problem at hand. This property is a drawback if the dynamic of an evolutionary search procedure is such that different parameter settings are more suitable at different stages of the search procedure. As an example, one would anticipate that a large mutation rate is beneficial at the beginning of the search to facilitate exploration, while a small mutation rate might be more suitable towards the end of a search procedure to promote exploitation of promising search regions. Methods that control parameters (online) during a run account for such dynamic scenarios. For instance, in the motivating example given above, the mutation rate may be changed according to some deterministic rule based on the evaluation or generation counter (Holland, 1975; Fogarty, 1989) (deterministic parameter control). More sophisticated techniques account for the state in which an algorithm is, for example, by monitoring the population diversity or the fitness improvement between generations, and then use these measurements to trigger changes in the parameter settings according to some heuristic rule (Rechenberg, 1973; Bäck, 1996) (adaptive parameter control); an example of an adaptive scheme is the famous "1/5 success rule" of Rechenberg (1973).[1] A different approach, which has its origin in the evolution strategy community, is to consider parameters like the mutation rate as additional genes in the chromosome and evolve them alongside with the actual decision variables of a problem (Schwefel, 1977; Hansen and Ostermeier, 2001; Beyer and Schwefel, 2002). This form of parameter control can be seen as a self-adaptive scheme; the obvious advantage of this scheme is that it does not require the algorithm designer to come up with a heuristic rule according to which parameters are changed.

Another example of self-adaptive parameter control is the dynamic selection of parameter configurations or operators based on their recent performance. Davis (1989) developed the first approach of this kind. He assigned each operator a probability of being selected in a reproduction process, and this probability was proportional to the fitness of individuals created with a particular operator. That

[1] This rule states that the ratio of successful mutations to all mutations should be 1/5; a successful mutation is one where the offspring is fitter than its parent. If the ratio is greater than 1/5, then the mutation step size should be increased, otherwise decreased.

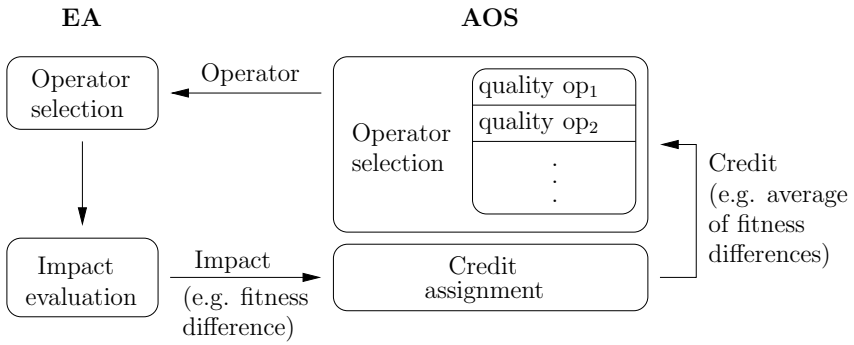

Figure 3.5: The general scheme of an EA employing an adaptive operator selection scheme: the EA asks for an operator and the operator selection mechanisms returns one based on the empirical quality estimates of the operators; after applying the operator, its impact is assessed (e.g. by the fitness difference between the generated individual and its parent); the credit assignment scheme transforms the the impact into a credit, which in turn is used to update the quality estimates of operators. Figure reproduced from (Fialho, 2010).

is, operators that are more successful in creating fitter solutions are assigned a greater probability for being selected in future. There is a variety of self-adaptive parameter control approaches of this form, and, sometimes, they are also referred to as *adaptive operator selection* (AOS) approaches (Fialho, 2010). AOS approaches are defined by two concepts (see Figure 3.5): (i) the way credit is assigned to operators upon generating an offspring (*credit assignment mechanism*), and (ii) the way this credit is considered in the selection of an operator (*operator selection rule*). The credit assigned to an operator can, for instance, be the fitness difference between the generated offspring and its parents (Tuson and Ross, 1998; Lobo and Goldberg, 1997) or the current best individual (Davis, 1989). To account for delayed credit effects, it has also been proposed to pass credit back to operators involved in the generation of ancestors of the offspring (Davis, 1989; Julstrom, 1995; Pearson et al., 2001). The operator selection rule first updates the empirical quality estimates of operators based on the credits obtained (e.g. by taking the average of credits obtained so far), and then uses these estimates to select the next operator. Operators can be selected, for instance, with a probability that is proportional to the quality estimates. Two popular operator selection rules that follow this procedure are *probability matching* (Goldberg, 1990) and *adaptive pursuit* (Thierens, 2005).

Other AOS approaches transform the operator selection rule into a *multi-armed bandit* (MAB) problem (Mahajan and Teneketzis, 2008). A MAB problem

is a non-associative, purely selectional learning problem originally designed to test the ability of an algorithm to trade off exploration against exploitation (Robbins, 1952a); MAB problems are related to problems in the field of reinforcement learning, which we introduce in Section 3.6. In a MAB problem one is faced with a number of actions, each action being associated with an unknown reward distribution from which a reward is drawn upon taking the action. The goal of the algorithm is to maximize the sum of rewards received over a number of actions taken. When considering operator selection as an MAB problem, the available actions would correspond to the operators, and the reward to some credit associated with the application of an operator. Fialho (2010) provides a comprehensive review of AOS approaches that consider operator selection as an MAB problem. In general, these approaches employ the same credit assignment schemes as other AOS approaches. However, their operator selection scheme employs a bandit algorithm, such as the *upper confidence bound (UCB) algorithm* (Auer et al., 2002). We will introduce this algorithm in detail in Section 5.2.2, where we investigate an AOS approach to deal with one of the resourcing issues (ephemeral resource constraints) considered in this thesis.

Note that parameter tuning and control has not only been considered for EAs but also for other *metaheuristics*, in particular local search algorithms. However, although these metaheuristics share many common features with EAs, they are not as deeply researched as EAs. In general, the field around self-tuning search heuristics is becoming increasingly popular in recent years. The recent self-* search workshop (Ochoa et al., 2010) and track (Ochoa and Schoenauer, 2011) held at the international conferences PPSN and GECCO, respectively, are further evidence of the burgeoning interest in this subject within the evolutionary computation community.

The pure task of tuning an EA can be accompanied by the task of actually trying to understand how different parameter configurations might affect search. For instance, to investigate the effect of different representations one may look at measures characterizing the problem difficulty (Rothlauf, 2006); examples of such measures include the *fitness-distance correlation* (Jones and Forrest, 1995) and the *negative slope coefficient* (Vanneschi et al., 2006). Distance measures that account for the choice of operators have been analyzed e.g. by Altenberg (1997). The concept of *heritability* (Mühlenbein and Paaß, 1996) is another measure that has been employed in the analysis of operators. For detailed descriptions of

these concepts please refer to the references provided. Of course, an understanding of the performance effect of operators and their parameter settings can also be gained from a theoretical point of view; the previous section has introduced various approaches that can be employed for this purpose.

Finally we want to remark that when tuning and comparing algorithms, one must not forget that according to the *no free lunch* theorem (Wolpert and Macready, 1997), all search algorithms have identical performance when their performance is averaged over all possible search spaces. Although this implies that there is no single general-purpose search algorithm, it also implies that, over a restricted set or class of problems (one which is not closed under permutation) (Schumacher et al., 2001; Igel and Toussaint, 2004), performance differences must exist and therefore a best algorithm or set of algorithms must also exist. In the following sections we will look at different problem classes, and successful strategies employed to tackle these problems; we begin with problems subject to some form of uncertainty.

3.2 Optimization in uncertain environments

Uncertainty in optimization can be due to a variety of sources. In this section, we focus on sources related to noise, and the search for robust and reliable solutions.

3.2.1 Optimization subject to noise

Uncertainty in the form of *noisy* (inaccurate) fitness values is a common side effect in problems where solutions are evaluated through physical experimentation or computer simulations (e.g. Monte Carlo simulations). For instance, when trying to find potent combinations of drugs, replicating the experiment of a drug combination will yield variations in the fitness measurements even if identical drugs and concentrations of drugs are used.

Noise can be attributed to a number of sources: Inaccuracies in realizing the decision variables of a genotype, e.g. concentrations of drugs, can introduce noise during the genotype-phenotype mapping, while measurement errors, human errors, and sampling errors can introduce noise during the fitness measurement process. Additional perturbations in the fitness may be due to uncontrolled environmental factors, such as ambient temperature and deviations in the performance of experimental equipment (e.g. instruments and robots).

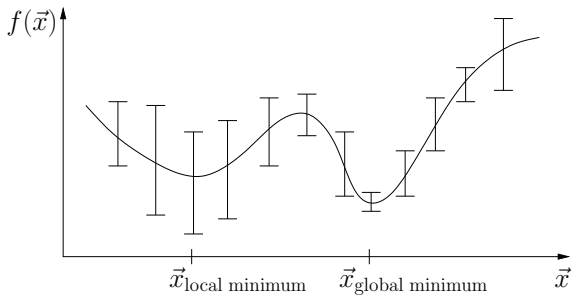

Figure 3.6: Visualization of the effect of a noisy fitness function f. The error bars indicate the level of noise for different solutions $\vec{x}$. There does not need to be a correlation between the noise level and the fitness. The large noise level at the local minima $\vec{x}_{\text{local minima}}$ may mislead the search to this region and thus away from the region containing the true optimal solution $\vec{x}_{\text{global minimum}}$.

Arguably the most common noise model considered is additive white Gaussian noise (Beyer, 2000), where noise is added to the true fitness value according to a Gaussian distribution $N(0, \sigma)$ with zero mean and standard deviation σ. Alternative noise models include uniform and Laplacian noise but these tend to be less common in real-world systems.

The challenge associated with noise is that one cannot be sure whether an improvement obtained by a decision variable change is a real improvement or the effect of noise (see Figure 3.6). In the extreme case of large noise, there is no useful selection information (because fitness improvements can be equally due to parameter changes and noise), causing the parameters to be optimized to perform a random walk (Beyer, 2000). However, noise is not always bad. A certain amount of it might be helpful during the initial phases of the search (Rana et al., 1996) as well as improve the reliability of convergence to the global optimum (Bäck and Hammel, 1994; Levitan and Kauffman, 1995) (because noise can allow an optimizer to escape from local optima).

The standard approach for coping with noisy fitness functions within EAs is to repeat the evaluation of a solution several times (referred to as the *sampling rate*) and then take the average over the fitness values to approximate its true fitness. This approach is referred to as *resampling* (Fitzpatrick and Grefenstette, 1988) in the evolutionary computation community; recall that the experimental design discipline refers to the same approach as *replication* (see Section 2.4). Obviously, an increase in the sampling rate results in a more accurate fitness approximation but this comes at the cost of a larger number of (potentially expensive)

evaluations. Suggestions on how to set or adapt the sampling rate, such as increasing the sample rate for individuals exhibiting high variances in their fitness value, and more sophisticated techniques, have been proposed e.g. by Aizawa and Wah (1994); Stagge (1998); Sano and Kita (2000); Branke and Schmidt (2004). The alternative to obtaining highly accurate fitness values is to perform 'rough' evaluations but allow an EA to consider many more candidate solutions per generation; this effect is achieved by an *increase of the population size* (Fitzpatrick and Grefenstette, 1988). It is unclear which of the two alternatives, accurate or rough evaluations, is more efficient, given a fixed number of fitness evaluations. Theoretical and empirical studies analyzing the trade-off between the sampling rate and the population size have been conducted e.g. by Fitzpatrick and Grefenstette (1988); Beyer (1993); Bäck and Hammel (1994); Hammel and Bäck (1994); Rattray (1996); Miller (1997); Beyer (2000).

The degree of the selection pressure (i.e. the degree to which fitter solutions are favoured in parental selection) has a significant performance impact in the presence of noise too. Several researchers, including Miller (1997); Beyer (2000), have shown that fitness proportionate selection is less sensitive to the level of noise than tournament selection. On the other hand, selection schemes with high selection pressure, such as tournament selection with relatively large tournament sizes (Beyer (2000) considers a tournament size of 5), can yield better final results.

Some researchers suggest to diverge from common parental and environmental selection schemes in the presence of noise. Markon et al. (2001), for instance, suggest to replace a parent by its offspring only if the offspring is fitter than some pre-defined threshold. On the other hand, Hughes (2001) suggests not to select the individual that has a higher mean fitness than some other individual but the one that has a lower probability of being worse than some other individual. In other words, Hughes recommends to minimize the probability of making a wrong decision. He analyzed this approach in a multi-objective optimization setup. Unfortunately, the number of studies considering noise and other forms of uncertainty within a multi-objective optimization is limited, and so is also our understanding in this domain.

The use of elitism to keep track of the best solutions found does not seem to improve the convergence behavior of an EA in the presence of noise; at least for some versions of evolution strategies (ESs) (Beyer, 1993, 2000). The reason is the risk of overemphasising elite solutions, which may misguide the search.

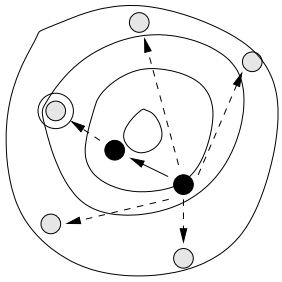

Figure 3.7: A (1,λ)-ES with $\lambda = 5$ using rescaled mutations. The five (gray) offspring candidate solutions are generated by mutating the (black) candidate solution in the center. All offspring are inferior to their parent. The parent of the next generation is constructed by taking a reduced mutation step in direction of the best offspring (circled). Figure reproduced from Arnold (2005).

To circumvent this issue, Büche et al. (2002) looked at various re-evaluation techniques and strategies that limit an individual's lifetime.

A noise-handling approach that has its origin in the ES community is the method of *rescaled mutations* (Rechenberg, 1994). The idea of this approach is to make large mutation steps when generating offspring in order to cancel out the effect of noise and thus increase the chance of selecting the single, truly best offspring (see Figure 3.7). Since the mutation steps are large, the offspring are likely to have a poorer fitness than the parent. The hope is that a reduced (rescaled) step towards the best offspring will yield a solution that is fit enough to replace the current parent and be carried over to the next generation. The obvious challenge with this approach is to appropriately set the rescaling factor; theoretical and empirical work addressing this challenge include e.g. (Beyer, 1998, 2000; Arnold, 2005).

Several other noise-handling strategies have been proposed over the years. The interested reader is referred to Jin and Branke (2005) for a review.

3.2.2 Searching for robust and reliable solutions

Let us now have a closer look at uncertainties associated with the search for robust and reliable solutions. We can define *robust solutions* as ones that are fitness insensitive to small perturbations in the decision variables or environmental conditions (e.g. manufacturing tolerances), and *reliable solutions* as ones that remain feasible or safe under such variation (Branke, 2001; Knowles, 2009). Perturbations in the optimization environment may result in a changing fitness landscape and thus cause a shift in the optimal solution. It is the general focus of the field

of dynamic optimization (see Section 3.4) to track these shifting optima. However, if shifts in global optima are random and small, one may prefer to seek a fit *robust solution* instead of tracking an optimal solution that may be located on a shaky peak (Branke, 2001). To search for robust solutions one can apply similar strategies as for noisy optimization, due to the similarities between the two optimization tasks (Jin and Branke, 2005): Taguchi's robust design methods are a popular approach originating from the experimental design discipline, while averaging techniques, population up-sizing, and other approximation techniques are more common within EAs. For an overview of approaches to robust optimization and robustness measurement techniques please refer to Beyer and Sendhoff (2007).

The search for *reliable solutions* is a task that has been considered only recently by the evolutionary computation community. In fact, this task may arise only when the optimal solution is located close to the constraint boundary. The common approach taken in this domain is to assume that the uncertainty of design variables and problem parameters is known in form of some probability distribution. This assumption allows then to transform the constraints into probabilistic constraints, and consider a solution as feasible if its probability of satisfying each constraint is greater than some pre-defined threshold. The advantage of this approach is that the constraints can be considered as deterministic, and standard techniques for coping with constraints (see Section 3.3) can be applied to find reliable solutions. However, commonly one is interested in the overall reliability (joint probability) of a solution in terms of satisfying all constraints rather than each single constraint on its own. It remains a challenge to compute this joint probability, and several approximation methods have been proposed to do this calculation. The interested reader is referred to Deb et al. (2009) for an overview of these techniques and the application of EAs to reliability-based optimization.

3.2.3 Further forms of uncertainties in optimization

There are two more main sources that can introduce uncertainty during an optimization process. The first is related to the problem of tracking shifting optima, mentioned above in the context of robust solutions. Here, uncertainty is due to the fact that it is often unknown when the environment changes, and in which direction the optimal solution shifts upon an environmental change. It is the focus of the dynamic optimization field to deal with these issues, and we will introduce

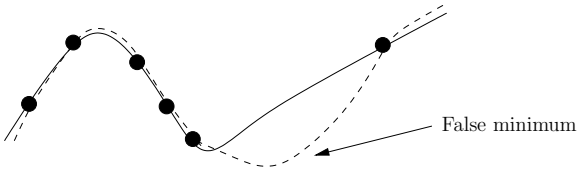

Figure 3.8: Visualization showing how an approximation error in a meta-model may lead to a false minimum. The solid line represents the true fitness function, the dashed line the approximated function, and the six dots the sample points. Figure reproduced from Jin (2005).

this field in more detail in Section 3.4.

Another common source of uncertainty is associated with the application of response surface techniques (also known as surrogates or meta-models in the context of evolutionary search). Remember that the main purpose of meta-models is to approximate the fitness function when fitness evaluations are expensive (see Section 2.5 and 2.6). However, whilst evaluating the real fitness function returns the true fitness of a solution, albeit sometimes with noise, fitness values taken from the meta-model may be subject to approximation errors (see Figure 3.8). For a review of techniques for coping with such approximation errors please refer to e.g. Jin et al. (2002); Ong et al. (2004).

Having gained an overview of different forms of uncertainty that may be encountered in optimization, we can generally conclude that in the presence of limited evaluation numbers one has to accept a lower solution quality. The question that arises is whether it is worth risking expensive evaluations to find the best optimal solution, or is it rather advisable to search for a robust or reliable but perhaps lower quality solution; work addressing this question include e.g. (Jin and Sendhoff, 2003). Although many real-world problems are subject to different forms of uncertainties simultaneously, there is unfortunately only little work in the literature considering approaches for such problems.

3.3 Constrained optimization

So far, we did not pay much attention to how constraints on the decision variables are dealt with during an evolutionary search. Roughly speaking, an optimization problem may be subject to two types of (static) constraints: box constraints (also known as parametric constraints or domain constraints) on individual decision variables, which are usually always present, and a set of linear and/or

non-linear constraints that further restricts the values of the decision variables. The issue when applying EAs to constrained problems is that (simple) search operators (and encodings) are blind to constraints and thus do not guarantee that feasible parent individuals will provide feasible offspring. This issue led to the development of many different constraint handling methods of which we briefly discuss some, but for a more detailed description we refer e.g. to Michalewicz and Schoenauer (1996); Coello (2002); Mezura-Montes (2009). Although dynamic constraints, such as ERCs considered in this thesis, are materially different to standard (linear and non-linear) constraints, as we shall see later, it is nevertheless informative to consider how constraints in EAs are usually handled.

Methods based on preserving feasibility of solutions. Some methods falling into this category preserve feasibility by making use of special representation schemes (Bean, 1994) and/or special operators (suitable for these schemes) (Michalewicz and Nazhiyath, 1995; Davis, 1991). For instance, Davidor (1991) used an EA with a variable-length representation to generate trajectories of a robot; and, to cope with this representation he designed an *analogous crossover*, which defines crossover points in parent chromosomes based on their phenotypic similarities. Other methods in this category utilize decoders (Koziel and Michalewicz, 1999; Kime and Husbands, 1997), which aim at mapping the original feasible region into another easier to optimize space, or scan the boundary of the feasible region using some strategy that coordinates jumps between the feasible and infeasible region (Glover, 1977; Schoenauer and Michalewicz, 1996). Note that the latter approach relies on the (often true) assumption that the global optimal solution is located at the boundary of the feasible region. The drawback of some of the strategies falling into this category is that they require an initial feasible solution. Michalewicz and Nazhiyath (1995), for instance, find this solution by iteratively generating random solutions in the search space until a feasible one is found. It is easy to see that this method can be time-consuming for applications where the feasible region is relatively small compared to the infeasible region.

Methods based on penalty functions. Although preserving feasibility is sometimes preferred, most evolutionary computation techniques use constraint-handling techniques that are based on the concept of (exterior) penalty functions, which in some way penalize infeasible individuals. The degree of the penalty

often depends on either the distance of an infeasible individual from the feasible region or on the effort to repair an infeasible individual. Different designs have been proposed for maintaining an appropriate balance between objective and penalty function. The simplest designs always maintain a constant penalty strength (static penalties) (see e.g. Homaifar et al. (1994); Michalewicz (1995)), or simply reject an infeasible individual and then generate new individuals until a feasible one is found (death penalties) (Rechenberg, 1973; Schwefel, 1975; Bäck et al., 1991). Other designs use a more sophisticated approach and increase the penalty strength gradually over time (dynamic penalties) (Joines and Houck, 1994; Kazarlis and Petridis, 1998), use concepts inspired by simulated annealing (Kirkpatrick et al., 1983) (annealing penalties) (Michalewicz and Attia, 1994; Carlson and Shonkwiler, 1998), or change the penalty strength depending on the quality of individuals in the population (adaptive penalties) (see e.g. Bean and Hadj-Alouane (1992)). And again another interesting approach, which is known as segregated genetic algorithm (Riche et al., 1995), aims at approaching the feasible optimal solution from both sides of the boundary of the feasible region by evolving a population that maintains roughly two subpopulations, one containing (more likely) infeasible individuals and the other one (more likely) feasible individuals. Recently, a (static) penalty function based method has been proposed that ranks individuals stochastically according only to their penalty function or to their fitness values (Runarsson and Yao, 2000). To reduce the number of expensive evaluations, Runarsson (2004) extended the stochastic ranking method further by fitting a response surface to both the penalty and the fitness function.

Methods based on a search for feasible solutions. There are three main techniques falling into this category. One technique, known as behavioral memory approach (Schoenauer and Xanthakis, 1993), tackles the constraints of a problem one after another whereby it is always ensured that a new constraint is only tackled if there is a sufficient number of individuals in the population satisfying all previously tackled constraints. A second group of techniques (see e.g. Powell and Skolnick (1993)) are similar to penalty functions and also penalize infeasible individuals but at the same time strictly favor feasible individuals over infeasible ones. The third main technique in this category incorporates the concept of repairing infeasible individuals. To date, many different repairing schemes as well as replacement strategies for dealing with the repaired (feasible) individuals have been proposed (see e.g. Liepins and Vose (1990); Nakano and Yamada (1991);

Liepins and Potter (1991); Nakano (1991); Orvosh and Davis (1994)); we will describe one repairing procedure, which is employed by the GENOCOP III approach of Michalewicz and Nazhiyath (1995), in more detail in Section 3.4.1. Typically, a replacement strategy is concerned with the question whether evolution should consider only the fitness value of a repaired solution (Baldwinian evolution), or the fitness and also the repaired solution (genotype) itself (Lamarckian evolution). For a recent survey of repairing methods we refer to Salcedo-Sanz (2009).

Hybrid methods. This group of constraint-handling methods contains EAs that are combined with other optimization techniques, which tend to be numerical-based. Hybrid methods include, amongst others, EAs that make use of Lagrange multipliers (see e.g. Adeli and Cheng (1994)), co-evolutionary models (Paredis, 1994), and concepts of ant colony optimization (Bilchev and Parmee, 1996).

Methods based on multi-objective optimization. Recently, the application of multi-objective optimization methods to constrained problems has gained popularity. Typically, these methods regard each constraint as an objective, which are then optimized alongside the actual objective function using techniques from multi-objective optimization (Montes, 2004). A comprehensive survey of multi-objective EAs and their application to constraints has been given by Deb (2001).

Considering the above constraint-handling methods, it is apparent that infeasible individuals (though unwanted) are often allowed to coexist with feasible individuals. The reason for allowing infeasible individuals to enter a population is that it may increase the probability of finding even fitter individuals, particularly on complex problems with rugged fitness landscapes and disjoint feasible regions. When dealing with problems where individuals violating a constraint cannot be evaluated, such as problems subject to ERCs, these techniques need to be modified or replaced by other techniques. Two simple techniques that do allow the evaluation of feasible individuals only are the death penalty and repairing. Alternatively, one can avoid evaluating an infeasible solution altogether and simply assign a poor fitness value to it. We will investigate the performance of these methods later in the presence of ERCs. Unfortunately, there is still little work on guidelines suggesting how to select the most appropriate constraint-handling technique and its parameters for a certain application.

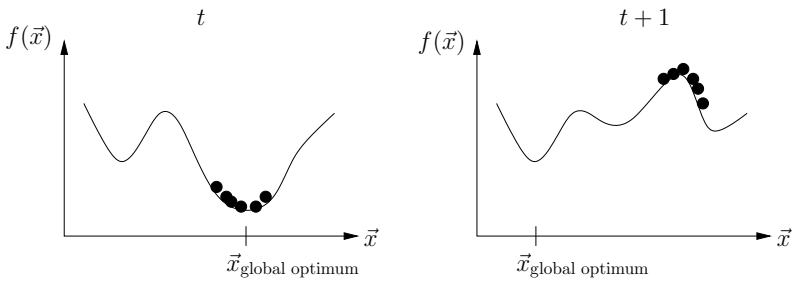

Figure 3.9: Visualization of how the location of the current best solution $\vec{x}_{\text{global optimum}}$ at time t (left) may shift at time $t+1$ (right) due to a change in the fitness function (solid line). This change requires the population (indicated by the black dots) to track the new optimal solution.

3.4 Dynamic optimization

The field of *dynamic optimization* is traditionally concerned with problems that are subject to stochastically changing fitness functions (landscapes). Reasons for a change in the fitness landscape can be, for instance, that new jobs to be scheduled arrive in a dynamic job shop scheduling problem, or that new cities to be visited have to be considered in a dynamic TSP. The natural objective when dealing with dynamic problems is to track the potentially shifting optima upon a fitness landscape change (see Figure 3.9). The book of Branke (2001) provides a comprehensive overview of research on how EAs can be used to achieve this objective. Let us briefly review some seminal work in this domain.

Tracking of shifting optima is initiated by a change in the fitness landscape. In applications like dynamic job shop scheduling and dynamic TSP (Larsen, 2000), it is explicitly known when the landscape undergoes a change. However, there are also applications where the point of time of an environmental change is not known explicitly. As an example, consider deviations in the quality of raw materials or wearing out of machines, both factors being common in engineering and manufacturing applications. Gradually or more suddenly, optimal operating conditions may shift unexpectedly in this case. Such applications require a mechanism that detects changes in the fitness landscape (and subsequently triggers the tracking of optima). A commonly-used mechanism to detect environmental changes is to watch out for deteriorations in the time-averaged best performance of the population (Cobb, 1990; Cobb and Grefenstette, 1993; Vavak et al., 1996). Alternatively, one may re-evaluate several individuals at each generation, and assume an environmental change when the fitness of at least one of these individuals

has changed (Branke, 1999; Carlisle and Dozier, 2000).

Various strategies have been proposed to track potentially shifting optima upon detecting an environmental change. A naïve one is to restart the optimization process from scratch (Raman and Talbot, 1993; Kidwell and Cook, 1994). In essence, this strategy turns a dynamic optimization problem into a problem where a sequence of independent static optimization problems is to be solved. Although being simple, this strategy can be effective if the fitness landscape undergoes a severe change; we will demonstrate this later in Section 6.1.5. However, most of the time, a restart policy is rather impractical as reusing genetic information obtained in the past can help to track optima quickly. There are many strategies that aim to tackle exactly this drawback of the restart policy. Roughly speaking, we can divide the working procedure of these strategies into three groups: diversity control, memory-based and multi-population approaches. We introduce each approach in more detail in the following.

Diversity control approaches account for a changing environment either by maintaining a diverse population throughout the search, or by increasing diversity whenever a change in the landscape is detected. Techniques falling into the former category include the set of *niching techniques* (Mahfoud, 1995). These techniques are also often augmented on EAs to tackle static optimization problems because they are supportive in detecting multiple optima. Examples of niching techniques include *fitness sharing* (Goldberg and Richardson, 1987) and *crowding* (De Jong, 1975; Mahfoud, 1995). Sharing aims to discourage over-populated regions in the search space. It achieves this by reducing the fitness of individuals by a factor proportional to the number of individuals located in the same region. Crowding aims to achieve a similar objective but does this by allowing offspring to replace one of their parents only if they are less-fit and similar to the offspring; Algorithm 3.2 shows the pseudocode of a simple but effective crowding procedure, *deterministic crowding*. Another approach to constantly maintaining a diverse population is to replace, at each generation, some part of the population with randomly generated individuals, also referred to as *random immigrants* (Grefenstette, 1992).

A common procedure for introducing diversity into the population upon detecting an environmental change is to temporarily increase the mutation rate. While the method of Cobb (1990), *triggered hypermutation*, increases the mutation rate drastically for a single generation, the *variable local search* method

Algorithm 3.2 A single generation of deterministic crowding (Mahfoud, 1995)

Require: f (objective function), μ (parent population size), *Pop* (current population)
1: **for** $i = 1$ to $\mu/2$ **do**
2: generate two offspring $\vec{x}^{(1)}$ and $\vec{x}^{(2)}$ by selecting two parents, $\vec{p}^{\,(1)}$ and $\vec{p}^{\,(2)}$ randomly from *Pop*, and then recombining and mutating them
3: **if** (distance($\vec{p}^{\,(1)}, \vec{x}^{(1)}$) + distance($\vec{p}^{\,(2)}, \vec{x}^{(2)}$)) $\leq$ (distance($\vec{p}^{\,(1)}, \vec{x}^{(2)}$) + distance($\vec{p}^{\,(2)}, \vec{x}^{(1)}$)) **then**
4: **if** $f(\vec{x}^{(1)}) > f(\vec{p}^{\,(1)})$ **then** replace $\vec{p}^{\,(1)}$ with $\vec{x}^{(1)}$
5: **if** $f(\vec{x}^{(2)}) > f(\vec{p}^{\,(2)})$ **then** replace $\vec{p}^{\,(2)}$ with $\vec{x}^{(2)}$
6: **else**
7: **if** $f(\vec{x}^{(2)}) > f(\vec{p}^{\,(1)})$ **then** replace $\vec{p}^{\,(1)}$ with $\vec{x}^{(2)}$
8: **if** $f(\vec{x}^{(1)}) > f(\vec{p}^{\,(2)})$ **then** replace $\vec{p}^{\,(2)}$ with $\vec{x}^{(1)}$

of Vavak et al. (1996) increases the range of mutation (e.g. the number of decision variables mutated) slowly, depending on the search progress made after an environmental change. Several researchers including Angeline (1997); Bäck (1998) have also investigated the application of self-adaptive mutation rates (see Section 3.1.2) for dynamic optimization.

The idea of **memory-based approaches** is to augment an EA with the ability to store genetic information, which has been of high-quality in the past, and retrieve this information at some later stage of the optimization. Such an approach can be beneficial when parts of the environment/problem can return to a formerly visited state in which case the retrieved information may be used to speed up tracking of optima. Memory can be used in two forms (Branke, 2001): *implicitly* or *explicitly*. Implicit memory is provided through the use of a redundant representation. A popular approach is to use a diploid representation, i.e. one where each chromosome possesses two or more genes at each locus (one dominant and one recessive), combined with a dominance scheme that governs the expression of each allele in the phenotype (Goldberg and Smith, 1987; Ng and Wong, 1995; Hadad and Eick, 1997). A potential drawback of implicit memory is that the information stored may not be used in an efficient way. Approaches employing explicit memory aim at overcoming this drawback. For instance, Trojanowski et al. (1997) equips each individual of the population with the ability to remember the genotype of a fixed number of its ancestors. These ancestors are then re-evaluated upon an environmental change, and the fittest of them replaces the current individual if it is fitter. Whilst this is a simple example of an EA with explicit memory, it highlights some of the challenges associated with this form

of memory: it requires the algorithm designer to define what information should be stored in the memory, and what information should be retrieved under which circumstances.

The purpose of **multi-population approaches** is to evolve multiple populations simultaneously and thus explore multiple promising regions in the search space at the same time. Niching approaches have a similar purpose but they do not divide the population explicitly into multiple subpopulations. One of the early multi-population approaches is the shifting balance GA of Oppacher and Wineberg (1999). Inspired by the shifting balance theory of Wright (1932) (see Section 3.1.1), this GA splits the population into one large core population and a number of small colony populations (also referred to as *demes*). The purpose of the demes is to facilitate genetic drift, which was postulated by Wright to be necessary in the optimization of a fitness landscape. The core population is responsible for exploiting a promising search region, and it sends (forces) the colony populations out to explore new areas in the search space. Enforcement is achieved by a mechanism that pushes a colony away from the core whenever the two populations are too close to each other. In regular intervals, the core gains information about the colonies through immigration of colony members, allowing the core to shift its search focus to potentially more promising regions. A potential downside of multi-population approaches is the definition of several properties including the number and size of the different populations, the migration scheme, and the merging/'forking' scheme of populations. These properties tend to interact in sophisticated ways in modern multi-population approaches, such as the self-organizing scouts approach of Branke et al. (2000).

3.4.1 Dynamically-constrained optimization

Recently, evolutionary search has also been considered for continuous problems subject to *dynamic* constraints (Nguyen, 2011; Richter and Dietel, 2011; Richter, 2010). The purpose of these constraints is to simulate changes in the feasible search region occurring as a function of the evaluation or generation counter. As these constraints yield a moving feasible region, the location of the global optimum can move as well. Notice that constraints that prevent the temporary evaluation of solutions only, such as ERCs, do not cause a shift in the feasible region nor in the global optimum; (ERCs can cause a shift in the set of evaluable

solutions only but this will become clearer in the next chapter).

Similarly with static constraints (see Section 3.3), maintaining infeasible solutions in the population of an EA seems an important aspect in the presence of dynamic constraints. In fact, Singh et al. (2009) ensure that some fraction of the population consists of (fit) infeasible solutions by favouring them over feasible ones in environmental selection. The resulting mixed population can accelerate search along the constraint boundary.

An interesting approach for dealing with dynamic constraints has been proposed by Nguyen and Yao (2009a). Rather than tracking the moving global optima, it is proposed to track the moving feasible region by maintaining always a reference set of feasible solutions. Solutions belonging to this set can then be involved in the repairing procedure of infeasible solutions. To repair an infeasible solution, Ngyuen and Yao borrow the repairing procedure of GENOCOP III (Michalewicz and Nazhiyath, 1995). This procedure employs two stages: first, a path between the infeasible solution and a feasible solution (from the reference set) is created; then, along this path, solutions are generated at random until a feasible one is found, which replaces the original infeasible solution. Clearly, the drawback of this repairing approach is that it can be a time-consuming matter to generate a feasible solution (at random). To ensure the existence of a reference set of feasible solutions, solutions belonging to it are checked for feasibility every generation and repaired (using the same approach as above) if needed. Unfortunately, the authors do not specify how to proceed in the case where all solutions of the reference set become suddenly infeasible. This situation does not occur in GENOCOP III because the algorithm is designed for static constraints with the assumption that the population contains at least one feasible solution.

Clearly, our understanding of the effect of dynamic constraints on optimization is still limited, and more research is needed in this emerging field. The recent studies of Jin et al. (2010); Oh et al. (2011) suggest that understanding the dynamics associated with changing constraints can also be beneficial when dealing with problems featuring naturally static constraints. The objective of these studies is to efficiently tackle constrained problems; in particular, problems where the constraints yield disconnected feasible regions. The authors propose a dynamic approach that first builds an approximate model for each constraint function, and then uses this model to manipulate the feasible region. At the beginning of the optimization, the model enlarges the original feasible region to facilitate a less

constrained search. As the search proceeds, the feasible region is shrunk incrementally to eventually match the original feasible region. The hope is that this procedure enables an optimizer to jump more easily between the originally disconnected feasible regions. Of course, the assumption made here is that solutions that are infeasible to the original problem can be evaluated.

We will see later that ERCs are of dynamic nature too. In fact, rather than being only changed based on some counter or periodically, ERCs may arise due to some random event (e.g. a machine breakdown), or uncontrolled factors (e.g. delays in the delivery of raw materials required in an experiment). Perhaps more interesting is the scenario where ERCs arise due to previously made decisions. Such time-linkage scenarios are common in online optimization, which we will consider in more detail in the next section.

3.5 Online optimization

Online optimization deals with problems where, as time goes by, decisions have to be made without any knowledge of future events (Borodin and El-Yaniv, 1998). Generally, it is assumed that decisions cannot be revoked and thus that *time-linkage effects* (Bosman, 2005) may arise (we explain this effect in more detail below). For instance, in the online version of a TSP one needs to decide how vehicles should be routed to serve a new customer; and, in caching problems it needs to be decided which page should be removed from the memory (cache) if the cache is full. An *online algorithm* makes these decisions based on past events such that some cumulative score, e.g. miles driven by a vehicle or number of page faults, is optimized.

Before we say something more about online algorithms, we want to describe how the performance of these algorithms is measured in this domain as this is different from standard optimization. Is it easy to see that the quality of an online algorithm depends on the sequence of events it is faced with during an optimization run. The standard approach to measure this quality is to compare the performance of an online algorithm on each event sequence with that of an *optimal offline algorithm*, which has complete knowledge of the future. An online algorithm is said to be *competitive* when the ratio (also known as *competitive ratio*) between its performance and that of the optimal offline algorithm

is bounded. This performance analysis method is known as *competitive analysis* (Sleator and Tarjan, 1985), and it falls within the framework of worst-case complexity (Borodin and El-Yaniv, 1998). Alternatively, competitive analysis can be viewed as a game between an online player and a malicious *adversary*. The adversary (optimal offline algorithm) has knowledge about the online algorithm run by the online player, and it aims to construct an event sequence that is costly to process for the online player but inexpensive for the optimal offline algorithm. An aspect worth mentioning is that randomized online algorithms, i.e. algorithms that involve some random component in the decision making process to an event, are more difficult to fool by an adversary than deterministic online algorithms.[2]

The application of EAs to online optimization is not new. In fact, dynamic optimization problems are tackled in an online fashion if the dynamical changes are unknown beforehand. Note that an event sequence consists here of different fitness landscapes and/or constraints rather than a sequence of page requests to be dealt with. A challenge that has received less attention in the field of evolutionary online (dynamic) optimization are *time-linkage* effects (Psaraftis, 1988; Branke and Mattfeld, 2005; Bosman, 2005; Nguyen and Yao, 2009b), as Bosman termed it. A time-linkage effect refers to a scenario where decisions made now may influence the environment and score that can be obtained in future; in turn this may affect the cumulative score that ought to be optimized. A simple example of a time-linkage effect is that visiting customers on time in a routing problem increases customer satisfaction and thus may result in additional customers (in certain geographical regions) in future.

A promising approach to cope with time-linkage effects seems to be to *anticipate* future changes. For instance, Branke and Mattfeld (2005) realizes anticipation by searching for solutions that are not only fit but also flexible in the sense that they can easily be adjusted to upcoming environmental changes. For a dynamic job-shop scheduling problem where new jobs arrive stochastically during the course of the optimization, the authors showed that the avoidance of early idle times can support the flexibility of a solution. Consequently, to facilitate the search for fit and flexible solutions, they extended the objective function to additionally penalize early idle times.

The approach taken by Bosman and his collaborators Bosman (2005); Bosman

[2]There are subtleties associated with how competitive analysis should be carried out and interpreted in the presence of randomized algorithms that we will not go into here.

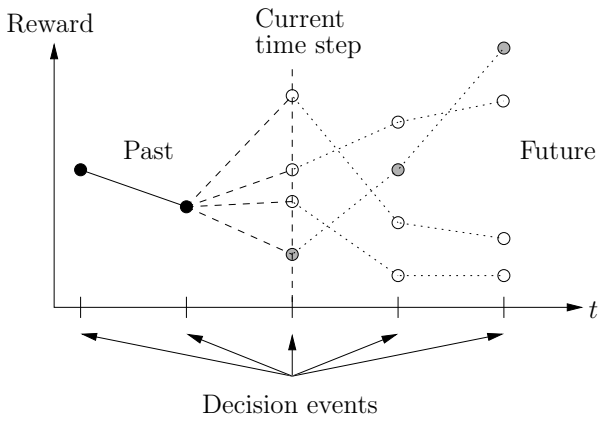

Figure 3.10: A schematic illustration of the effect of anticipation. Anticipation may help to make a decision that seems poor at the current time step but better in the long run (see path of gray dots). Figure reproduced from Bosman and Poutré (2007).

and Poutré (2007); Bosman (2007) to tackle online time-linkage problems is to predict (or anticipate) the future by learning from the past. Technically speaking, "the value of the dynamic objective function for future time steps is estimated by minimizing the generalization error over the time period that contains the data to learn from" (Bosman, 2005). The estimation is then used to select the search path with the highest objective function value (see Figure 3.10). Hutzschenreuter et al. (2010) applied this approach successfully to hospital resource management, where the task is to efficiently allocate resources to units of a hospital. In this application, the hospital environment was modelled by parametrized pathways of groups of patients (Hutzschenreuter et al., 2009). Anticipation was realized by simulating different resource allocation policies on this model offline (optimization of policy parameters was done using an EA) and then making allocation decisions online based on the policy that was optimal with respect to some future time span.

Later, when we deal with ERCs, we will encounter time-linkage effects too. However, there is a crucial difference between problems subject to ERCs and standard online dynamic problems, such as dynamic job-shop scheduling and the hospital resource management problem described above: while the objective in standard online (dynamic) optimization is to optimize some cumulative score, the objective in problems subject to ERCs is to find a single optimal solution. Optimizing some cumulative score is also the goal in the field of reinforcement learning (RL), which we introduce next, as a prelude to explaining how RL can be used to help tune EAs for challenging domains, such as problems subject to dynamic constraints.

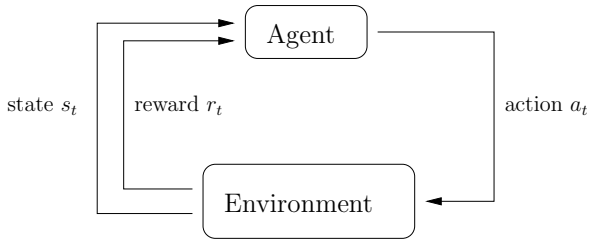

Figure 3.11: The agent-environment interaction in reinforcement learning: at each time step t, the agent receives some representation of the environment's state s_t and the reward r_t for reaching this state, and on that basis it selects an action a_t.

3.6 Reinforcement learning

The field of reinforcement learning (RL) offers a variety of ideas and algorithms that have been employed in the evolutionary computation community for tuning and controlling of EA operators and their parameters. We give a brief introduction to RL here, and review some work investigating evolutionary search in combination with RL in the following section. The interested reader is referred to the book of Sutton and Barto (1998) and the survey of Kaelbling et al. (1996) for a more complete overview of the RL field.

Generally speaking, RL is a computational approach to learning from interaction of an agent with an environment whereby the aim of an RL agent is to learn some *optimal policy* π^*, a mapping from an environmental *state* $s_t \in S$ to an *action* $a_t \in A(s_t)$ (see Figure 3.11), so as to maximize some *discounted return*:

$$R_t = \sum_{j=0}^{\infty} \gamma^j r_{t+j+1}, \tag{3.2}$$

where $r_{t+1} \in \mathbb{R}$ is a numerical *reward* received after taking action a_t in state s_t, and γ $(0 < \gamma \leq 1)$ is the *discount rate*. The discount rate determines the value of future rewards at the present time step. As the value of γ decreases, the importance of future rewards reduces so that in the case where $\gamma \approx 0$ the RL agent is myopic and aims only at maximizing the immediate reward. In learning problems where the agent-environment interaction can be divided into sub-sequences (known as *episodes*), such as single runs of an EA, an agent aims at maximizing the return at the end of each episode. Let us define the above mentioned concepts including the assumptions made in RL in more detail.

The common assumption made in RL is that environmental states satisfy the

Markov property, meaning that the state and reward at time $t+1$ depends only on the state and action at time t (rather than on the complete history of an agent), or:

$$P(s_{t+1} = s, r_{t+1} = r|s_t, a_t, r_t, s_{t-1}, a_{t-1}, ...r_1, s_0, a_0) = P(s_{t+1} = s, r_{t+1} = r|s_t, a_t). \tag{3.3}$$

It is common to assume the Markov property even when states do not summarize all the relevant information accurately (Sutton and Barto, 1998). A learning task that satisfies the Markov property is referred to as a *Markov decision process* (MDP). MDPs extend Markov chains (see Section 4.3.1) with the notion of actions and rewards. Equivalently, we can think of MDPs that feature a single action in each state, and a reward of zero, as Markov chains. To keep the number of actions and states in an MDP manageable, especially with respect to tabular-based RL algorithms (of which more later), it is common to quantize the state and action space if they are continuous.

In the presence of an MDP, some policy π, and the corresponding probability $\pi(s, a)$ of taking action a when in state s, we can calculate the expected return received after starting in some state s_t and then following π according to

$$V^\pi(s) = E_\pi \{R_t|s_t = s\} = E_\pi \left\{ \sum_{j=0}^{\infty} \gamma^j r_{t+j+1}|s_t = s \right\}, \tag{3.4}$$

where V^π is called the *state-value function for policy* π. In a similar fashion we can define the *action-value function for policy* π, $Q^\pi(s, a) = E_\pi \{R_t|s_t = s, a_t = a\}$, as the expected return received after taking action a in state s and following π thereafter. We can say that a policy π is better or equal to another policy π' if it has an expected return that is greater than or equal to that of π' for each state $s \in S$, or $V^\pi(s) \geq V^{\pi'}(s), \forall s \in S$. The goal of an RL agent is to find (ideally quickly and reliably) a policy π^* that is at least as good as any other policy; π^* is referred to as the *optimal policy*. The corresponding state-value function for an optimal policy, also called *optimal state-value function*, is defined by $V^*(s) = \max_\pi V^\pi(s), \forall s \in S$; we can define the *optimal action-value function* analogously by $Q^*(s, a) = \max_\pi Q^\pi(s, a), \forall s \in S, \forall a \in A(s)$.

Many RL agents search for an optimal policy by finding the optimal value function. If we have complete knowledge of an environment as an MDP, i.e. we know $\pi(s, a), \forall s \in S, a \in A(s)$, and the expected rewards associated with

Algorithm 3.3 Tabular TD(λ) for estimating V^π

```
1: V(s) = 0, e(s) = 0, ∀s ∈ S
2: for each episode do
3:    initialize s
4:    repeat
5:       take action a = π(s), and observe reward r and next state s'
6:       update e(s) for all s ∈ S according to Equation (3.7)
7:       update V(s) for all s ∈ S according to V(s) = V(s) + α(r + γV(s') − V(s))e(s)
8:       s = s'
9:    until s is terminal state
```

any state-action transition, then we can find an optimal policy using *dynamic programming* methods (see Chapter 4 in the book of Sutton and Barto (1998)). However, although many RL methods aim to achieve the same effect as dynamic programming methods, such methods are limited in their use because a complete model of the environment is rarely available and because they tend to be computationally expensive; thus we do not consider dynamic programming methods any further here.

In the common case where the environmental dynamics are unknown, the agent estimates the value function from experience it gains by interacting with the environment. A popular approach for learning from experience is the *temporal difference* (TD) method of Sutton (1988). This method updates the value of state s_t immediately after visiting that state. After taking action a_t in state s_t and subsequently observing a reward r_{t+1} and the new state s_{t+1}, the TD method updates the estimate $V(s_t)$ by

$$V_{t+1}(s_t) = V_t(s_t) + \alpha(r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)), \tag{3.5}$$

where $0 < \alpha \leq 1$ is the learning rate. As α increases the estimate $V_{t+1}(s_t)$ relies more on $r_{t+1} + \gamma V_t(s_{t+1})$ and less on its previous estimate $V_t(s_t)$.[3]

Equation (3.5) is the update rule used by a specific TD method, referred to as TD(0). A drawback of TD(0) is that updating a state-value estimate $V(s)$ only upon visiting the state s is rather inefficient and may require the agent a long time to estimate the correct value function. A more general version of the algorithm, TD(λ) (see Algorithm 3.3), updates $V(s)$ not only upon visiting s, but also upon visiting any subsequent state (during that episode). To realize this

[3]The term temporal difference refers to the difference between the two estimates $r_{t+1} + \gamma V_t(s_{t+1})$ and $V_t(s_t)$.

procedure, TD(λ) maintains a kind of memory, called the *eligibility trace* e. The task of the trace is to keep track of previously visited states that are eligible for being updated based on currently obtained rewards. To account for the trace, the update rule of $V(s)$ changes to

$$V_{t+1}(s) = V_t(s) + \alpha(r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t))e_t(s), \forall s \in S. \tag{3.6}$$

Eligibility traces can be updated in various ways. One example are *replacing traces*, which update the trace according to

$$e_t(s) = \begin{cases} 1 & \text{if } s = s_t, \\ \gamma\lambda e_{t-1}(s) & \text{otherwise,} \end{cases} \tag{3.7}$$

where $0 \leq \lambda \leq 1$ is the *decay factor* of the trace. For $\lambda = 0$ the update rule is equivalent to Equation (3.5). As λ increases, more importance is given to the actual reward received at each time step rather than the value estimates of subsequent states. Another popular trace option is an *accumulating trace*. This trace increases the eligibilities of states by some constant upon visiting them, rather than resetting them to a constant.

The update rules introduced above tell us how to *estimate the state-value function* V for an arbitrarily deterministic policy π. Ideally we would like to *control (improve) a policy* with the aim to approximate eventually the optimal policy π^*. One way to improve π is to select in each state s the action $a^* = \arg\max_a Q^\pi(s, a)$ that is expected to yield greatest return according to the estimates Q^π. Since $Q^\pi(s, a^*) \geq V^\pi(s), \forall s \in S$ (Bellman, 2003), the new policy will be at least as good as π. To be able to control a policy we thus must explicitly estimate the value of an action, or more generally the action-value function Q^π.[4]

Selecting always the best action seems to be a good strategy at first glance because it ensures that we optimally exploit the the current estimates. The drawback of this strategy, however, is that by taking always the current best action we will never be able to try new actions and thus limit our chances of finding an even better policy. Also, our current estimates may be inaccurate due to early sampling errors, preventing the truly best action from being selected. To overcome

[4]Clearly, if we would know the next state s_{t+1} we are moving to after taking an action a_t, then we do not need to estimate $Q^\pi(s_t, a_t)$. We could achieve the same effect by picking the action a_t that will yield the state s_{t+1} with the greatest state-value estimate $V^\pi(s_{t+1})$.

Algorithm 3.4 Tabular Sarsa(λ), an on-policy TD control method

1: $V(s,a) = 0$, $e(s,a) = 0, \forall s \in S, a \in A(s)$
2: **for** each episode **do**
3: initialize s, a (the action a can be e.g. selected at random from $A(s)$)
4: **repeat**
5: take action $a = \pi(s)$ (e.g. ϵ-greedy), and observe reward r and next state s' // the initial policy π can be e.g. to select a random action a from $A(s)$ in each state s
6: choose $a' = \pi(s')$ (e.g. ϵ-greedy)
7: update $e(s,a)$ for all $s \in S$ and $a \in A(s)$
8: update $Q(s,a)$ for all $s \in S$ and $a \in A(s)$ according to $Q(s,a) = Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))e(s,a)$
9: $s = s'$,$a = a'$
10: **until** s is terminal state

these issues an agent must explore new actions. A simple approach for trading off the exploitation of our current knowledge and the exploration of new actions is to select the currently optimal action a^* for most of the time and only occasionally, say with probability ϵ, a suboptimal one; this action selection method is referred to as *ϵ-greedy*. This way we allow each action to be selected with a non-zero probability $\pi(s,a) > 0$ at any time. Theoretically, this property ensures that in the limit as the number of actions increases, each state-action pair (s,a) will be sampled an infinite number of times, guaranteeing that the estimates of $Q(s,a)$ converge to the true values; in turn this means we can approximate the optimal policy (given we have infinite time). Exploration may also be encouraged by initializing all entries of $Q(s,a)$ with an optimistically large value (Schmidhuber, 1991). This way we ensure that all actions are selected at least once. One way to test the ability of an action selection method to trade off exploration with exploitation is to run it on a multi-armed bandit (MAB) problem (see Section 3.1.2 for a description of this problem).

An example of a TD algorithm with a policy control mechanism, Sarsa(λ) (Rummery and Niranjan, 1994), is shown in Algorithm 3.4. This RL algorithm belongs to the class of *on-policy* control method because it estimates the value of a policy while using it for control. RL algorithms that use two different policies for these tasks are known as *off policy* control methods; Q(λ) (Watkins, 1989) is a famous example of this algorithm class. The advantage of off-policy methods is that they allow learning of the value function for a policy even if experience is available from another policy. The obvious drawback, however, is that this may reduce

the learning speed. There are many more RL algorithms, each having its own weaknesses and strengths; for a review please refer to Sutton and Barto (1998).

3.6.1 Reinforcement learning in evolutionary optimization

The application of RL in optimization goes back to the seminal work of Zhang and Dietterich (1995, 1996); Zhang (1996), which investigated RL in the context of job-shop scheduling. The aim of this work was to learn a repair-based scheduler that incrementally repairs (resource) constraint violations with the goal to obtain schedules of short durations. Zhang and Dietterich considered complete schedules as states, and used domain-specific heuristics as the actions. The reward was a hand-crafted function based on the number of violated constraints and repairs performed. The RL agent learnt a policy offline on problems involving a small number of jobs, which was then tested on larger problems. Zhang and Dietterich demonstrated that the repair-based scheduler can outperform a non-learning iterative repair algorithm.

Within the evolutionary computation community, RL has been used for tuning and controlling of EA operators and their parameter values. Müller et al. (2002) were one of the first to combine evolutionary search with RL. They employed RL to learn online how to adapt the mutation step size within ESs depending on the success rate achieved with a mutation step size. The success rate after a fixed number of mutations represented the state, and the actions were the options to increase, decrease, or keep the current step size. The conclusion of this study was that the choice of the reward function as well as the lower and upper bounds of the mutation step size is crucial to obtain optimal performance. Eiben et al. (2007) investigated a similar approach for online adaptation of various parameters involved in a GA, including the mutation and crossover rate, tournament size used in the selection of parents, and population size.

Similar to the study of Zhang and Dietterich, Pettinger and Everson (2003); Pettinger (2002) learnt a policy offline using a test environment and then tested it in a different environment. The goal here was to learn when to switch between different crossover and mutation operators employed by an EA during an optimization process. Interestingly, the RL agent was also permitted to select the type of individuals (either a random individual from the fittest 10% of the

population, or one from the remaining population) to which the operators are applied. The (discretized) state space in this application was characterized by the current time step, the average population fitness, and the diversity of the population. And, the reward after applying an operator was the relative fitness difference between the fittest offspring generated and the fittest parent.

Recently, reinforcement learning has gained popularity in the field of *hyper-heuristics* (Cowling et al., 2000; Burke et al., 2009). A hyper-heuristic can be seen as a method that automates the process of selecting heuristics from a given set of (low-level) heuristics with the goal to improve the search performance. The heuristics may represent different operators or strategies needed to construct a solution. RL has been employed in this domain to select online between these heuristics; see e.g. Nareyek (2003); Meignan et al. (2010); McClymont and Keedwell (2011).

According to the classification of parameter setting methods of Eiben et al. (1999) (see Section 3.1.2), we can assign RL-based approaches that learn some policy π offline and online to the branch of parameter tuning techniques, respectively, adaptive or self-adaptive parameter control techniques.

3.7 Chapter summary

We have now reviewed a variety of techniques for analyzing and tuning of EAs to the main problem classes related to closed-loop optimization problems with resourcing issues. In particular, we looked at problems featuring uncertainty (e.g. noise), constraints, changing landscapes, and online decision making. Understanding the techniques reviewed is helpful when trying to tackle novel challenges in optimization. As we will see in the remainder of this thesis, many of the strategies we consider for coping with resourcing issues are related and inspired by ideas described in this chapter. The use of Markov chains for modeling EA dynamics, for instance, is employed in the next chapter to theoretically analyze the effect of ERCs on evolutionary search.

Chapter 4

Ephemeral Resource Constraints: Definitions and Concepts

In the previous two chapters we have introduced the class of closed-loop optimization problems, where solutions are evaluated by conducting experiments, and reviewed how evolutionary algorithms (EAs) have been extended to cope with some of the common challenges that arise in such problems and others, such as noise, constraints, changing environmental conditions, and real-time decision making. In this chapter we look at a related but novel challenge in closed-loop optimization deriving from resourcing issues. We introduce the framework to describe a particular resourcing issue: the temporary non-availability of resources required in the evaluation process of solutions. This resourcing issue yields an optimization problem with a static fitness function but dynamic constraints, which we call ephemeral resource constraints (ERCs). After formally defining problems subject to ERCs, we introduce a variety of ERC types that were encountered by Knowles (2009) in his collaborative work and that seem to be common in real-world applications. Finally, we present an initial theoretical investigation on the impact of ERCs on evolutionary search. The aim of this investigation is to gain initial insights into how ERCs can affect the performance of an EA for different configuration choices. Our analysis looks at configuration choices related to different selection and reproduction schemes used within a simple EA.

4.1 Ephemeral resource-constrained optimization problems (ERCOPs)

Compared with standard optimization problems, defining an ephemeral resource-constrained optimization problem (ERCOP) is more involved and requires knowledge of the closed-loop environment that is to be modeled. The reason is that the dependency of the ephemeral resource-constraints (ERCs) on some variables, such as time, function evaluations, costs and previous solutions evaluated, does not fit into the standard definition of an optimization problem. How exactly these variables are handled or simulated thus needs careful consideration.

Consider again Figure 2.1 of Section 2.1, which illustrates the basic setup of a closed-loop problem: Solutions are planned on a computer using an optimization algorithm, e.g. an EA, and then submitted for evaluation to the experimental platform. Depending on the platform, we may submit for evaluation a single solution, or, if solutions can be evaluated in parallel, then an entire population plan (i.e. a generation). When the experiment is completed, a generation of fitness values is returned, and the loop is repeated. But this iterative process is complicated by the fact that resources are needed to conduct the experiments, and these might run out e.g. as a function of time, or previous actions taken (or both). If the EA requests the evaluation of a solution that cannot be evaluated then something must happen, and this something must be defined. In a real experiment, a human might intervene to order some new resources, or allow the EA to generate some alternative solution(s).

The purpose of an ERCOP is to *simulate* the above kinds of experimental scenarios, including the way that non-evaluable solutions arise (i.e. as a function of parameters such as time, search history, or costs), and how they are to be handled.

4.1.1 General problem definition of an ERCOP

To define ERCOPs formally let us recall the definition of a generic optimization problem of Section 2.1, and add to it the notion of a time-ordered search, and the concept of a non-evaluable solution, as follows.

An optimization problem of generic form can be defined as

$$\begin{aligned} &\text{maximize } \; y = f(\vec{x}) \\ &\text{subject to } \; \vec{x} \in X, \end{aligned} \tag{4.1}$$

where $\vec{x} = (x_1, ..., x_l)$ is a solution vector and X a feasible search space. The static objective function (also known as fitness function) $f : X \mapsto Y$ represents a mapping from X into the objective space $Y \subset \mathbb{R}$.

A *black box optimization algorithm* a, e.g. an EA, for solving the above problem can be represented as a mapping from previously visited solutions to a single new solution in X, as suggested by Wolpert and Macready (1997). Formally, $a : \phi(t) \mapsto \vec{x}_t$, where the *search history* $\phi(t) = \{(\vec{x}_1, y_1), ..., (\vec{x}_{t-1}, y_{t-1})\}$ denotes the time-ordered set of solutions visited until time step $t-1$, and $\vec{x}_i$ and y_i indicate the X value, respectively, the corresponding Y value of the ith successive element in $\phi(t)$. We augment this notion of a search algorithm with the ability to visit a *null solution*, $x_{\text{null}} \notin X$, with the effect that the algorithm can 'wait' for a time step without evaluating a solution. An optimizer might submit null solutions, for example, if it wishes to wait until a missing resource is again available. In fact, this approach has been employed by Schwefel (1968) in his flashing nozzle problem (see preface of Chapter 1).

Now we are able to define the general ERCOP. While in standard optimization problems, the objective value $f(\vec{x}_t)$ of a feasible solution $\vec{x}_t \in X$ is $y_t = f(\vec{x}_t)$, in an ERCOP, we have

$$y_t = \begin{cases} f(\vec{x}_t) & \text{if } \vec{x}_t \in E_t(\sigma_t) \subseteq X \\ \text{null} & \text{otherwise,} \end{cases} \tag{4.2}$$

where $E_t(\sigma_t)$ represents the *set of evaluable solutions* (or *evaluable search region*) at time step t. In our case, $E_t(\sigma_t)$ is defined by a set of *schemata* into which solutions have to fall in order to be evaluable;[1] we introduced the concept of schemata in Section 3.1.1 in the context of the schema theorem.[2] The set $E_t(\sigma_t)$

[1]Notice that if schemata represent evaluable solutions, then E_t is the union of schemata. If schemata represent non-evaluable solutions, as it will be the case with random ERCs (see Section 4.2.4), then E_t is the complement of the union of schemata.

[2]Notice that in the context of ERCs, schemata are not related to the Schema Theorem or schema processing per se; here we use the concept of schemata merely as a convenient way to identify any subspace of the solution space in order to specify regions that become non-evaluable.

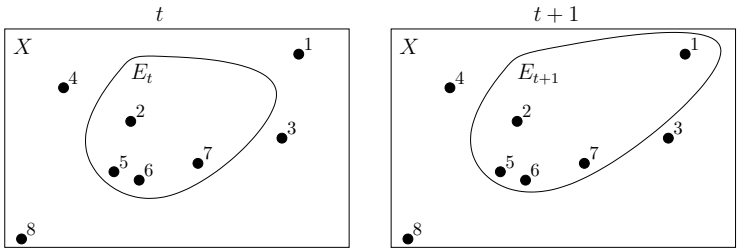

Figure 4.1: A schematic diagram showing how a population consisting of the solutions 1,2,3,4,5,6,7,and 8 might be distributed across the feasible search space X and the evaluable search space E_t. At time step t, only the solutions 2,5,6, and 7 can be evaluated while the solutions 1,3,4, and 8 must be repaired to be evaluable. Each evaluation might cause a change in E_t.

may *change over time* depending on a set of problem-specific and time-evolving parameters σ_t. The availability of resources required for the evaluation of solutions depends on these parameters. Thus, the set σ_t may include parameters such as various types of counters (e.g. cost, time and evaluation counters), the search history (which may be used to encode non-availabilities of resources due to previously made decisions), random variables (which may encode random events like machine breakdowns), and so forth. How exactly the set E_t changes depending on the parameters σ_t is modeled by the ERCs.

Note that the objective function f (thus also the global optimum) is *static* and does not change over time in a standard ERCOP; it is just the set of solutions evaluable at each time step t that may change. In this context, *repairing a solution* means to modify, through some repair function, the genotype of a solution that is not in E_t such that it is forced into E_t; i.e. the outcome of a repairing step is a solution that falls into all schemata that define E_t (assuming that the schemata are *non-contradictory*; i.e. $E_t = \emptyset$).

Compared to standard (dynamic) constraints, the meaning of ERCs is different: a solution $\vec{x}$ that violates an ERC at time t, or $\vec{x} \notin E_t$, is not *infeasible* but *non-evaluable* at time step t. That is, the experiment that is associated with $\vec{x}$ — be it involved in the genotype-phenotype mapping or the fitness assessment of the phenotype — cannot be conducted and thus the objective function is undefined (or null) for solution $\vec{x}$ at time t.[3] Figure 4.1 illustrates a common situation in an ERCOP with respect to the distribution of solutions across E_t and the feasible

[3]Notice the difference between a *null solution* and a solution with a fitness value of *null*: A null solution is submitted with the purpose to skip an evaluation, while a solution with fitness null is submitted with the purpose of being evaluated but then is not due to a lack of resources.

search space X.

Time in an ERCOP can be seen as the *simulated time* defined by the real closed-loop experimental problem that is to be simulated. Hence, time may refer not only to function evaluations of single solutions, as is the case in standard optimization problems, but also e.g. to real time units (e.g. seconds) or cost units (e.g. pounds). This notion of time allows ERCs to be dependent amongst others on the number of evaluated solutions, expenses, or a certain date, such as days of the week (independent of the number of evaluated solutions).

The normal assumption is that all evaluations take equal time or resources, but this need not be the case. Generally, experiments may be of different durations and have non-homogeneous costs in terms of the financial or temporal resources they require.

4.1.2 Further comments on ERCOPs

The definition of ERCOPs we gave in Section 4.1.1 represents the type of ERCOPs we consider in this thesis. There are several ways in which this definition can be extended.

We consider ERCOPs that feature a static objective function f. However, it is also realistic for ERCOPs to feature a dynamic function f (see Section 3.4) in addition to dynamic constraints. For instance, resources in form of raw material or instruments may be replaced with new ones during the optimization process, causing a change in f and potentially a shift in the global optimum. We will consider optimization problems with a dynamic function f when investigating problems subject to changing variables in Chapter 6, but this will not be done within an ERCOP framework.

Another extension to ERCOPs is to optimize not only a single objective f but several $f_1, ..., f_m$ (Deb, 2001). For instance, in a drug discovery problem one may not only aim to find potent drug mixtures but also ones that are unlikely to cause side effects, and involve a small number of drugs (as this reduces costs when manufacturing drug mixtures in mass quantities).

Moreover, the objective function f is typically noisy in closed-loop scenarios (see Section 3.2), and there may be standard constraints defining the feasible search space X (see Section 3.3). All these aspects are important to be accounted for but, for the sake of understanding the effect of ERCs on EAs, we believe it is best to start with a standard problem setup, meaning non-noisy objective values

and unconstrained search spaces.

Finally, we remark that an ERCOP can be cast as a reinforcement learning problem rather than a pure optimization problem. For various reasons that will become clear later, we postpone a discussion related to this subject until Chapter 7.

4.2 Specific types of ephemeral resource constraints (ERCs)

In this section we introduce four ERC types that were encountered by Knowles (2009) in his collaborative work and that seem to be common in real-world applications: commitment relaxation ERCs, periodic ERCs, random ERCs, commitment composite ERCs; Table 4.1 gives examples of these constraints (more detailed examples are given in the sections introducing the respective ERC types), and the Nomenclature section summarizes notation related to ERCs, which we will introduce in more detail next. Before we define each ERC type, we want to introduce three elements that are common to all ERC types considered. These elements are the constraint time frame, the activation period and the constraint schema.

4.2.1 Fundamental elements of ERCs

The *activation period* $k(ERC_i)$ of $ERC_i, k \in \mathbb{Z}^+$, is the number of counter units for which that constraint remains active, once it is 'switched on'. Similar to time steps, counter units may refer to function evaluations of a single solution, a set of solutions (in case experiments are conducted in parallel), real time units (e.g. seconds), a calendar period (e.g. Tuesday 2-4pm), or something else. In this thesis, they refer to function evaluations of a single solution.

The *constraint time frame* (ctf) of ERC_i is $\{t|t^{\text{start}}_{\text{ctf}}(ERC_i) \leq t < t^{\text{end}}_{\text{ctf}}(ERC_i)\}$ where t represents some counter unit, as above.[4] The constraint ERC_i may be active only during the ctf and not outside of the ctf. That is, if we assume an ERCOP to be subject to a single constraint, then we have $E_t(\sigma_t) \subseteq X, \forall t \in \text{ctf}$, and $E_t(\sigma_t) = X, \forall t \notin \text{ctf}$. The period of time $0 \leq t < t^{\text{start}}_{\text{ctf}}$ and $t^{\text{end}}_{\text{ctf}} \leq t \leq T$ (T is

[4]We are looking at the optimization scenario where the ctf is a single continuous period of time but it also realistic to have a ctf that is separated by unconstrained periods.

Table 4.1: Different types of ephemeral resource constraints (ERCs) and examples.

ERC	*Examples*
Commitment relaxation ERC (models the temporary commitment to a certain resource upon using it)	• Commitment to a certain instrument setting for some time once it is selected. • Commitment to a certain drug for some time (e.g. until the drug is used up) once it is given to the robot for mixing.
Periodic ERC (models availabilities of resources at regular time intervals only)	• Periodic availability of certain engineers required to operate certain instruments. • Periodic availability of certain drugs due to logistic limitations.
Random ERC (models temporary non-availabilities of randomly selected resources)	• Non-availability of certain instrument settings due to machine/instrument failures. • Delays in the delivery of drugs.
Commitment composite ERC (models scenario where variables of a solution define a composite, e.g. a tyre, that needs to be locally available for the solution to be evaluated. Composites need to be ordered in advance, are associated with a reuse number and shelf life, and have to be kept in capacity limited storage.)	• When optimizing the crash test abilities of a vehicle, the tyre setup may be subject to tuning. A tyre would represent a composite that costs money and must be ordered in advance, used up within a certain period, and involved in at most a certain number of tests; storage space for tyres may be limited too. • Drugs that must be ordered in advance and in batches only, have shelf lives, and need to be kept in capacity-limited storage.

the *total optimization time*) is the *preparation period*, respectively, the *recovery period* (see Figure 4.2). The duration of these two periods has a significant effect on the performance of an EA, as we will see later.

The restriction imposed by an ERC during the activation period can be of different forms. In our case, resources are associated often directly with individual solution variables, which allows us conveniently to use the notion of schemata to describe availabilities of resources. We say that solutions have to fall into a particular *constraint schema* $H(ERC_i)$ (associated with a constraint ERC_i) in order to be evaluable. From Section 3.1.1, where we introduced the concept of schemata in the context of the schema theorem, we know that a schema H represents a particular subset of solutions that share some common properties. To

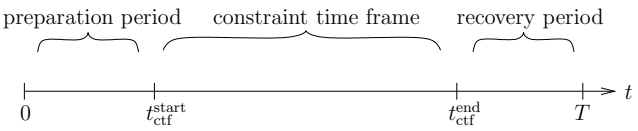

Figure 4.2: An illustration of how the available optimization time T can be divided into the preparation period $0 \leq t < t_{\text{ctf}}^{\text{start}}$, the constraint time frame $t_{\text{ctf}}^{\text{start}} \leq t < t_{\text{ctf}}^{\text{end}}$, and the recovery period $t_{\text{ctf}}^{\text{end}} \leq t \leq T$.

make it more specific, let us consider a drug discovery problem with the objective being to identify the most potent mixtures of drugs drawn from a library of size $l = 5$; a decision variable is binary indicating whether a drug is present (1) or absent (0) in a drug mixture, which is an entire bit string. In this example, the constraint schema $H = (*1 * *0)$ would describe the set of all solutions (drug mixtures) for which the drugs associated with the second and fifth bit position are present and absent, respectively. The wildcard symbol $*$ means that we do not care whether the drug associated with that bit position is present or absent in a mixture (i.e. bit values can be either 0 or 1). Recall also the two general properties of a schema, its *order* $o(H)$ and its *length* $l(H)$, representing the number of defined bit positions and the distance between the first and last defined bit position, respectively; for the above example we have $o(H) = 2$ and $l(H) = 3$. In the presence of multiple constraints ERC_i, solutions have to fall into the union of the schemata $\vec{x} \in \bigcup_i H_i$ associated with the constraints.[5] The meaning of the ERC related parameters is also summarized in the Nomenclature section.

In this thesis we consider discrete search spaces, mainly of pseudo-boolean nature or $X \in \{0, 1\}^l$. In non-discrete ones, H might restrict solution parameters to lie within or out of certain parameter value ranges rather than to take specific parameter values. This scenario may occur, for example, when optimizing the concentration of drugs involved in a drug cocktail. Here, we may have available only a limited amount of the individual drugs, causing the concentration of a drug to be upper bounded.

[5] Alternatively, the schemata H_i may represent resources that are not available for evaluation, thus define the evaluable region E_t negatively. In this case, solutions are evaluable if they are a member of the space spanned by the complement of the union of schemata or $\vec{x} \in \neg(\bigcup_i H_i)$.

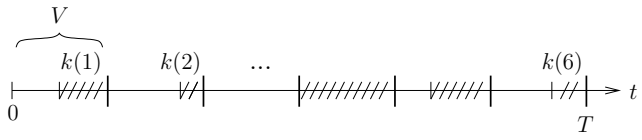

Figure 4.3: An illustration of how a commitment relaxation ERC may partition the optimization time into epochs of length V, and how it may be potentially activated. The activation period $k(j)$ during the jth epoch is represented by the dashed part.

4.2.2 Commitment relaxation ERCs

The first type of ERCs we introduce are *commitment relaxation ERCs*. Generally speaking, this ERC forces or commits an optimizer to a specific variable value combination (i.e. constraint schema) for some (variable) period of time whenever it uses this particular combination. Forcing a variable or linked combination of variables to be fixed for some time models real-world problems involving change-over costs of one sort or another. In particular, if changing a variable's value would incur some (large) change-over cost, such as a cleaning step, a component replacement, or a testing phase, then such changes to the variable may be made taboo for some period. Often, the change-over is much cheaper if done at a particular time step immediately *after* component replacements or cleaning (which is commonly done *routinely* rather than reactively), and so the variable can be allowed to change at that point.

A commitment relaxation ERC simulates the above scenario, so that some variable(s) setting (or schema) H, is forbidden from changing during a period of time we refer to as an *epoch*. We denote by V the duration of the epoch, and define the *activation period* $k(j), 0 \leq k(j) \leq V$, to be the duration of the period of time we have to commit to a particular setting H during the jth epoch. Note, the length of the activation period may change with each new epoch depending on when the particular setting H is selected by the optimizer. To describe the setting H we can conveniently use a constraint schema. For example, we would use $H = (*1 * *0)$ to state that a commitment is associated with the instrument setting for which the value of bit position 2 and 5 is set to 1 and 0, respectively.

Figure 4.3 illustrates the partition of the optimization time into epochs, and a possible distribution of activation periods. Imagine the six epochs illustrated by the figure to represent six working days, each consisting of $V = 9$ hours (assuming working hours to be from 8am to 5pm). The limitation that causes the commitment relaxation ERC to arise, can be:

Algorithm 4.1 Implementation of a commitment relaxation ERC

Require: $t_{\text{ctf}}^{\text{start}}$ (start of ctf), $t_{\text{ctf}}^{\text{end}}$ (end of ctf), V (duration of an epoch), H (constraint schema)

```
1: commitmentRelaxationERC(x⃗, t){
2: if t = 0 then
3:    last_activation = 0; k = 0    // initialize auxiliary variables
4: if t ∈ ctf then
5:    if t − last_activation ≥ k then
6:       if x⃗ ∈ H then
7:          last_activation = t; k = V − t mod V
8:       return true      // x⃗ is evaluable
9:    else if x⃗ ∉ H then
10:      return false     // x⃗ is not evaluable
11:   else
12:      return true     // x⃗ is evaluable
13: else
14:      return true }    // x⃗ is evaluable
```

"*In an optimization problem involving the selection of instrument settings, the configuration,* c, *once set, cannot be changed during the remainder of the working day.*"

In the above example, the constraint schema H represents the parameter combination that corresponds to instrument configuration c. The length of an activation period is bounded by $0 \leq k(j) \leq 9$. For instance, imagine we select instrument configuration c in the middle of the day, say at 1pm, as indicated by epoch $j = 1$ in the figure. This will activate the ERC for a period of $k(1) = 4$ (=5pm-1pm) hours (indicated by the dashed part). Activating the ERC later, earlier, or not at all during a working day changes $k(j)$ accordingly. Clearly, whether we actually evaluate solutions during an activation period (and thus use the setting c), or submit null solutions during this period, depends on the constraint-handling strategy employed; it is only that if we decide to evaluate a solution during an activation period, then it needs to be from H.

From Figure 4.3 it is also apparent that the total number of constraint activations during the optimization can vary between $0 \leq j \leq \lceil T/V \rceil$. That is, we might be lucky and the ERC may be never activated, e.g. if solutions belonging to H do not lie on an optimizer's search path, but already one activation may introduce enough solutions from H into the population such that future activations might be more likely.

Algorithm 4.2 Illustration of how we manage ERCs and the evaluation of solutions

Require: ERC_1,...,ERC_r (set of ERCs), f (objective function), T (time limit)
1: t (global time variable representing the current time step)
2: functionWrapper($\vec{x}, t$){
3: y_t=null
4: **if** $t < T$ **then**
5: **if** $\vec{x}$ is a null solution **then**
6: $\vec{x}_t = \vec{x}$
7: **else**
8: **if** $\vec{x}$ satisfies the ERCs ERC_1,...,ERC_r **then**
9: $\vec{x}_t = \vec{x}$; $y_t = f(\vec{x}_t)$
10: **else**
11: $\vec{x}_t$=repair($\vec{x}, t$); $y_t = f(\vec{x}_t)$
12: t++
13: return $\vec{x}_t$ and y_t}

The corresponding implementation of a commitment relaxation ERC is defined by Algorithm 4.1. The method *commitmentRelaxationERC*($\vec{x}, t$) is defined by the parameters $t_{\text{ctf}}^{\text{start}}, t_{\text{ctf}}^{\text{end}}, V$, and H, and it takes as input a candidate solution $\vec{x}$ that is to be checked for evaluability and the current (global) time step t. The output is a boolean value indicating whether $\vec{x}$ is evaluable or not. The method maintains two auxiliary variables, *last_activation* and k, required to update the internal state of the constraint: Line 5 to 7 are responsible for the activation of the ERC and the setting of the activation period, while Line 9 ensures that solutions have to be in H during an activation. In future, we will denote a commitment relaxation ERC of this form by *commRelaxERC*($t_{\text{ctf}}^{\text{start}}, t_{\text{ctf}}^{\text{end}}, V, H$). The meaning of the parameters related to this and other ERC types is summarized in the Nomenclature section too.

An extension to this simple commitment relaxation ERC is to maintain not only one but several commitment relaxation ERCs with different constraint schemata H_i. In this case, we need to consider three aspects: (i) a solution is non-evaluable if it violates at least one ERC, (ii) a repaired solution has to satisfy all activated ERCs and not only the ones that were violated, and (iii) it needs to be checked whether a repaired solution activates an ERC that was not activated before. We will consider this extension later in Section 5.1.4. At this point, we would like to describe how we manage an ERC method, such as the method *commitmentRelaxationERC*($\vec{x}, t$). To keep things simple, we design a *function wrapper*, as shown in Algorithm 4.2, that takes care of the management of ERCs

and the evaluation of solutions. Calling this wrapper is similar to calling the objective function f in a standard optimization problem. The wrapper is defined by the set of ERCs, ERC_1,...,ERC_r, an ERCOP is subject to, the objective function f, and the time limit T. It takes as input a solution $\vec{x}$ to be evaluated, and the current (global) time step. If the solution is a null solution (which is checked at Line 5), then the time counter is only incremented. If the solution is not a null solution and satisfies all ERCs (i.e. all ERC methods return *true* at Line 8), then it is immediately evaluated, otherwise it is first repaired and then evaluated. The output of the wrapper is the (repaired) solution and its fitness; the fitness is null, if a null solution is submitted. The repairing method *repair($\vec{x}, t$)* needs to be implemented by the optimizer. The structure of the function wrapper depends on the ERCOP to be simulated and thus needs to be specified alongside the ERCs.

4.2.3 Periodic ERCs

A *periodic ERC* models the availability of a specific resource, represented by a constraint schema H, at regular time intervals. That is, the ERC is activated every P time steps (*period length*) for an activation period of exactly k time steps (see Figure 4.4). As the ERC models the availability of resources, an individual has to be a member of H during the activation period. An example of a periodic ERC is:

"*In an optimization problem requiring skilled engineers to operate instruments, on Mondays, only engineer* eng_i *is available.*"

In the above example, the activation period is $k = 1$ (assuming a time step is a day), the period length is $P = 7$ (i.e. a week), and the constraint schema H represents the parameter combination that corresponds to the instruments (or their settings) operated by engineer eng_i.

The corresponding implementation of a periodic ERC is defined by Algorithm 4.3. The method *periodicERC*$(\vec{x}, t)$ is defined by the parameters $t_{\text{ctf}}^{\text{start}}, t_{\text{ctf}}^{\text{end}}$, k, P, and H, and it takes as input a candidate solution $\vec{x}$ that is to be checked for evaluability and the current (global) time step t. The output is a boolean value indicating whether $\vec{x}$ is evaluable or not.

In future, we will denote periodic ERCs by $perERC(t_{\text{ctf}}^{\text{start}}, t_{\text{ctf}}^{\text{end}}, k, P, H)$. In Algorithm 4.2, the method *periodicERC*$(\vec{x}, t)$ is called at Line 8. A potential

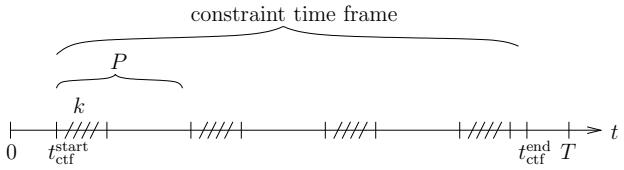

Figure 4.4: An illustration of a periodic ERC $perERC(t_{\text{ctf}}^{\text{start}}, t_{\text{ctf}}^{\text{start}}, k, P, H)$. The ERC is activated every P time steps for an activation period of always k time steps.

extension of a periodic ERC is that the period length and the activation period refer to different counter units. For example, consider the maintenance of machines. While maintenance might take hours (i.e. k might be measured in real time units), machines might need to be maintained after using them a certain number of times (i.e. P is measured in function evaluations).

4.2.4 Random ERCs

A *random ERC* models the temporary *non-availability* of randomly selected resources, whereby all resources correspond to constraint schemata of pre-determined fixed order $o(H)$. In other words, if a resource becomes unavailable, then we activate a new (plain) ERC for an activation period of k time steps and associate it with a constraint schema H for which the $o(H)$ order-defining bit positions and their values are selected at random. As the ERC models the non-availability of a resource, an evaluable solution *cannot be a member* of H; notice the difference to periodic and commitment relaxation ERCs where solutions have to be from H during activation periods. We allow multiple resources to be non-available at the same time, meaning an evaluable solution must not be a member of any of these constraint schemata. Let us denote the set of ERCs that is activated at time step t by $ActERCs_t$. We remove ERCs with an expired activation period from this set.

So far, we have not specified how a resource can become unavailable and thus how a new ERC is activated. Obviously, there are many options but since we want to use this ERC type to model random effects, such as unexpected machine breakdowns, we activate a new ERC at each time step with an independent probability of p; we call this probability the *activation probability*.

We want to avoid situations in which there is no evaluable solution. This is simply achieved by removing any ERCs from the set $ActERCs_t$ that have a constraint schema that is contradictory to the constraint schema of a newly activated ERC. Clearly, this can be realized differently. For example, by not

Algorithm 4.3 Implementation of a periodic ERC

Require: $t_{\text{ctf}}^{\text{start}}$ (start of ctf), $t_{\text{ctf}}^{\text{end}}$ (end of ctf), k (length of the activation period), P (period length), H (constraint schema)

```
1: periodicERC(x⃗, t){
2: if t ∈ ctf ∧ (t − t_ctf^start) mod P < k ∧ x⃗ ∉ H then
3:     return false      // x⃗ is not evaluable
4: else
5:     return true }     // x⃗ is evaluable
```

allowing a contradictory ERC to enter $ActERCs_t$ in the first place. Alternatively, we may allow contradictions and consequently be perhaps unable to evaluate any solution from time to time. Figure 4.5 shows how the set $ActERCs_t$ may vary over time and the effects of enforced ERC removals using our approach.

A simplified real-world example of a random ERC is:

"*In a multi-stage production process, a manufactured product runs through a series of stages, where, at each stage, the product is processed by a single machine which progresses it towards its final manufactured state. We wish to optimize the production facility/process in the following sense. For each stage there is a choice from among several machines (or machine types) capable of advancing the product to the next stage; we wish to select the best machine to use at each stage. Each machine may breakdown at any time step with a probability of 2% in which case it needs to be repaired; standard repairing takes 5 hours including test runs. Repairing time may increase if machine failures are more severe than expected* (A in Figure 4.5) *but, if necessary, a machine can be used before the test runs are finished* (B in Figure 4.5)."

In this example we have an activation probability of $p = 0.02$, an activation period of $k = 5$ (assuming a time step is one hour) and a fixed order of any constraint schema of $o(H) = 1$ (because a single machine might break down).

It is easy to see that a random ERC with a small p and k is unlikely to harm the optimization much (even if $o(H)$ high) as machines rarely break down and in case they do, they are quickly repaired. Clearly, the opposite is the case if either of the parameters is large; in this case, even a low order $o(H)$ can severely harm the optimization.

The corresponding implementation of a random ERC is defined by Algorithm 4.4. The method $randomERC(\vec{x}, t)$ is defined by the parameters $t_{\text{ctf}}^{\text{start}}, t_{\text{ctf}}^{\text{end}}$,

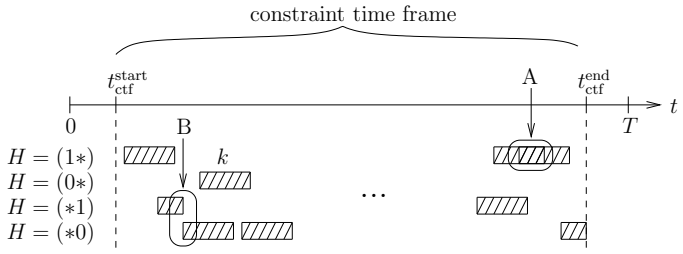

Figure 4.5: A random ERC with an order of $o(H) = 1$, an activation period of k time steps and an activation probability of $p \approx 2/k$ ($k \geq 2$) is applied to binary strings of length $l = 2$. The $4(=2^{o(H)}\binom{l}{o(H)})$ possible constraint schemata are represented on the left side. Each dashed block represents the activation period of an ERC in the set $ActERCs_t$. One can see that multiple ERCs can coexist, including ones with identical constraint schemata (A), and that an ERC is removed from $ActERCs_t$ if its activation period expires or if its constraint schema contradicts the one of a newly activated ERC (B).

$k, o(H)$, and p. The method takes as input a candidate solution $\vec{x}$ that is to be checked for evaluability and the current (global) time step t, and outputs a boolean value indicating whether $\vec{x}$ is evaluable or not. Inside the method, we use $H_i, i = 1, ..., |ActERCs_t|$ to refer to the constraint schema of the ith ERC in the set $ActERCs_t$ at time step t. The call to $rand(0, 1)$ returns a random, uniformly distributed real value between 0 and 1.

In future, we will denote random ERCs by $randERC(t^{\text{start}}_{\text{ctf}}, t^{\text{end}}_{\text{ctf}}, k, o(H), p)$. In Algorithm 4.2, the method $randomERC(\vec{x}, t)$ is called at Line 8. We want to point out here that we do not analyze the effect of random ERCs in this thesis; we included the definition of the ERC for completeness. For an analysis of random ERCs on evolutionary search we refer to a technical report (Allmendinger and Knowles, 2010).

4.2.5 Commitment composite ERCs

The last type of ERCs we introduce here are *commitment composite ERCs*; this ERC type is slightly more complex than the other types because it combines several real-world limitations. A commitment composite ERC occurs when some variables of a candidate solution define a *composite* that requires resources to be locally available (e.g. in a cache) in order for the solution as a whole to be realized and/or evaluated. We can conveniently use the notion of schemata to describe the resource-requiring composite part of a solution. For example, assuming a binary representation of solutions, we would use $H_{\#} = \{**\#\#\#*****\#\#\}$ to state that

Algorithm 4.4 Implementation of a random ERC

Require: $t_{\text{ctf}}^{\text{start}}$ (start of ctf), $t_{\text{ctf}}^{\text{end}}$ (end of ctf), k (length of the activation period), $o(H)$ (fixed order of a newly arising constraint schema), p (activation probability)

```
1: ActERCs_t = ∅ (set of ERCs activated at time step t)
2: randomERC(x, t){
3: if t ∈ ctf then
4:     remove any ERCs from ActERCs_t whose activation period is expired
5:     if rand(0,1)< p then
6:         H' = createH(o(H)) // create a new ERC of order o(H) at random at random, and
           time stamp the ERC with t to keep track of its activation period; add the ERC to
           the set ActERCs_t, and remove any ERCs from this set whose constraint schema is
           contradictory to H
7:     for i = 1 to |ActERCs_t| do
8:         if x ∈ H_i then
9:             return false      // x is not evaluable
10:    return true     // x is evaluable
11: else
12:    return true }   // x is evaluable
```

bit positions 3, 4, 5, 11 and 12 define a composite; we refer to the bit positions denoted by # as the *composite-defining bits*, and the order $o(H_{\#})$ to be the number of composite-defining bits in the schema (we refer to $H_{\#}$ as the *high-level constraint schema*). In the problems tackled in this thesis the composite-defining bits are static, and form a part of the ERC problem definition.

When a solution is to be evaluated, we must look at the composite-defining bits of its genotype and compare them to a local cache of composites (in Algorithm 4.5 this cache is represented by the variable *SCInfo*). Each composite in the local cache is indexed by a bit-string of the same length as the order of the high-level constraint schema. If there is a match, the solution can be evaluated; if not, the solution may not be evaluated at the current time step.

We define the cache to be made up of a number of storage cells, $\#SC$. Typically, the number of storage cells is smaller than the space of possible composites, which is $2^{o(H_{\#})}$ in a binary search space. A composite available in a storage cell may be used in the evaluation of more than one solution: each composite may be used up to RN (reuse number) times and has a shelf life of SL time steps, and we assume $SL \geq RN$. Finally, the composites available in the cache at time t are a function of previous purchase orders made, and a fixed time lag TL between a purchase being made and it arriving. When composites arrive at a particular time, they are immediately put in a storage cell (and any existing composite in that cell is discarded); which storage cell is selected is defined either at the time of purchase or at the time of arrival (see Line 4 and 5 of Algorithm 4.5).

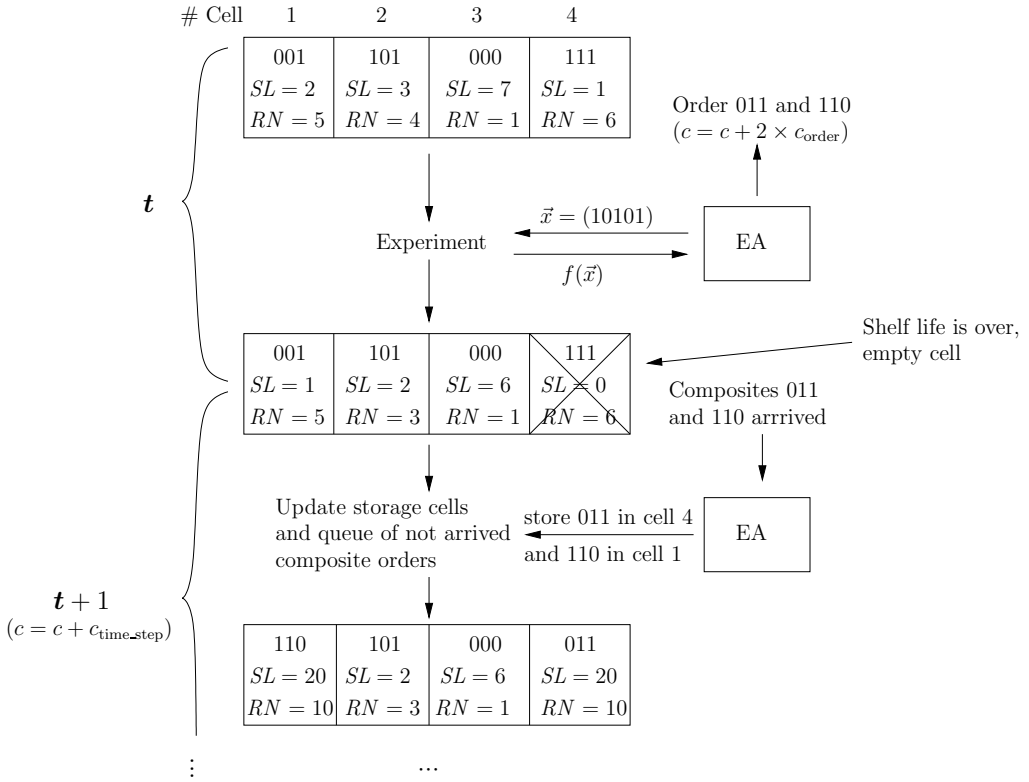

Figure 4.6: A visual example of the commitment composite ERC $commCompERC(H_{\#} = \{\#\#\# * *\}, \#SC = 4, TL = 1, RN = 10, SL = 20)$; each composite order and time step costs c_{order} and $c_{\text{time\_step}}$ units, respectively. The evaluation step at time step t reduces the reuse number of the composite in cell 2. At the same time step, the shelf life of the composite in cell 4 expires, and two new composites are ordered. One time step later, $t + 1$, the ordered composites arrive and put into cells determined by the EA.

To make the constraint more realistic we associate costs of c_{order} and $c_{\text{time\_step}}$ units with each submitted composite order and time step, respectively. The available budget, which cannot be exceeded, is denoted by C; any composite can be purchased as often as desired, as long as we are within the budget. Figure 4.6 gives a visual example of the ERC.

An example of a commitment composite ERC is:

"*In an optimization problem involving the selection/design of vehicle parts least harmful to pedestrians in case of a crash, we wish amongst others to identify the most suitable configuration for the tyres of the vehicle. A tyre is made up of several parameters, such as size, thickness, and rubber material. Upon defining these parameters, we order the tyres, which is associated with a fixed cost of 500*

Algorithm 4.5 Implementation of a commitment composite ERC

Require: $t_{\text{ctf}}^{\text{start}}$ (start of ctf), $t_{\text{ctf}}^{\text{end}}$ (end of ctf), $H_{\#}$ (high-level constraint schema), $\#SC$ (number of storage cells), TL (time lag), RN (reuse number), SL (shelf life), c_{order} (cost per submitted composite order), $c_{\text{time\_step}}$ (cost per time step), C (budget)

$c = 0$ (cost counter), $SCInfo = \emptyset$ (storage cell information), $QInfo = \emptyset$ (composite queue information) // Initialize global state variables
commitmentCompositeERC($\vec{x}, t$){
if $t \in$ ctf **then**
    collect composite orders if within budget, and increment c by c_{order} for each submitted order; add all ordered composites to the queue *QInfo* and time stamp them with t; let the optimizer choose the storage cell indexes into which the ordered composites should be stored upon their arrival
    empty the storage cells in *SCInfo* that belong to composites whose shelf life is over; remove any composites from *QInfo* whose time lag is over, and add them to the storage by updating *SCInfo*; if the optimizer did not specify previously the storage cell indexes into which arriving composites should be put, then let it choose the cell here
    if $c < C$ **then**
        $P = \emptyset$ (auxiliary set containing the indexes of storage cells that contain a composite that matches the composite required by the solution $\vec{x}$; this set is only important when $\vec{x}$ requires a composite that is available in storage multiple times)
        for $i = 1$ to $|SCInfo|$ **do**
            if $\vec{x} \in H_i$ **then**
                $P = P \cup i$
        if $|P| = 1$ **then**
            increment the reuse number of the composite stored in the storage cell with the index represented by the element of P; if the composite is used up, then empty the corresponding storage cell
            return *true* // $\vec{x}$ is evaluable
        if $|P| > 1$ **then**
            let the optimizer decide from which storage cell the composite should be taken, and subsequently increment the reuse number of the selected composite; if the composite is used up, then empty the corresponding storage cell.
            return *true* // $\vec{x}$ is evaluable
        return *false* // $\vec{x}$ is not evaluable (because the composite needed is not available)
    return *false* // $\vec{x}$ is not evaluable (because we are above the budget C)
else
    return *true* } // $\vec{x}$ is evaluable

Pounds and a delivery period of 3 days. To allow for a valid assessment of tyres, a set of tyres can be involved in at most 5 crash test trials, and can be kept in storage for not more than 1 month. The storage itself is limited in size to 10 sets of tyres. Every day of crash testing involves a fixed charge of 3000 Pounds including things like labor, rent of venue, and electricity."

In this example a composite is a tyre and the composite-defining bits are the variables defining a tyre. Ordering tyres is associated with a time lag of $TL = 3$ (assuming a time step is one day), and tyres have a reuse number of $RN = 5$ and

a shelf life of $SL = 30$ (assuming one month consists of 30 days). The number of storage cells is $\#SC = 10$, and the costs associated with a composite order and time step are $c_{\text{order}} = 500$ Pounds and $c_{\text{time\_step}} = 3000$ Pounds, respectively.

The corresponding implementation of a commitment composite ERC is defined by Algorithm 4.5. The method *commitmentCompositeERC*$(\vec{x}, t)$ is defined by the parameters $t_{\text{ctf}}^{\text{start}}$, $t_{\text{ctf}}^{\text{end}}$, $H_{\#}$, $\#SC$, TL, RN, SL, c_{order}, $c_{\text{time\_step}}$, and C. It takes as input a candidate solution $\vec{x}$ that is to be checked for evaluability and the current (global) time step t, and outputs a boolean value indicating whether $\vec{x}$ is evaluable or not. The variables representing the cost counter c, storage cell information *SCInfo* (contents in form of schemata, remaining shelf lives and remaining reuse number), and the queue *QInfo* of submitted but not arrived composites, are global state variables and visible to the optimizer.

In addition to repairing, three reactive tasks need to be defined by the optimizer to deal with this ERC type: (i) ordering of composites (Line 4), (ii) assigning storage cell indexes to arriving composites (Line 4 or 5), and (iii) deciding which composite to use if multiple identical ones are available (Line 15). With this ERC, repairing involves modifying a non-evaluable solution such that it falls into the constraint schema of a composite that is currently in storage.

In future, we will denote a commitment composite ERC of this form by *commCompERC*$(H_{\#}, \#SC, TL, RN, SL)$.[6] In Algorithm 4.2, the method *commitmentCompositeERC*$(\vec{x}, t)$ is called at Line 8.

4.3 Theoretical analysis

Having defined ERCOPs and several ERCs, we are now ready to analyze their effect on evolutionary search. We begin with an initial theoretical investigation on the impact of periodic ERCs on two selection and reproduction schemes commonly-used within EAs; the next chapter will continue with an empirical analysis. The theoretical analysis uses the concept of Markov chains, which we first mentioned in Section 3.1.1. The next section gives a more detailed introduction to Markov chains. We then derive the Markov model (transition probabilities) that account for periodic ERCs in Section 4.3.2, and analyze the simulation results in Section 4.3.3. Finally, Section 4.3.4 summarizes the insights gained

[6] We leave out the variables $t_{\text{ctf}}^{\text{start}}, t_{\text{ctf}}^{\text{end}}, c_{\text{order}}, c_{\text{time\_step}}$, and C from *commCompERC*(...) for ease of presentation. They will be specified where appropriate.

from the theoretical study.

4.3.1 Markov chains

A *Markov process* is a random process that has *no memory* of where it has been in the past such that only the current *state* of the process can influence the next state. If the process can assume only a finite or countable set of states, then it is usual to refer to it as a *Markov chain* (Norris, 1998). In other words, assuming that $X_t, t = 0, 1, ...$ are random variables with variable X_t depending on X_{t-1} only, then we can think of a Markov chain as a sequence $X_0, X_1, X_2, \ldots$ of random events occurring in time (Reeves and Rowe, 2003). Suppose $S_0, ..., S_\mu$ are the $\mu+1$ possible values that each of the variables X_t can take. Then, a chain moves from a state S_a at time t, to a state $S_{a'}$ at time $t+1$ with a probability of $p_{a,a'} = P(X_{t+1} = S_{a'}|X_t = S_a)$. The probabilities $p_{a,a'}$ $(a, a' = 0, ..., \mu)$ are called *transition probabilities* and form the $\mu+1 \times \mu+1$ matrix $\mathbf{P}$, the *transition matrix*. Thus, the probability that the chain is in state S_a at time t is the ath entry in the *probability vector*

$$\vec{u}_t = \vec{u}_0 \mathbf{P}^t, \tag{4.3}$$

where $\vec{u}_0$ is the $(\mu+1)$-dimensional probability (column) vector that represents the initial distribution over the set of states.

When an EA is modeled by a Markov chain it is easy to see that the population is the natural choice for describing a state. The transition probabilities then express the likelihoods that an EA changes from a current population to any other possible population after applying the stochastic effects of selection, crossover and/or mutation. It is also possible to consider other effects such as noisy fitness functions (Nakama, 2008), niching (Horn, 1993) and elitism (He and Yao, 2002). Once the transition matrix is calculated it can be used to calculate a variety of measurements, such as the first hitting time of a particular state or the probability of hitting a state at all. An overview of tools of Markov chain analysis can be found in any general textbook on stochastic processes, such as the books of Norris (1998); Doob (1953).

The drawback of modeling EAs with Markov chains is that the size of the required transition matrix grows exponentially in both the population size and string length. To keep Markov chain models manageable it is therefore common to use small population sizes and string lengths (Goldberg and Segrest, 1987;

Horn, 1993). Other options, which allow the modeling of more realistic EAs, are to make simplifying assumptions about the state space (Mahfoud, 1991) or to use matrix notation only (Vose and Liepins, 1991; Nix and Vose, 1992; Davis and Principe, 1993).

4.3.2 Modeling ERCs with Markov models

In this section we derive the transition probabilities for EAs optimizing in the presence of periodic ERCs. Our Markov chain model is based on the model of Goldberg and Segrest (1987), which considers a simple environment composed of two individual types: Type A has always a fixed objective value (or fitness) of $f(A)$, while type B has a fitness of $f(B)$. This limitation allows for an intuitive definition of states. For a fixed population size of μ, there are $\mu+1$ possible states, where state S_a represents a population with a type A individuals and $\mu - a$ type B individuals. Furthermore, in this simple EA model we do not apply mutation and crossover such that an offspring shall be simply a copy of the selected parent.

Goldberg and Segrest (1987) used this model to investigate the effect of drift for a simple EA that used a generational reproduction scheme combined with fitness proportionate selection. They also extended the model to include mutation. Horn (1993) extended it further to include niching. We extend it to include periodic ERCs and use the resulting model to analyze the impact of the ERC on two selection strategies, fitness proportionate and binary tournament selection, and two reproduction schemes, generational and steady state reproduction, both without elitism.

4.3.2.1 Selection probabilities

Under *fitness proportionate selection* (FPS) we choose an individual of the current population to serve as a parent (in our environment, to be in the next population) with a probability that is proportional to its (relative) fitness. In our simple environment, the probability of choosing a type A individual for the next population while being in a state S_a is simply

$$P_a(A) = \frac{af(A)}{af(A) + (\mu - a)f(B)} . \tag{4.4}$$

As there are only two individuals types in total, the probability of choosing a type B individual is $P_a(B) = 1 - P_a(A)$. From the above equation it is apparent that once a uniform population is reached, i.e. $a = 0$ or μ, there is no chance of selecting individuals from the other type. Thus, the two corresponding states S_0 and S_μ are *absorbing states*.

Under *tournament selection* we first randomly select a number of individuals from the population (with replacement) and then perform a tournament among them with the fittest one serving subsequently as a parent. It is common to use a tournament size of two, which will also be used here; this selection strategy is known as *binary tournament selection* (BTS). The result of a tournament is clear: the individual with the higher fitness wins the tournament; there is a draw if an individual meets another individual with the same fitness in which case the winner is randomly determined; and an individual will be the winner of a tournament with itself. We distinguish two cases regarding the fitness of the individual types: (i) $f(A) = f(B)$ and (ii) $f(A) > f(B)$. For case (i) the probability of selecting a type A individual is

$$P_a(A) = \left(\frac{a}{\mu}\right)^2 + \frac{a(\mu - a)}{\mu^2}, \tag{4.5}$$

where the first term is the probability of selecting two type A individuals, and the second term the probability of selecting one type A and B individual (which is $2\frac{a(\mu-a)}{\mu^2}$) accounted for by the fact that the type A individual will win the tournament on average in half of the cases only (i.e. $2\frac{a(\mu-a)}{\mu^2}/2 = \frac{a(\mu-a)}{\mu^2}$). If $f(A) > f(B)$, then a type A individual will win all tournaments it is involved in, resulting in following selection probability:

$$P_a(A) = \left(\frac{a}{\mu}\right)^2 + 2\frac{a(\mu - a)}{\mu^2}. \tag{4.6}$$

4.3.2.2 Transition probabilities

In our environment, the transition probabilities depend on the selected reproduction scheme, which in turn depends on the selected selection strategy. We first consider a *generational reproduction scheme* as already used in the original genetic algorithm of Holland (1975); we denote this scheme by GGA. With GGA, the entire current population is replaced by the offspring population. That is, μ selection steps are carried out per time step (with replacement). Using the selection probability $P_a(A)$ either for FPS or BTS, the transition probabilities

$p_{a,a'} = P(X_{t+1} = S_{a'}|X_t = S_a)$ for GGA of moving at time t from a state S_a with a type A individuals, to a state $S_{a'}$ with a' type A individuals at time $t+1$, are defined as follows:

For $a = 0$

$$\begin{aligned} p_{a,a} &= 1 \\ p_{a,a'} &= 0, \quad a' = 1, ..., \mu. \end{aligned} \tag{4.7}$$

For $0 < a < \mu$ and $0 \leq a' \leq \mu$

$$p_{a,a'} = \binom{\mu}{a'} P_a(A)^{a'} (1 - P'_a(A))^{\mu - a'}.$$

For $a = \mu$

$$\begin{aligned} p_{a,a'} &= 0, \quad a' = 0, ..., \mu - 1 \\ p_{a,a} &= 1. \end{aligned}$$

With *steady state reproduction*, the population is updated after each selection step. Usually, an offspring individual replaces the worst individual in the population. This replacement strategy, however, is elitist and ensures that the number of the less fit individual type in the population does not increase. Thus, to allow for a fair comparison with GGA, an offspring does not replace the worst individual in the population but a randomly chosen one regardless of its fitness; we denote this reproduction scheme by SSGA (rri), where 'rri' refers to replacing a random individual. It has been shown elsewhere (Syswerda, 1991) that GGA and SSGA (rri) yield similar performance. Bearing in mind that one time step corresponds to one selection step with SSGA (rri), we obtain the following transition probabilities:

For $a = 0$

$$\begin{aligned} p_{a,a} &= 1 \\ p_{a,a'} &= 0, \quad a' = 1, ..., \mu. \end{aligned} \tag{4.8}$$

For $0 < a < \mu$

$$\begin{aligned}
p_{a,a'} &= 0, \quad a' = 0, ..., a-2 \\
p_{a,a-1} &= (1 - P_a(A))\frac{a}{\mu} \\
p_{a,a} &= P_a(A)\frac{a}{\mu} + (1 - P_a(A))\frac{\mu - a}{\mu} \\
p_{a,a+1} &= P_a(A)\frac{(\mu - a)}{\mu} \\
p_{a,a'} &= 0, \quad a' = a+2, ..., \mu.
\end{aligned}$$

For $a = \mu$

$$\begin{aligned}
p_{a,a'} &= 0, \quad a' = 0, ..., \mu - 1 \\
p_{a,a} &= 1.
\end{aligned}$$

The transition probabilities of either GGA or SSGA (rri) will be the entries of the transition matrix $\mathbf{P}$.

4.3.2.3 Constrained transition probabilities for a periodic ERC

We have mentioned in the previous section that GGA performs μ selection steps per time step, while SSGA (rri) performs one selection step per time step. To be able to compare the effect of an ERC on the two reproduction schemes, we thus express ERCs in this section in terms of selection steps rather than time steps.

Let us now derive the transition probabilities in the presence of a periodic ERC. For this, consider the general periodic ERC, $perERC(i\mu, (i+1)\mu, k, \mu, H = (A))$ $(i \in \mathbb{N}, 0 \leq k \leq \mu)$, which is activated at selection step $i\mu$ for a period of μ selection steps, i.e. one time step (or generation) for GGA and μ time steps for SSGA (rri). During the activation period of $k \leq \mu$ selection steps, we can only select (and evaluate) type A individuals. Let us assume that if we select a type B individual during this period, this individual is repaired by simply forcing it into the right schema; i.e. it is converted into a type A individual. This repairing procedure is a simple constraint-handling strategy for dealing with non-evaluable solutions; alternative constraint-handling strategies will be introduced in the next chapter. Before we derive the constrained transition probabilities for GGA we want to point out a few aspects:

- If we are in state S_0 and the ERC is activated, then S_0 is not an absorbing state anymore and we move directly to state S_k.
- As a population contains at least k type A individuals after lifting the constraint, we are not able to move to a state $S_{a'}$ with $a' < k$ during the constrained generation (time step).
- The ERC reduces the number of freely selected offspring down to $\mu' = \mu - k$.
- Moving to a state $S_{a'}$ with $a' > k$ is already achieved by selecting $a'' = a' - k$ (instead of a') type A individuals from the current population.

Considering these points, we derive for the time step for which the ERC is activated the following constrained transition probabilities for GGA:

For $a = 0$

$$\begin{aligned} p_{a,a'} &= 0, \quad a' = 0, ..., k-1, k+1, ..., \mu \\ p_{a,k} &= 1. \end{aligned} \tag{4.9}$$

For $0 < a < \mu$ and $0 \leq a < k$

$$p_{a,a'} = 0$$

For $0 < a < \mu$ and $k \leq a' \leq \mu$

$$p_{a,a'} = \binom{\mu'}{a''} P_a(A)^{a''} (1 - P_a(A))^{\mu' - a''}.$$

For $a = \mu$

$$\begin{aligned} p_{a,a'} &= 0, \quad a' = 0, ..., \mu - 1 \\ p_{a,a} &= 1. \end{aligned}$$

The above periodic ERC is set such that the activation period of k selection steps is upper bounded by the population size μ, and, in the case of GGA, starts and ends within a single time step (generation). This does not need to be necessarily the case. In fact, a periodic ERC can feature an activation period k that is so long that it constrains selection steps within two or more successive generations, or so short that several activation periods may start during a

single generation. In such scenarios, one needs to constrain all generations that are subject to constrained selection steps. The number of constrained selection steps within a generation, referred to as k in Equation (4.9), is then simply the sum of all selection steps that happen to be constrained during any particular generation. That is, depending on the ERC, the number of constrained selection steps may change between generations.

With SSGA (rri), the population is updated after each selection step, which remember is a single time step with this scheme. This means that we need to determine for each selection step (time step) separately whether it lies within the activation period and thus is constrained or not. During the activation period, the periodic ERC of above prevents us from moving from a current state S_a to a state S_{a-1}, which can only be reached if a type B individual replaces a type A individual. As above, if the constraint is active, then the state S_0 is not an absorbing state anymore, and we move directly to state S_1. We obtain the following new transition probabilities for each of the k constrained time steps:

For any $a = 0$

$$\begin{aligned} &p_{a,a'} = 0, \quad a' = 0, 2, 3, ..., \mu \\ &p_{a,1} = 1. \end{aligned} \tag{4.10}$$

For any $0 < a < \mu$

$$\begin{aligned} p_{a,a'} &= 0, \quad a' = 0, ..., a-1 \\ p_{a,a} &= \frac{a}{\mu} \\ p_{a,a+1} &= \frac{\mu - a}{\mu} \\ p_{a,a'} &= 0, \quad a' = a+2, ..., \mu. \end{aligned}$$

For any $a = \mu$

$$\begin{aligned} p_{a,a'} &= 0, \quad a' = 0, ..., \mu - 1 \\ p_{a,a} &= 1. \end{aligned}$$

We will denote the transition matrix with the constrained transition probabilities by $\mathbf{P}_c$.

4.3.2.4 Calculating proportions of individual types in a population

One way to analyze the impact of an ERC on different selection and reproduction schemes is to monitor the proportion of the two individual types in a population. To do so one needs to first calculate the probability of ending up in any of the possible states $S_i, i = 0, ..., \mu$ after t time steps. In an unconstrained environment, this can be done according to Equation (4.3) (see Section 4.3.1) using the transition matrix $\mathbf{P}$ derived in Section 4.3.2.2; in this equation, the μ+1 state probabilities at time t are represented in form of the probability vector $\vec{u}_t$. In a constrained environment we cannot use the transition matrix $\mathbf{P}$ across all t time steps but have to swap it with the constrained transition matrix $\mathbf{P}_c$ (see Section 4.3.2.3) for time steps that consist of constrained selection steps; this dependence of the transition matrix on time makes it a *non-homogeneous Markov chain*. Let us consider the same periodic ERC as in the previous section but this time with a constraint time frame spanning over $g \in \mathbb{N}$ periods (as opposed to exactly one), i.e. $perERC(i\mu, (i+g)\mu, k, \mu, H = (A))$. For this periodic ERC we can calculate the probability vector at any time step t for GGA as follows:

$$\begin{aligned}
\vec{u}_t &= \vec{u}_0\mathbf{P}^t, && 0 \leq t < i,\\
\vec{u}_t &= \vec{u}_0\mathbf{P}^i\mathbf{P}_c^{t-i}, && i \leq t < g+i,\\
\vec{u}_t &= \vec{u}_0\mathbf{P}^i\mathbf{P}_c^g\mathbf{P}^{t-g-i}, && g+i \leq t,
\end{aligned}$$

where the entries of the transition matrices $\mathbf{P}$ and $\mathbf{P}_c$ are calculated using Equations (4.7) and (4.9), respectively. The probability vector $\vec{u}_0$ of the initial state distribution has a value of 1 at the sth entry and a value of 0 in the others, if we want to start with a population of exactly s type A individuals.

One time step with GGA corresponds to μ time steps with SSGA (rri). To compute the probability vector $\vec{u}$ for SSGA (rri) we thus need to look at the state distributions at time step $t\mu$:

$$\begin{aligned}
\vec{u}_{t\mu} &= \vec{u}_0\mathbf{P}^{t\mu}, && 0 \leq t < i,\\
\vec{u}_{t\mu} &= \vec{u}_0\mathbf{P}^{i\mu}(\mathbf{P}_c^k\mathbf{P}^{\mu-k})^{(t-i)\mu}, && i \leq t < g+i\\
\vec{u}_{t\mu} &= \vec{u}_0\mathbf{P}^{i\mu}(\mathbf{P}_c^k\mathbf{P}^{\mu-k})^g\mathbf{P}^{(t-g-i)\mu}, && g+i \leq t,
\end{aligned}$$

where the transition matrices $\mathbf{P}$ and $\mathbf{P}_c$ are calculated according to the Equations (4.8) and (4.10), respectively.

Having obtained the probabilities of ending up in all the different states, we can calculate the expected proportions $c_t(A)$ and $c_t(B)$ of type A and B individuals in a population at time step t (or $t\mu$ in the case of SSGA (rri)) as follows (note that this calculation does not depend on whether a problem is subject to ERCs or not):

$$c_t(A) = \frac{1}{\mu}\sum_{j=0}^{\mu} j u_t^j \ , \ c_t(B) = 1 - c_t(A), \tag{4.11}$$

where u_t^j is the jth entry of the probability vector $\vec{u}_t$.

4.3.3 Simulation results

This section uses the measure of the expected individual type proportion to analyze the impact of period ERCs on two selection strategies, FPS and BTS, and two reproduction schemes, GGA and SSGA (rri). We consider first the case where both individual types have equal fitness values, and then the case where they are different. If not otherwise stated, the population size is set to $\mu = 50$.

4.3.3.1 Identical fitness values: $f(A) = f(B)$

In this case there is no selection pressure and thus both selection strategies behave identically. Ideally, an EA maintains an equal proportion of the two individual types in the population. However, because of genetic drift this is impossible and an EA eventually converges to a uniform population ($S_i = 0, n$). As the probability of ending up in one of the two states is proportional to the initial state, the expected individual type proportion is identical to the initial proportion, which is specified by $\vec{u}_0$. Thus, for a random initialization, the expected proportion is 0.5.

From Figure 4.7 we can see that an expected proportion of 0.5 is achieved until selection step 400 at which we activate the periodic ERC, $perERC(400, 450, 20, 50, H = (A))$, which has a unique activation period of $k = 20$ selection steps.[7] This ERC forces us to evaluate $k = 20$ type A individuals and subsequently, reduces (increases) the proportion of type B (A) individuals in the population. After the

[7]Note, in an EA performing optimization of a function, the number of performed selection steps displayed on the x-axes of Figure 4.7 would be equivalent to the number of performed function evaluations.

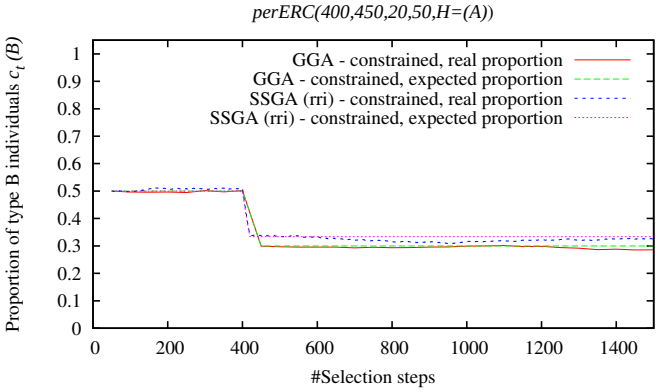

Figure 4.7: A plot showing the proportion of type B individuals $c_t(B)$ for GGA and SSGA (rri) as a function of the number of selection steps for the ERC $perERC(400, 450, 20, 50,\ H = (A))$. Both individual types have equal fitness and the constraint settings used are given above the plot. The terms *real* and *expected* refer to proportions obtained by actually running the EA, respectively, by running the Markov chain. The EA results are averaged across 500 independent runs.

ERC is lifted at selection step 420, the expected individual type proportion does not get back to the initial proportion. Although this effect can be put down to the specifics of the model (no selection pressure towards either individual type), we will see in the following theoretical and experimental studies, several results which display a similar pattern. That is, *a constraint can have a permanent or long-lived effect on search performance even if it was active for a short time only.*

From the figure we can also see that the proportion is affected more severely for GGA than for SSGA (rri). The reason that SSGA (rri) is more robust is that with this reproduction scheme there is a chance that an offspring of type A replaces another type A individual that is currently in the population. Of course, if an offspring replaces a solution of the same type, then this will not affect the proportion. By contrast, with GGA, all offspring are carried over to the population of the next generation. This causes the proportion of type B individuals in the population to be a linear function of the activation period. This effect is also apparent from Figure 4.8, where the performance of both reproduction schemes is shown as a function of the activation period k. From the figure one can see that SSGA (rri) is able to maintain a proportion of around $c_t(B) = 0.2$ after an activation period of $k = 50$, which is equal to the population size. On the other hand, GGA cannot maintain a single type B individual in the population because of its linear dependence on k. Note, in the case where $k > 50$, the constraint is activated for more than one time step when using GGA. For example, for $k = 70$

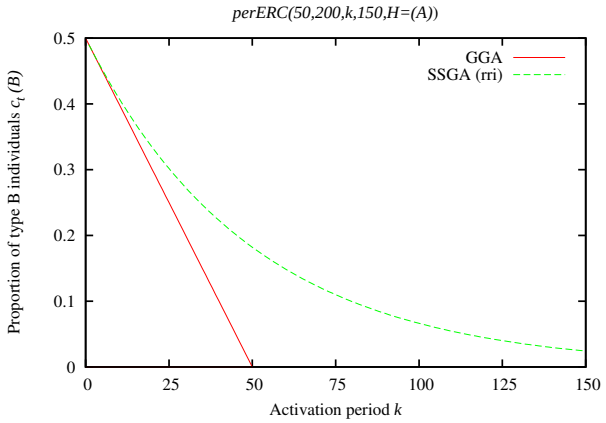

Figure 4.8: A plot showing the proportion of type B individuals $c_t(B)$ for GGA and SSGA (rri) at selection step 200 as a function of the activation period k for the ERC $perERC(50, 200, k, 150,\ H = (A))$. Both individual types have equal fitness.

the constraint restricts all 50 selection steps within one time step and 20 selection steps within the subsequent one.

As the Markov chain results are exact we will omit the experimentally obtained proportions in the following plots.

4.3.3.2 Different fitness values: $f(A) \neq f(B)$

When both individual types have different fitness values, the aim of an EA is to converge as quickly as possible to a population state consisting only of the fitter individual type. We focus our investigations mainly on the more interesting case where an ERC has a negative effect on the convergence behavior. Hence, the fitness of the individual type that we have to select during the activation period, in our case type A, needs to be lower than the fitness of type B individuals. If not otherwise stated, the fitness values are set to $f(A) = 1.0$ and $f(B) = 1.3$, meaning there is little selection pressure towards selecting the fitter individual type (which slows down convergence speed); later we will investigate the impact of the fitness ratio between type A and B individuals in more detail.

As the basis for our analysis we use the periodic ERC $perERC(50, 400, 20, 50,\ H = (A))$. This ERC is activated after the initialization (i.e. at selection or evaluation step 50) for seven periods, each consisting of $P = 50$ selection steps whereby $k = 20$ of them are constrained. Figure 4.9 shows the impact of the periodic ERC on the expected proportion $c_t(B)$ for all combinations of the selection and reproduction schemes: GGA with FPS and SSGA (rri) with FPS (top plot),

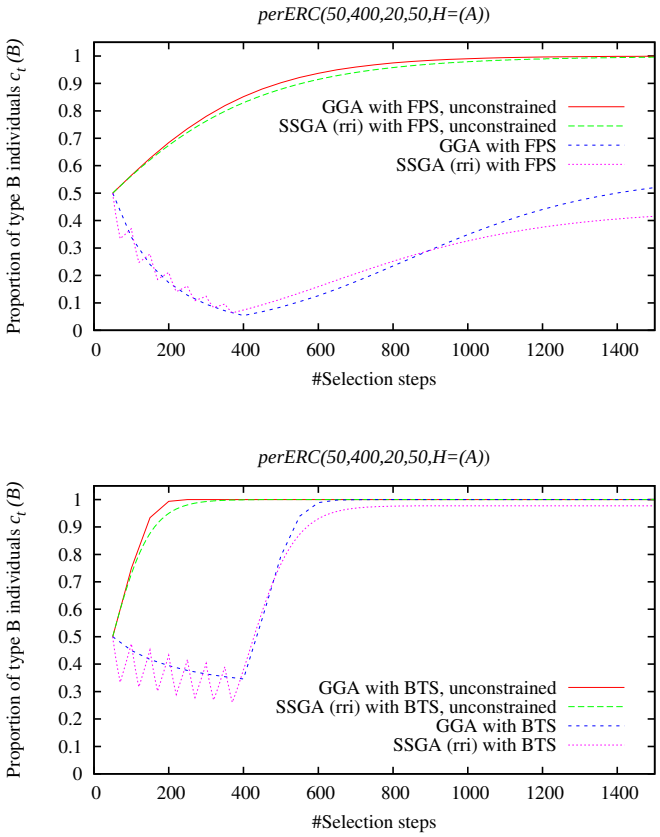

Figure 4.9: Plots showing the proportion of type B individuals $c_t(B)$ for FPS (top) and BTS (bottom) as a function of the number of selection steps for the ERC $perERC(50, 400, 20, 50,\ H = (A))$. The term *unconstrained* refers to the proportions obtained in an ERC-free environment.

and GGA with BTS and SSGA (rri) with BTS (bottom plot).[8]

We want to point out that during activation periods, SSGA (rri) with BTS and FPS perform identically, since independently of selection type, an A offspring will replace an individual selected at random. But during the inactive periods, the stronger selection pressure of BTS recovers more of the B-to-A replacements, so that overall BTS maintains a higher proportion of Bs. This behavior can be seen in the zigzag shape, where there is the same steep fall off of fitness in both methods, but a steeper recovery for BTS. Overall, the same is true for GGA, (BTS is better for the same reason) but it is not possible to see this so clearly in the plots.

Figures 4.10 to 4.12 indicate how the proportion of type B individuals is

[8]We get the zigzag-shaped line for SSGA (rri) during the constraint time frame because $c_t(B)$ is plotted after each time step containing here of one selection step. For GGA the change in $c_t(B)$ is smooth because a time step consist of μ selection steps.

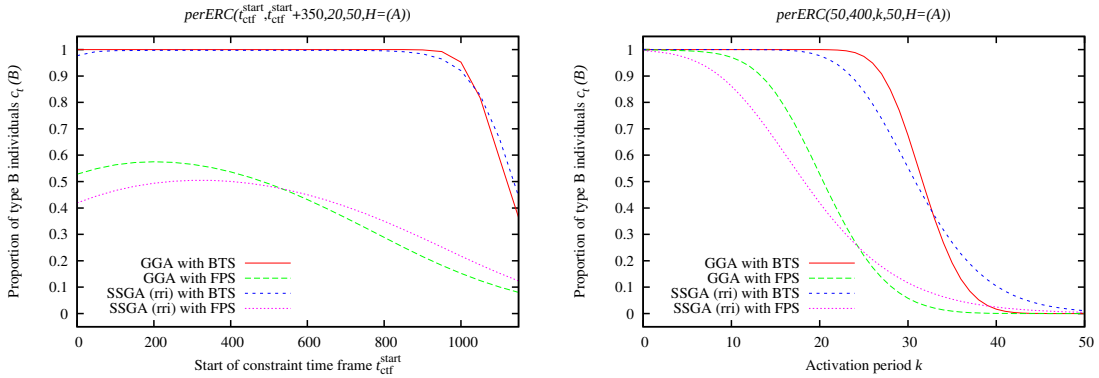

Figure 4.10: Plots showing the proportion of type B individuals $c_t(B)$ at selection step 1500 as a function of the start of the constraint time frame $t_{\text{ctf}}^{\text{start}}$ (left) and the activation period k (right) for the ERCs $perERC(t_{\text{ctf}}^{\text{start}}, t_{\text{ctf}}^{\text{start}} + 350, 20, 50,\ H = (A))$ and $perERC(50, 400, k, 50,\ H = (A))$, respectively.

affected when altering the constraint parameters. We can observe that:

- Longer activation periods degrade the performance of all EAs (see right plot of Figure 4.10).

- Fixing the constraint time frame duration, but translating it (see left plot of Figure 4.10), yields a non-monotonic effect on performance (of all EAs, but most apparently with FPS): more preparation time gives more time to fill the population with fit individuals, whereas little recovery time detriments final fitness. These two effects trade off against each other.

- Increasing the duration of the constraint time frame (see Figure 4.11) degrades performance.

- Changing the fitness ratio (see Figure 4.12) has only a switching effect on BTS (when the fitter individual changes), but for FPS the ratio smoothly affects final proportion up to a saturation point.

- Overall, comparing GGA with SSGA we see that SSGA achieves the higher proportion of fit individuals during the constraint time frame, and it recovers more rapidly after the constraint is lifted, but its rate of recovery does not reach the rate achieved by GGA, and ultimately GGA reaches a higher proportion (see Figure 4.8 and 4.9). This can be explained by the replacement strategy of SSGA (rri): offspring may replace individuals in the population that are from the same type. During the activation period,

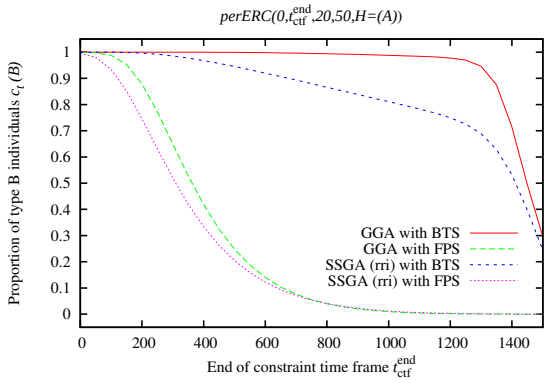

Figure 4.11: A plot showing the proportion of type B individuals $c_t(B)$ at selection step 1500 as a function of the end of the constraint time frame $t_{\text{ctf}}^{\text{end}}$ for the ERC $perERC(0, t_{\text{ctf}}^{\text{end}}, 20, 50,\ H = (A))$.

this is beneficial as the number of poor type A individuals in the population does not increase linearly with the activation period. However, during the unconstrained selection steps, this may be disruptive in the sense that fit type B offspring may replace other type B individuals of the current population, which slows down the convergence.

4.3.4 Summary of theoretical study

We used Markov chains to analyze the impact of periodic ERCs for a simple environment and EA model. The environment was composed of only two individual types and the EA model applied only a selection operator. In the EA model we considered two selection strategies, FPS and BTS, and two reproduction schemes, GGA and SSGA (rri). We observed that for one and the same reproduction scheme, BTS is more robust than FPS due to its independence to the fitness value of the individual types. However, FPS was able to match and even outperform the performance of BTS if the ratio of the individual type fitnesses was high, i.e. if a larger selection pressure than for BTS was obtained. The crucial difference between the two reproduction schemes we considered is that GGA carries out many selection steps before the population is updated, while SSGA (rri), or steady state reproduction in general, carries out only a single one. This enables SSGA (rri) during the activation periods to replace less fit individuals with other less fit individuals of the current population, but also prevents SSGA (rri) on the long run from a quicker convergence in the remaining periods. By contrast, the performance of GGA depends linearly on the activation period but there are

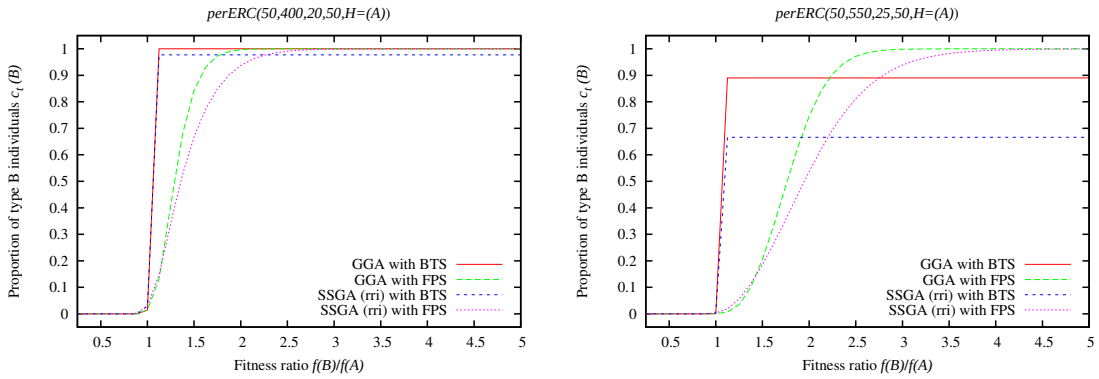

Figure 4.12: Plots showing the proportion of type B individuals $c_t(B)$ at selection step 1500 as a function of the fitness ratio $f(A)/f(B)$ for the ERCs $perERC(50, 400, 20, 50, H = (A))$ (left) and $perERC(50, 550, 25, 50, H = (A))$ (right).

now drawbacks if the ERC is not activated. This crucial difference between the reproduction schemes means that SSGA (rri) is able to outperform GGA during the activation period and in situations where the advantage over GGA gained in the activation period(s) can be maintained until the next activation period or until the end of the optimization. In terms of the constraint parameters, this occurs when there is a long activation period, a short recovery period, and the constraint time frame is set late.

4.4 Chapter summary

Ephemeral resource-constrained optimization problems (ERCOPs) are problems where feasible solutions can be temporarily non-evaluable due to a lack of resources required for their evaluation. In the first part of this chapter, we formally defined such problems, and introduced several types of real-world ephemeral resource constraints (ERCs).

In the second part of the chapter, we have conducted an initial theoretical analysis on the effect of one of the ERC types, periodic ERCs, on two selection and two reproduction schemes commonly-used within EAs. We employed the theory of Markov chains to model a simple optimization environment, and yielded two main conclusions: (i) binary tournament selection tends to be more robust and allows for a quicker convergence towards a better population state than fitness proportionate selection, and (ii) a non-elitist steady state reproduction scheme tends to be more robust than a generational reproduction scheme during periods

where the ERC is active, while the opposite is the case during inactive periods of the ERC. Although a larger theoretical investigation is necessary to make definite conclusions regarding other ERC types and more complex optimization environments, our findings indicate that the choice of algorithm setup is important to obtain optimal performance. Moreover, the analysis indicated that the optimal algorithm setups depends on the parameter settings of an ERC.[9] While this chapter has dealt with basic EAs only, the next chapter proposes and empirically analyzes the performance of a number of constraint-handling strategies (augmented on basic EAs) for coping with the different ERC types introduced in this chapter.

[9]In (Allmendinger and Knowles, 2010) we have empirically shown that the findings gained in our theoretical investigation can be valid also for more complex optimization environments, EA models, and other ERC types. We do not include this particular empirical analysis in this thesis but instead focus on the development and empirical analysis of different constraint-handling strategies.

Chapter 5

Strategies for Coping with Ephemeral Resource Constraints

In the previous chapter we formally defined ephemeral resource-constrained optimization problems (ERCOPs), and introduced several types of ephemeral resource constraints (ERCs). We also presented a theoretical study of the effect of one of the ERC types, periodic ERCs, on the choice of simple EA configurations, and observed that algorithm setup is crucial to obtain optimal performance. In this chapter we continue our analysis of ERCOPs using an empirical approach. Our objective is to develop and empirically analyze the performance of various strategies for coping with the ERCs introduced in the previous chapter. We first propose and investigate static constraint-handling strategies for dealing with non-evaluable solutions in Section 5.1. We then investigate whether learning online and offline when to switch between these static strategies during an optimization process is more efficient than using the static strategies themselves (Section 5.2). Finally, we develop online resource-purchasing strategies for coping with commitment composite ERCs in Section 5.3.

5.1 Static constraint-handling strategies

In this section, we introduce and analyze five static constraint-handling strategies for dealing with non-evaluable solutions in an ERCOP. We first introduce the strategies in Section 5.1.1. Then we outline the experimental setup as used in the analysis of the strategies in Section 5.1.2, and conduct the analysis itself in Section 5.1.3. The analysis will consider commitment relaxation ERCs and

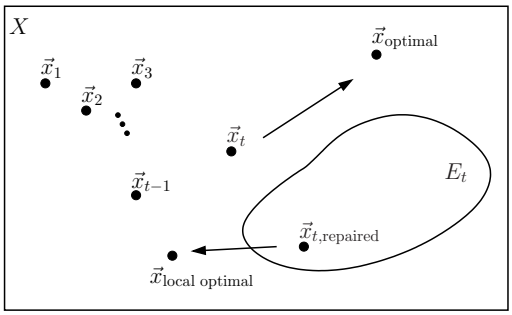

Figure 5.1: An illustration of how the targeted search direction of an optimizer (indicated by the arrow starting at $\vec{x}_t$), which might be towards an optimal solution $\vec{x}_{\text{optimal}}$, may change due to ERCs (which define the evaluable search space E_t). The reason is that repairing a solution causes a kind of 'drift' effect that alters the search direction, which, depending on the fitness landscape, may be towards a local optimum; the solutions $\vec{x}_1$ to $\vec{x}_{t-1}$ indicate the search history.

periodic ERCs (both ERC types were introduced in Section 4.2.2 and 4.2.3, respectively) but the strategies are applicable (in similar form) also to other ERC types. In Section 5.1.4 we present a case study that illustrates one way in which one might select a suitable strategy for an ERCOP with largely unknown search space properties, which is a common situation in the real world. In Section 5.1.5 we draw together the findings from the experimental analyses.

5.1.1 Specific static constraint-handling strategies

This section introduces five constraint-handling strategies for dealing with non-evaluable solutions in an ERCOP. Three of the strategies (forcing, regenerating, and the subpopulation strategy) actually apply repairing (i.e. modify the genotype of a solution) and two (waiting and penalizing) avoid it in some way in order to prevent drift-like effects in the search direction; see Figure 5.1 for a graphical illustration of how repairing can trigger a drift in the search direction.

In the description of the strategies we assume that multiple ERCs of a particular ERC type (commitment relaxation or periodic ERC) with non-overlapping (or non-contradictory) constraint schemata $H_i, i = 1, ..., r$, may be activated at a given time step. That is, there is always an evaluable solution, or $\exists \vec{x} \in \bigcup_i H_i$. We also assume that we know which resources are available and thus that the schemata H_i are known to the optimizer.

1. Forcing: This strategy simply *forces* a non-evaluable solution $\vec{x}$ into the constraint schemata H_i of all activated ERCs. In other words, all bits that do

not match the order-defining bit values of the schemata H_i of all activated ERCs are flipped, and the solution so obtained is returned for evaluation. We employed this strategy in the theoretical analysis presented in Section 4.3 (to force type B individuals into type A individuals during activation periods of a periodic ERC). Repairing strategies of this kind have been used previously, particularly, in the optimization of constrained combinatorial problems (see Section 3.3).

A potential drawback of this strategy is that enforcing a change in the values of some solution bits may destroy a potentially good combination of bit values between the changed and the remaining bits present in the original solution. Later, we will investigate this aspect more closely.

2. Regenerating: The aim of this strategy, which is similar to the *death penalty* approach (see Section 3.3) often used in the evolution strategies community, is to overcome the potential drawback of forcing. In fact, as the name of the strategy suggests, upon encountering a non-evaluable solution, *regenerating* iteratively generates new solutions from the empirical distribution of the current offspring population (i.e. it generates new offspring from the current parent set) until it generates one that is evaluable, i.e. falls into the schemata H_i of all activated ERCs, or until L trials have passed without success. In the latter case, we select the solution, generated within the L trials, that is *closest* to the schemata H_i of all activated ERCs and apply forcing to it. Here, *closest* refers to the solution with the smallest sum of Hamming distances to the schemata H_i of all activated ERCs;[1] ties between several equally-closest solutions are broken randomly. Thus the method always returns an evaluable solution (except in the 'deadlock' situation where multiple ERCs with overlapping H_i are activated simultaneously, in which case no solution is evaluable).

A potential drawback of this strategy is that for large L it can be computationally and/or resource expensive if checking of constraints is costly, while for small L, it could be that it reduces often to the forcing strategy.

3. Subpopulation strategy: Let us assume there is only one ERC, i.e. $r = 1$. In this case, alongside the actual population, we maintain also a *subpopulation* SP of maximum size J that contains the fittest solutions from H_1 evaluated so far. A non-evaluable solution is then dealt with by generating a new solution based on this subpopulation. If the maximum population size of SP, J, is not

[1] Notice that the Hamming distance between a solution $\vec{x}$ and a schema H is calculated based on the order-defining bits of H only (as the non-order-defining bits can be either 0 or 1-bits).

reached, then a new solution from H_1 is generated at random, otherwise we apply one selection and variation step using the same algorithm as the one we augment the constraint-handling strategies on; if the new solution is non-evaluable, which may happen due to mutation, we apply forcing to it. To update the subpopulation upon evaluating a solution from H_1 we use a steady state or $(J + 1)$-ES reproduction scheme. We use this reproduction scheme because, depending on the ERC, the number of evaluated solutions from H_1 might be small, in which case a generational reproduction scheme is likely to result in a slow convergence.

Generally, a drawback of the subpopulation strategy is that if we have more than one ERC, i.e. $r > 1$, then the number of subpopulations we need is upper-bounded by 2^r, the power set of the total number of ERCs. With multiple ERCs, we generate a solution using the subpopulation that is defined by the (set of) schemata H_i of all activated ERCs.

4. Waiting: This strategy does not repair but it *waits* with the evaluation of a non-evaluable solution and the generation of new solutions until the activation periods of all ERCs that are violated by the solution have passed; i.e. the optimization *freezes*. The freezing period is bridged by submitting as many null solutions as required until the solution becomes evaluable.[2] The effect of this waiting strategy is identical to the way Schwefel (1968) handled unavailable conical rings in his flashing nozzle design problem (see preface of Section 1).

The advantage and disadvantage of waiting is obvious: it should prevent drift-like effects in the search direction caused by ERCs, but the waiting might result in a smaller number of solutions being evaluated (this depends upon whether time is a limiting factor).

5. Penalizing: Like waiting, this strategy does not repair. However, instead of freezing the optimization, a non-evaluable solution is *penalized* by assigning a poor objective value c to it. The effect is that evaluated solutions coexist with non-evaluated ones in the same population. However, due to selection pressure in parental and environmental selection, non-evaluated solutions are likely to be discarded as time goes by. As we will use the strategy within an elitist EA, and

[2]We want to point out that in the implementation of this strategy we use here we do not submit null solutions to bridge the freezing period. Instead, we bridge the freezing period by setting the (global) time counter t directly to the end of the longest activation period of all currently violated ERCs. The effect of this is identical to submitting null solutions, but this approach allows for a cleaner presentation of the EA and the function wrapper in Algorithm 5.1; in this algorithm, the bridging of the freezing period is performed in Line 25.

because we use a c that is the minimal fitness in the search space, a non-evaluated solution will never be inserted in a population (that is filled with previously evaluated solutions) in the first place.

This kind of penalizing strategy has been used by Schwefel, and Reynolds and Corne to cope with execution errors occurring in simulations (we mentioned this application in the preface of Section 1). But the approach is also generally popular in the genetic algorithm (GA) community; here, this penalizing approach can be seen as a *static penalty function method* (see Section 3.3).

The advantage of penalizing over waiting is that the optimization does not freeze upon encountering a non-evaluable solution; i.e. the solution generation process continues and thus solutions might actually be evaluated (without needing to penalize them) during an activation period. However, since solutions evaluated during an activation period will have to fall into the schemata H_i of all currently activated ERCs, penalizing might be subject to drift-like effects, thus potentially losing the advantage of waiting.

Figure 5.2 visualizes how the strategies forcing, regenerating, and the subpopulation strategy may repair a non-evaluable solution.

5.1.2 Experimental setup

This section describes the test functions f, the EA on which we augment the different constraint-handling strategies, and the parameter settings as used in the subsequent experimental analysis, which investigates the impact of commitment relaxation and periodic ERCs.

5.1.2.1 Search algorithm

We augment the different static constraint-handling strategies on a standard EA. The EA uses a standard $(\mu + \lambda)$-ES reproduction scheme, an elitist approach, which we believe would be generally applicable in this domain. The algorithm uses binary tournament selection (with replacement) for parental selection, which has shown to be a robust operator in the theoretical study (see Section 4.3.3). The algorithm also uses uniform crossover (Syswerda, 1989) and bit flip mutation. Algorithm 5.1 shows the pseudocode of the EA including the function wrapper as employed with the different static constraint-handling strategies. Additional performance-enhancing mechanisms commonly used in EAs, such as diversity

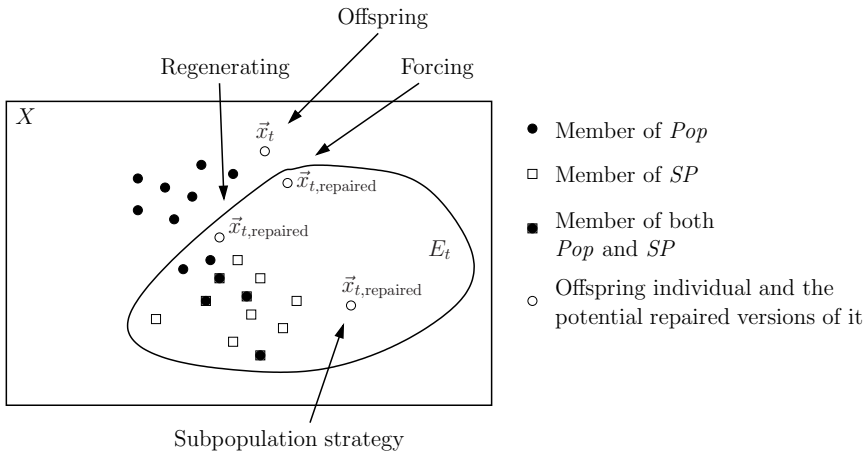

Figure 5.2: A depiction of the current population *Pop* (filled circles and squares) and an offspring individual $\vec{x}_t$, which is feasible but not evaluable (because it is in X but not in E_t). Solutions indicated by the filled squares coexist in both the actual EA population *Pop* and the population *SP* maintained by the subpopulation strategy. The three solutions $\vec{x}_{t,\text{repaired}}$ indicate repaired solutions that might have resulted after applying one of the three 'repairing' strategies to $\vec{x}_t$: while forcing simply flips incorrectly set bits of $\vec{x}_t$ and thus creates a repaired solution that is as close as possible to $\vec{x}_t$ but not necessarily fit, regenerating creates a new solution in E_t using the genetic material available in *Pop*. Similarly, the subpopulation strategy creates also a new solution but it uses the genetic material available in the subpopulation *SP* (empty and filled squares), which contains only solutions from E_t.

preservation techniques or adaptive parameter control (see Section 3.1.2), may affect the results, but are not considered in this particular study. Similarly, since we want to investigate the performance of a standard EA, the algorithm does not check whether a currently non-evaluable solution has been evaluated previously, i.e. identical solutions may be evaluated multiple times.

5.1.2.2 Test functions f

Since our aim in this study is to understand the effect of ERCs on EA performance on real closed-loop problems (ultimately), it might be considered ideal to use, for testing, some set of real-world ERCOPs, that is: real experimental problems featuring real resource constraints. That way we could see the effects of EA design choices (the constraint-handling strategy used) directly on a real-world problem of interest. But even granting this to be an ideal approach, it would be very difficult to achieve in practice due to the inherent cost of conducting closed-loop experiments and the difficulty of repeating them to obtain any statistical

Algorithm 5.1 Generational EA with static constraint-handling strategies

Require: ERC_1,...,ERC_r (set of ERCs), f (objective function), T (time limit), μ (parent population size), λ (offspring population size), #Strategy (number of selected constraint-handling strategy; see Section 5.1.1)

1: $t = 0$ (global variable representing the current time step), $Pop = \emptyset$ (current population), $OffPop = \emptyset$ (offspring population)
2: **while** $|Pop| < \mu \ \wedge \ t < T$ **do**
3: generate solution $\vec{x}$ at random
4: $\vec{x} = \text{functionWrapper}(\vec{x}, t)$
5: $Pop = Pop \cup \{\vec{x}\}$; t++
6: **while** $t < T$ **do**
7: $OffPop = \emptyset$
8: **repeat**
9: generate two offspring $\vec{x}^{(1)}$ and $\vec{x}^{(2)}$ by selecting two parents from Pop, and then recombining and mutating them
10: $OffPop = OffPop \cup \{\vec{x}^{(1)}\} \cup \{\vec{x}^{(2)}\}$
11: **until** $|OffPop| = \lambda$
12: **for** $i = 0$ to $|OffPop|$ **do**
13: **if** $t < T$ **then**
14: $\vec{x}_i = \text{functionWrapper}(\vec{x}_i, t)$ // $\vec{x}_i$ represents the ith solution of $OffPop$
15: t++
16: form new Pop by selecting the best μ solutions from the union population $Pop \cup OffPop$

17: functionWrapper$(\vec{x}, t)\{$
18: $y_t = \text{null}$
19: **if** $\vec{x}$ satisfies the ERCs ERC_1,...,ERC_r **then**
20: $\vec{x}_t = \vec{x}$; $y_t = f(\vec{x}_t)$
21: **else**
22: **if** #Strategy $= 1 \vee$ #Strategy $= 2 \vee$ #Strategy $= 3$ **then**
23: $\vec{x}_t = \text{repair}(\vec{x}, t)$; $y_t = f(\vec{x}_t)$
24: **if** #Strategy $= 4$ **then**
25: $t = t + \delta$; $\vec{x}_t = \vec{x}$ // δ is the number of time steps we have to wait until the activation periods of all ERCs that are currently violated by $\vec{x}$ have passed
26: **if** $t < T$ **then**
27: $y_t = f(\vec{x}_t)$
28: **if** #Strategy $= 5$ **then**
29: $\vec{x}_t = \vec{x}$; $y_t = c$ // c is a constant, representing poor fitness
30: return $\vec{x}_t$ and $y_t\}$

confidence in results seen. For this reason, most of our study will use more familiar *artificial* test problems augmented with ERCs (although in the case study, in Section 5.1.4, we do use data and constraints from a real closed-loop problem.)

The use of test functions to understand aspects of EA performance has long been a staple of EA empirical studies. It is a good approach generally, and appropriate also here, because problem features can be controlled and experiments can be repeated many times, allowing a thorough analysis of results. Our set of selected test functions comprises: (i) OneMax, (ii) a competing optima problem,

TwoMax, and (iii) a problem instance with many local optima, MAX-SAT. In the subsequent case study we will also see a variant of *NK* landscapes (*NK*α landscapes) being used to tune (offline) our method; we will introduce this test function here too. The use of these different landscape types should be sufficient to draw some tentative conclusions about the effects of ERCOPs generally, depending on results observed.

OneMax: The *OneMax* function is a bit counting function that is used frequently in the theoretical analysis of GAs (see e.g. Mühlenbein and Schlierkamp-Voosen (1993); He and Yao (2002)). For a binary solution vector $\vec{x} \in \{0,1\}^l$, it takes the sum over the bit values of all bit positions

$$\text{maximize} \quad f(\vec{x}) = \sum_{i=1}^{l} x_i$$

and has its optimum $f = l$ for the bit string consisting only of 1-bits. Although the OneMax problem features a unimodal landscape and so is supposed to be relatively easy to solve for an EA, it helps us to gain detailed insights into things like how ERCs affect the convergence speed and may cause premature convergence. As we will see in the experimental study, these insights form the foundation for understanding the impact of ERCs on more complex fitness landscapes.

TwoMax: The *TwoMax* function is a bimodal function that can be seen as the multimodal counterpart of OneMax. It has two local optimal solutions: solutions consisting only of 1-bits and 0-bits. In its original version, the slopes leading to these two solutions are symmetric. As the OneMax function, this function has been intensively studied in theoretical works on the analysis of GAs (see e.g. Pelikan and Goldberg (2000); Hoyweghen et al. (2002)). To make the TwoMax function more realistic and interesting, its symmetric property has often been broken (see e.g. Pelikan and Goldberg (2000); He and Yao (2002); Friedrich et al. (2008)), turning it into a function with a local and a global optimal solution. The two common modifications are to change either the steepness of either slope or to simply increase the fitness value of either optimal solution. We opt for the former, making the slope leading to the solution consisting only of 1-bits steeper. Hence, if #1s denotes the number of 1-bits in a solution vector $\vec{x}$ and $b > 1$ the factor by which the global optimal solution shall be fitter than the local optimal

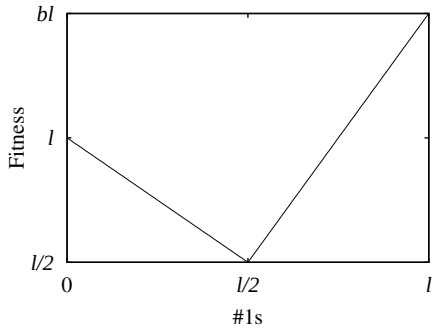

Figure 5.3: A TwoMax function with a local optimal solution at #1s = 0 and a global optimal solution at #1s = l. The parameter $b > 1$ specifies the factor by which the global optimal solution shall be fitter than the local optimal solution.

solution, then the TwoMax function is defined by

$$\text{maximize } f(\vec{x}) = \begin{cases} l - \#1s & \text{if } \#1\text{s} \leq \frac{l}{2}, \\ b\#1s & \text{otherwise.} \end{cases}$$

Figure 5.3 illustrates this TwoMax function. Our main reason for using an asymmetric version of the TwoMax function is to understand how much drift, coming from an ERC, is required to cause an optimizer that uses a specific strategy to climb up the suboptimal slope. Analyzing this property will help us to understand better the impact of ERCs on more challenging multimodal test problems, such as the MAX-SAT problem.

MAX-SAT: Given a boolean expression consisting of a set of clauses, which in turn are formed by binary variables $x_i, i = 1, ..., l$, the *maximum satisfiability* (MAX-SAT) problem (Zhang, 2001; Coppersmith et al., 2004) asks for the maximum number of clauses that can be satisfied by any assignment $\vec{x}$. A MAX-SAT problem is the optimization version of the decision problem, *satisfiability* (SAT), which only asks whether there is a satisfying assignment at all or not. MAX-SAT and SAT problems are of great theoretical interest, of course, due to SAT being the original problem proved NP-Complete by Cook, and still at the heart of attempts to determine the answer to the P vs NP problem. But it is of high practical relevance too, as many challenging real-world optimization problems can be efficiently formulated in SAT form (De Jong and Spears, 1989), e.g. hardware and software verification problems, and routing in FPGAs.

The instance considered is a uniform random 3-SAT problem and can be

downloaded online.[3] The instance has $l = 50$ variables and 218 clauses and is satisfiable. We treat this 3-SAT instance as a MAX-SAT optimization problem, with fitness calculated as the proportion of satisfied clauses.

Note, although there are local search algorithms for SAT or MAX-SAT problems, such as greedy SAT (Selman et al., 1992; Dechter, 2003), that can solve small problem instances easily, for optimizers that do not explicitly make use of the structure of the problem, such as standard EAs, the instances are generally hard to solve (due to the presence of many local optimal solutions) (Zhang, 2001; Prügel-Bennett, 2004; Qasem and Prügel-Bennett, 2010).

Apart from the relevance of MAX-SAT problems in theoretical and practical analysis, another reason for choosing this test problem is the convenient property of a *backbone* (the backbone of a MAX-SAT instance is the schema into which all the optimal solutions fall). We also conducted experiments on other challenging multimodal test problems, such as $NK\alpha$ landscapes (Kauffman, 1989), which we introduce next.

$NK\alpha$ landscapes: The general idea of the $NK\alpha$ landscapes (Hebbron et al., 2008) is to extend Kauffman's original NK model (Kauffman, 1989) to model epistatic network topologies that are more realistic in mapping the epistatic connectivity between genes in natural genomes. The $NK\alpha$ model achieves this by affecting the distribution of influences of genes in the network in terms of their connectivity, through a preferential attachment scheme. The model uses a parameter α to control the positive feedback in the preferential attachment so that larger α result in a more non-uniform distribution of gene connectivity. There are three tunable parameters involved in the generation of an $NK\alpha$ landscape: the total number of variables N (in our notation this variable is denoted as l), the number of variables that interact epistatically at each of the N loci, K, and the model parameter α that allows us to specify how influential some variables may be compared to others. Larger values of the parameter K yield more epistatic (rugged) landscapes, while larger values of α assign more influence (on the fitness) to a minority of variables;[4] for a value of $\alpha = 0$, the $NK\alpha$ model reduces to

[3] `http://people.cs.ubc.ca/~hoos/SATLIB/benchm.html`; the name of the instance is "uf50-218/uf50-01.cnf".

[4] We can think of the loci as nodes in a network with the number of edges going into a node and out of it representing the in-degree and out-degree of a node, respectively. Each node has the same in-degree of $K + 1$ (including the self-connection) but the out-degree approximates a power-law (with the parameter α being in the exponent) as a consequence of the preferential attachment scheme.

Table 5.1: EA parameter settings as used in the study of static constraint-handling strategies.

Parameter	*Setting*
Parent population size μ	50
Offspring population size λ	50
Per-bit mutation probability	$1/l$
Crossover probability	0.7

Table 5.2: Parameter settings of static constraint-handling strategies.

Strategy	*Parameter*	*Setting*
Regenerating	Number of regeneration trials L	10000
Penalizing	Fitness c assigned to non-evaluable solutions	0
Subpopulation strategy	Maximal size of subpopulation SP, J	30

Kauffman's original *NK* model with neighbors being selected at random. The *NK* model has already been used previously to analyze certain aspects of real-world closed-loop problems (see e.g. Thompson (1996b, 1997)).

5.1.2.3 Parameter settings

The parameter settings of the EA and the strategies are given in Table 5.1 and 5.2, respectively. The settings of the test functions are outlined in Table 5.3. We choose these search space sizes l as they correspond to typical search space sizes we have seen (e.g. a drug combinations problem with a library of about 30 drugs (Small et al., 2011)). The reason for setting the scaling factor b of the TwoMax function so small is that it makes the problem more challenging due to the low selection pressure to climb up the optimal slope. Regarding the optimization time T, remember that one time step corresponds to a function evaluation of a single solution or a single submitted null solution. In an ERC-free search, the optimization times as specified in Table 5.3 are usually not sufficient for an optimizer to locate a global optimal solution reliably. The reason we set T this way is that it gives us the opportunity to assess both positive and negative effects

Table 5.3: Parameter settings of test functions f as used in the study of static constraint-handling strategies.

Test function	*Parameter*	*Setting*
OneMax	Solution parameters l	30
	Optimization time T	700
TwoMax	Solution parameters l	30
	Scaling factor b	1.1
	Optimization time T	700
MAX-SAT	Solution parameters l	50
	Optimization time T	800
$NK\alpha$ landscapes	Solution parameters $N = l$	15
	Neighbors K	$\{2, 6\}$
	Model parameter α	$\{0, 2\}$
	Optimization time T	2250

of an ERC on the convergence speed and the solution quality obtained at the end of an algorithm run. In addition to the parameters above, we analyze the impact of the preparation and recovery time by considering different settings for T than specified in Table 5.3, but this will be pointed out where applicable.

Any results shown are average results across 500 independent algorithm runs. To allow for paired testing of the strategies, we use a different seed for the random number generator for each EA run but the same seeds for all strategies.

5.1.3 Experimental results

The performance of a strategy depends *inter alia* on the potential impact of an ERC on the population diversity and the optimization direction. To assess the impact on these two factors one has to consider: (i) what genetic material represented by a constraint schema H needs to be introduced into a population to cause a performance impact, (ii) how much of it, or, rather, how many individuals of a constraint schema need to be introduced into a population to cause a performance impact, (iii) at what stage during a run does it need to be introduced to yield a performance impact, and (iv) the effects of the preparation and recovery durations. We give detailed observations on these effects here, and summarise the key findings in Section 5.1.5.

5.1.3.1 Commitment relaxation ERCs

We first analyze the case where a constraint schema H represents poor genetic material. For OneMax and TwoMax, this means that the order-defining bits of H are set to 0. For the MAX-SAT instance, we represent a poor bit by flipping a randomly selected bit from the backbone of our instance[5]; for ease of presentation, also on this problem, we will write 1-bits to refer to 'good' bits, which are randomly selected unflipped bits from the backbone, and 0-bits to refer to their complements.[6]

Figure 5.4 gives an impression of how the final average best solution fitness is affected for the different strategies on OneMax. The results obtained on TwoMax and the MAX-SAT instance are very similar and are shown in Appendix A.1. For ease of presentation we normalize the fitness values of all test functions so that they lie in the range $[0, 1]$. We make the following observations from the figure.

- Generally, the ERCs affect search negatively, and strategy choice is important.

- The subpopulation strategy tends to perform better than forcing and regenerating (which perform similarly) for the majority of constraint parameters. The reason is that the subpopulation strategy generates fitter solutions from H and thus allows the EA to converge more quickly to a (suboptimal) population state containing many (copies of) optimal solutions from H. The subpopulation performs poorly for constraint settings that can cause a premature convergence towards search regions covered by H (see range $4 \leq o(H) \leq 8$ in the top left plot).

- With respect to the *order of the constraint schema* $o(H)$ (top left plot), we observe that there is a value that has the largest negative effect on the optimization; it is around 4 for the 'repairing' strategies (forcing, regenerating or the subpopulation strategy), and lower for penalizing and waiting.

[5]We identified the backbone of our MAX-SAT instance from optimal solutions obtained from running a generational GA for 1000 generations, 500 times independently. The backbone is of order 44.

[6]If not otherwise stated, then the order-defining bits of a constraint schema are chosen at random for each algorithm run; if an order-defining bit is a 1-bit, then the position of this bit is also chosen at random among the order-defining bits; i.e. a constraint schema denoted by $H = (010 * * * *)$ might actually be e.g. $H = (* * 1 * 0 * 0)$ or $H = (* * 00 * *1)$ in an algorithm run. Nevertheless, all strategies will be optimizing subject to the same constraint schemata.

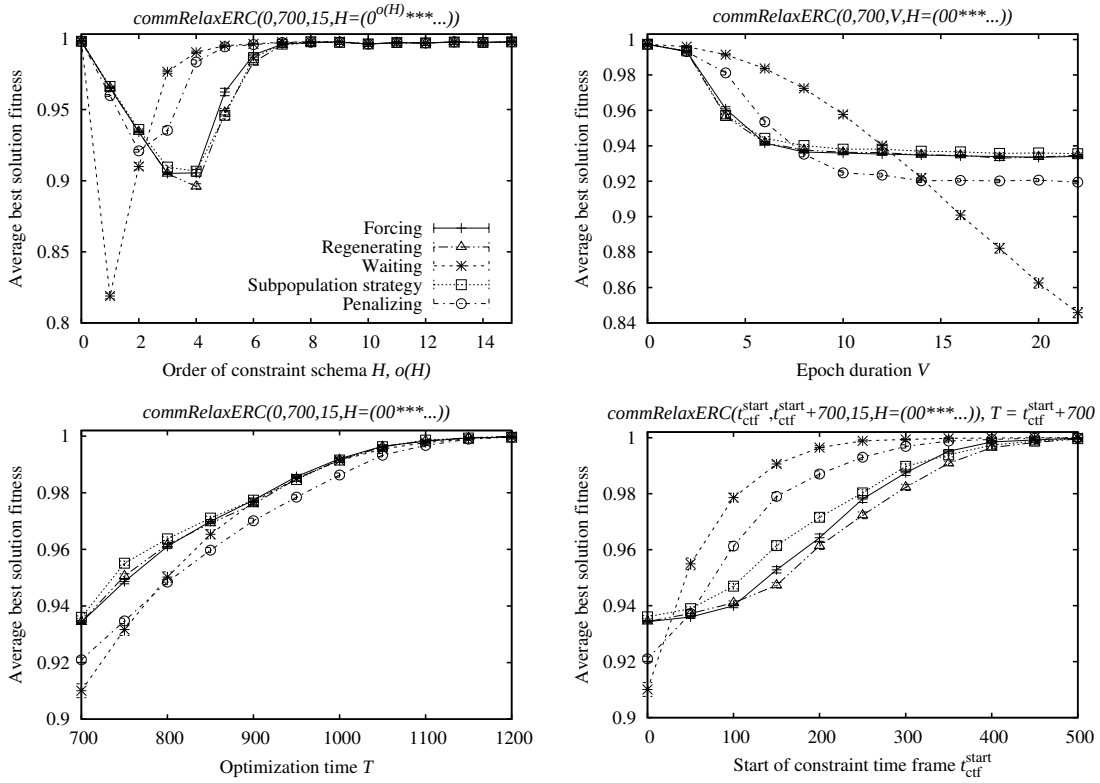

Figure 5.4: Plots showing the average best solution fitness found and its standard error on OneMax as a function of the order of the constraint schema $o(H)$ (top left), the epoch duration V (top right), the optimization time T (bottom left), and the start of the constraint time frame $t_{\text{ctf}}^{\text{start}}$ (bottom right). Note, while the optimization time in the top plots is fixed to $T = 700$, as specified in Table 5.3, the parameter T varies in the bottom plots.

The non-monotonic performance impact on the repairing strategies, and partially on penalizing, is due to two competing forces: (i) the probability of activating a constraint, which decreases exponentially with $o(H)$, and (ii) the probability that a constraint activation causes a shift in the search focus, which is greater for low orders. With respect to these two forces, an order of $o(H) \approx 4$ tends to have the worst trade-off. Penalizing performs better than the repairing strategies in the range $2 < o(H) < 8$ because the probability of having to penalize solutions increases exponentially with $o(H)$. We remark that (results not shown) the order for which the worst trade-off is obtained is a function of the string length l and the population size μ. In general, the worst trade-off shifts to only a slightly higher order than 4 as l and/or μ increase; the shift is only little because the probability of activating the ERC decreases exponentially with the order.

For waiting, the performance only depends on the probability of activating a constraint, causing the performance to be poorest at $o(H) = 1$ and improve exponentially thereafter.

- Longer *epoch durations* (larger V) degrade performance of all methods because of potentially longer activation periods during epochs (see top right plot). With penalizing, forcing, regenerating, and the subpopulation strategy a saturation point is reached beyond which further increases in the epoch duration have no effect. With waiting there is no saturation point because an increase in V results in longer waiting periods and thus a poorer performance. The reason that waiting performs best for small V (see range $0 < V \leq 14$) is that the waiting periods are short in this regime, allowing an optimizer to converge quickly away from search regions covered by H and prevent future constraint activations.

- When providing some *recovery time*, all strategies improve in performance (see bottom left plot). The recovery speed depends on how much time is required to introduce first diversity among the previously constrained bits before one can generate better solutions.

- With later *start times* of the constraint time frame, or, equivalently, longer preparation times, there is a positive effect on the performance of all strategies (see bottom right plot). This is because with a commitment relaxation ERC the later in the optimization one is, the less likely it is to enter a poor schema (a schema not on the optimization path) and activate a constraint; also, due to elitism, repaired solutions are less likely to be inserted into the population the later a constraint is activated. Thus later constraints are less disruptive.

Let us now analyze how constraint schemata that represent genetic material of different qualities affect the performance obtained with the constraint-handling strategies. Figure 5.5 shows the effect of different constraint schemata on the average best solution fitness (left plots) and the number of times the optimal solution has been evaluated (right plots) for forcing (top plots) and waiting (bottom plots) on OneMax. The plots on the left-hand side indicate the effect of an ERC on the convergence speed towards an optimal solution.

Although ERCs can have large effects on performance, one can see from the plots of Figure 5.5 that the majority of the constraint schemata do not have an

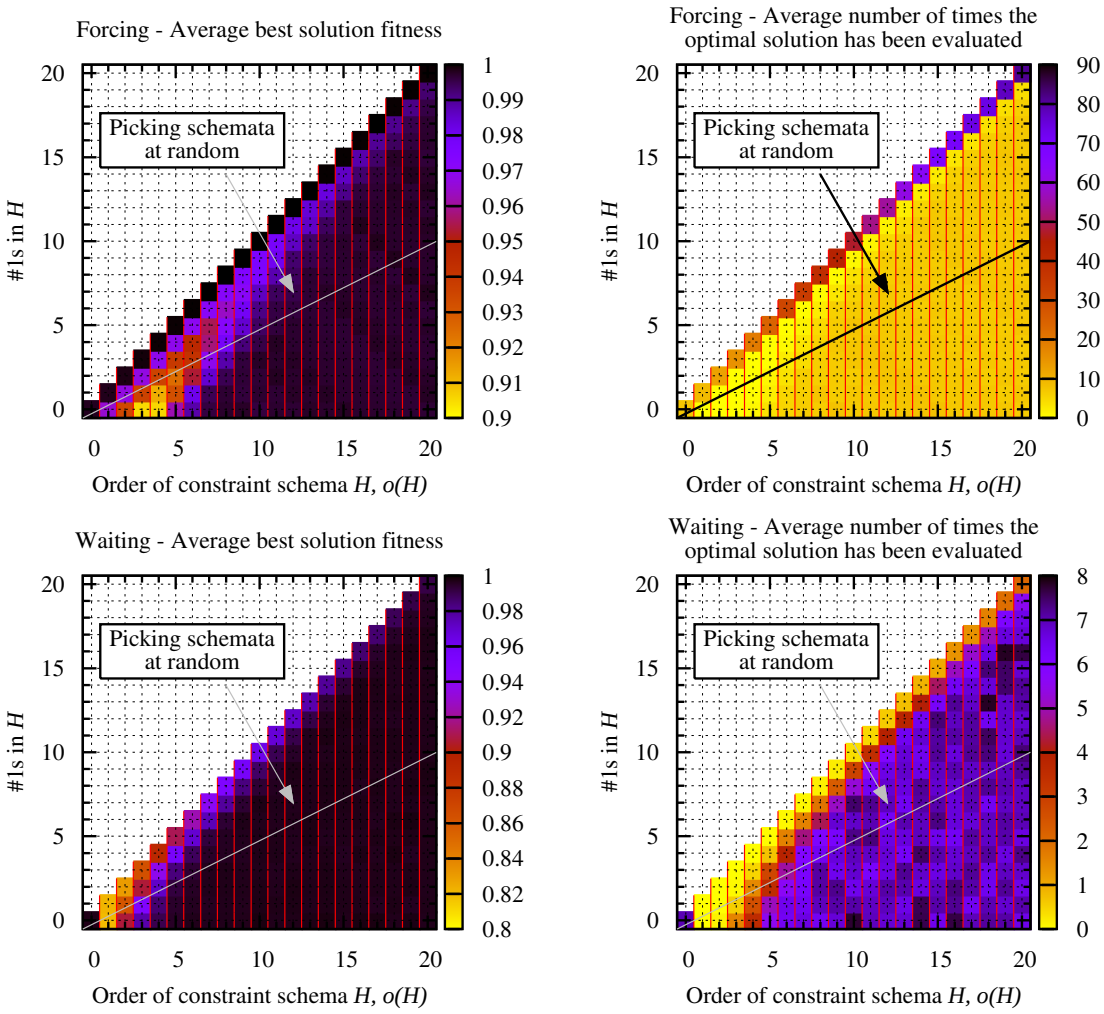

Figure 5.5: Plots showing the average best solution fitness obtained (left) and the average number of times the optimal solutions has been evaluated during the optimization (which is an indication of the convergence speed) (right) by forcing (top) and waiting (bottom) on OneMax as a function of the order of the constraint schema $o(H)$, and the number of order-defining bits in H with value 1 for the ERC $commRelaxERC(0, 700, 15, H)$. The straight line represents the expected performance when picking a schema (i.e. the order-defining bits and their values) with a particular order at random. The performance obtained in an unconstrained environment is represented by the square at $o(H) = \#1\text{s} = 0$.

impact on the performance at all compared to the unconstrained performance (which is represented by the square at $o(H) = \#1\text{s} = 0$). These are schemata that are unlikely to cause an activation at all because they either do not lie on an optimizer's search path (schemata with few 1-bits) or are associated with a generally low probability of being met by any individual (higher order schemata

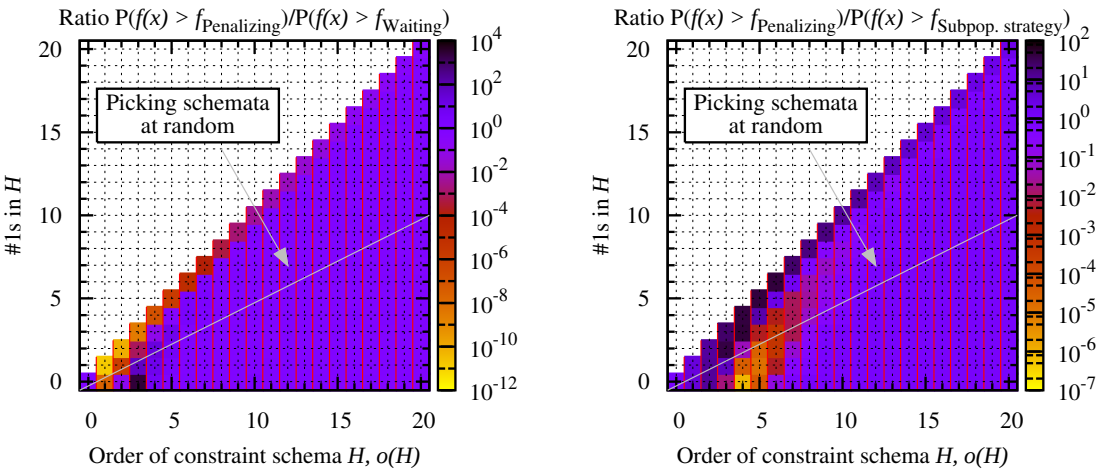

Figure 5.6: Plots showing the ratio $\mathrm{P}(f(x) > f_{\text{Penalizing}})/\mathrm{P}(f(x) > f_{\text{Waiting}})$ (left) and $\mathrm{P}(f(x) > f_{\text{Penalizing}})/\mathrm{P}(f(x) > f_{\text{Subpop. strategy}})$ (right) on OneMax as a function of the order of the constraint schema $o(H)$, and the number of order-defining bits in H with value 1 for the ERC $commRelaxERC(0, 700, 15, H)$; here, x is a random variable that represents the best solution from a set of solutions drawn uniformly at random from the search space and f_* the average best solution fitness obtained with strategy $*$. If $\mathrm{P}(f(x) > f_*)/\mathrm{P}(f(x) > f_{**}) > 1$, then strategy $**$ is able to achieve a higher average best solution fitness than strategy $*$ and a greater advantage of $**$ is indicated by a darker shading in the heat maps; similarly, if $\mathrm{P}(f(x) > f_*)/\mathrm{P}(f(x) > f_{**}) < 1$, then $*$ is better than $**$ and a lighter shading indicates a greater advantage of $*$.

around the straight line). Constraint schemata that represent poor genetic material (i.e. consist of many 0-bits) have only an impact if their order is low because an optimizer is searching in a different direction. Hence, constraint schemata that have a significant effect on the performance of both strategies are either of low order or contain many 1-bits (schemata along and near the diagonal). For these constraint setting regimes, we observe a similar non-monotonic effect on the performance of both strategies as we have seen previously for schemata representing poor genetic material only (indicated here by the row of squares with $\#1\text{s} = 0$). The difference is that the more 1-bits there are in H (i.e. as we go up the rows of squares), the less apparent becomes this non-monotonic (negative) performance effect. From Figure 5.6 it is apparent that the performance differences between the different strategies observed previously is also maintained largely (as we go up the rows of squares).

On the MAX-SAT instance, the range of constraint schemata causing a performance impact is smaller (see Figure 5.7). From the plot it is apparent that while again low-order schemata affect the performance significantly, higher-order

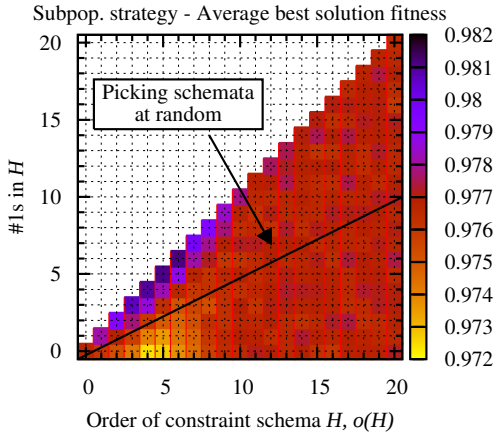

Figure 5.7: A plot showing the average best solution fitness obtained by the subpopulation strategy on a MAX-SAT problem instance as a function of the order of the constraint schema $o(H)$, and the number of order-defining bits in H set correctly for the ERC $commRelaxERC(0, 800, 15, H)$. The straight line represents the expected performance when picking a schema (i.e. the order-defining bits and their values) with a particular order at random.

schemata that represent near-optimal or optimal genetic material have only a little or no effect; the reason is that good genetic material is difficult to detect on this challenging problem, particularly within 800 time steps.

5.1.3.2 Periodic ERCs

With the insights we gained about the strategies when applying them to commitment relaxation ERCs, we can understand their behavior in the presence of a second type of ERC, periodic ERCs, more easily.

Figure 5.8 shows how the performance of the different strategies is affected by various constraint parameters of a periodic ERC on OneMax when H represents poor genetic material. Again, the results obtained on TwoMax and the MAX-SAT instance are similar and are shown in Appendix A.2. In comparison to commitment relaxation ERCs, the main difference we observe from Figure 5.8 is that waiting performs poorly and is also clearly dominated by penalizing for the majority of constraint settings. This is due to the fact that activation periods are set deterministically with periodic ERCs. In essence, waiting is likely to freeze the optimization during each activation period because of the low probability of generating solutions from H (regardless of the quality of the genetic material represented by H). With penalizing one is also unlikely to evaluate any solutions during an activation period. However, the fact that the optimization is

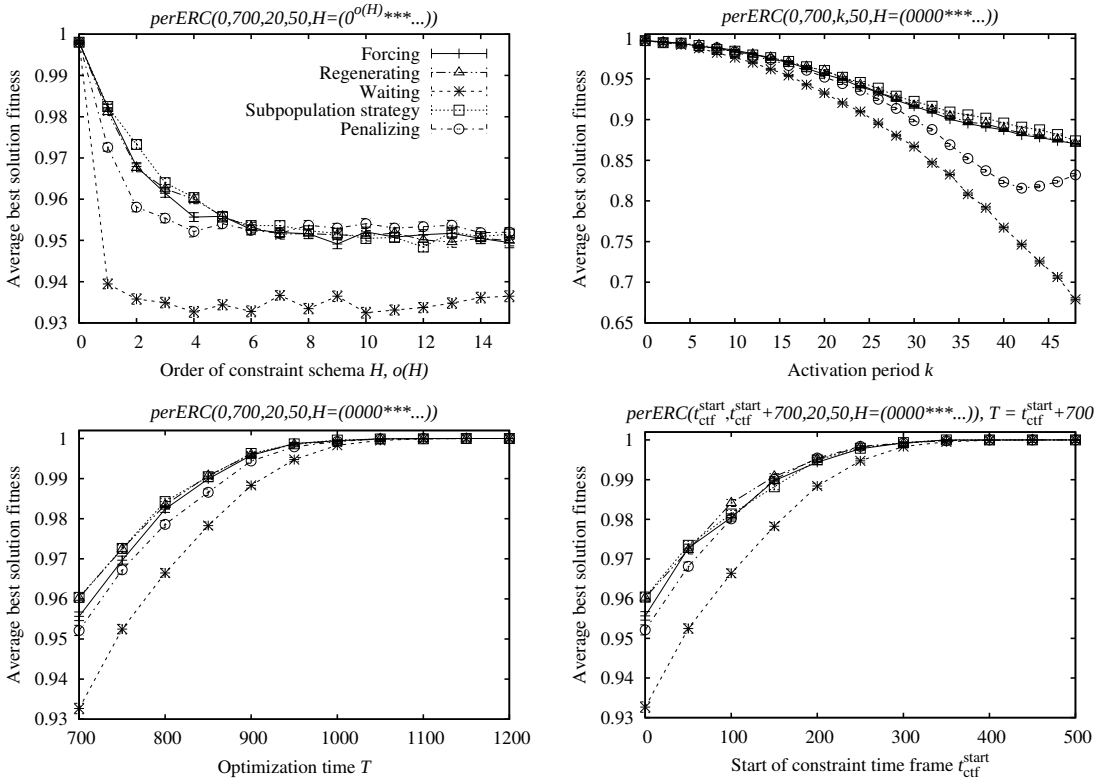

Figure 5.8: Plots showing the average best solution fitness found and its standard error on OneMax as a function of the order of the constraint schema $o(H)$ (top left), the activation period k (top right), the optimization time T (bottom left), and the start of the constraint time frame $t_{\text{ctf}}^{\text{start}}$ (bottom right).

not frozen is beneficial because offspring are generated using a more up-to-date parent population during unconstrained optimization periods.

The fact that activation periods are set deterministically with periodic ERCs means also that high-order constraint schemata have an impact on the performance and this is the case regardless of the genetic material they represent. In fact, from Figure 5.9 we see that the average best solution fitness obtained with the subpopulation strategy on OneMax decreases rather smoothly for all orders as the quality of the represented genetic material worsens. Comparing this average best solution fitness with the fitness obtained by penalizing (left plot of Figure 5.10), we observe that repairing is particularly beneficial for high-order constraint schemata that represent very good genetic material. Waiting, in turn, is inferior to penalizing across all the different constraint schemata but in particular for low-order schemata representing very good genetic material (see right plot of Figure 5.10).

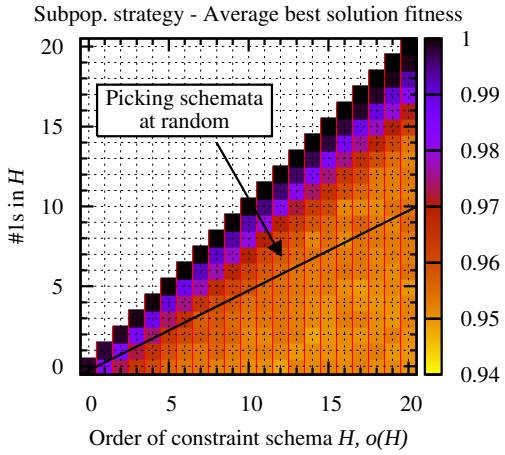

Figure 5.9: A plot showing the average best solution fitness obtained by the subpopulation strategy on OneMax as a function of the order of the constraint schema $o(H)$, and the number of order-defining bits in H with value 1 for the ERC $perERC(0, 700, 20, 50, H)$. The straight line represents the expected performance when picking a schema (i.e. the order-defining bits and their values) with a particular order at random.

On the MAX-SAT instance, results not shown here, one makes similar observations as on OneMax. However, because of the difficulty of finding good genetic material, even when setting a small number of bits correctly, a smooth decrease in the average best solution fitness, and differences between the performance of strategies, are more obvious for schemata of higher orders. Compared to OneMax, there are also small differences apparent in the results obtained on TwoMax; we show the results and discuss these differences in Appendix A.2.

5.1.4 Case study

In this section we demonstrate one way in which an appropriate constraint-handling strategy for an ERCOP may be selected in a real-world application. For this we use the same experimental setup as used in the instrument configuration application of O'Hagan et al. (2005, 2007). We mentioned this application in our review of closed-loop problems in Section 2.2, but give a more detailed description of the problem here.

Application description: The application is concerned with detecting as many metabolites in a biological sample as possible by optimizing the configuration of the instrument the samples are run through. Methods currently in use to analyze

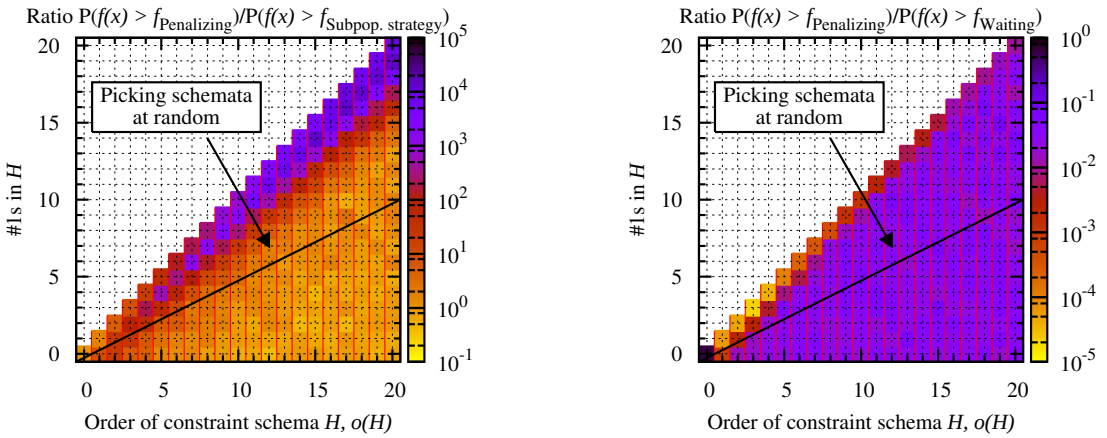

Figure 5.10: Plots showing the ratio $\mathrm{P}(f(x) > f_{\text{Penalizing}})/\mathrm{P}(f(x) > f_{\text{Subpop. strategy}})$ (left) and $\mathrm{P}(f(x) > f_{\text{Penalizing}})/\mathrm{P}(f(x) > f_{\text{Waiting}})$ (right) on OneMax as a function of the order of the constraint schema $o(H)$, and the number of order-defining bits in H with value 1 for the ERC $perERC(0, 700, 20, 50, H)$; here, x is a random variable that represents the best solution from a set of solutions drawn uniformly at random from the search space and f_* the average best solution fitness obtained with strategy $*$. If $\mathrm{P}(f(x) > f_*)/\mathrm{P}(f(x) > f_{**}) > 1$, then strategy $**$ is able to achieve a higher average best solution fitness than strategy $*$ and a greater advantage of $**$ is indicated by a darker shading in the heat maps; similarly, if $\mathrm{P}(f(x) > f_*)/\mathrm{P}(f(x) > f_{**}) < 1$, then $*$ is better than $**$ and a lighter shading indicates a greater advantage of $*$.

samples are ones that employ a separation step coupled to mass spectrometric detection. As the separation method, gas chromatography is considered as a highly resolving technique and the method of choice. The challenge with this approach, however, is that many instrumental parameters are involved in the process and that small changes in these parameters may have significant effects on the chromatographic performance. Variables or instrumental parameters represent things like the sample volume, split ratio, oven temperatures, ramp rates, and other settings of a gas chromatograph; in this application, we have $l = 15$ integer variables or instrument parameters making up a search space of 7.32×10^9 different instrument configurations. The fitness of a configuration is determined by running a biological sample through it, and measuring properties of the instrument's output signal, such as the number of peaks (or metabolites) detected in the chromatogram, the signal-to-noise ratio of the peaks, and the run time. O'Hagan et al. (2005, 2007) cast this application as a multi-objective optimization problem but here we consider only one objective, namely the number of peaks detected.

In general, when dealing with complex and sensitive instruments, such as mass spectrometer/gas chromatography equipment, the stability of the instrument and

the results can be affected when changing instrument parameters across wide ranges. In particular, if the oven temperature of the instrument is set low in one experiment, then it is expected that some metabolites remain stuck to the column (a tube in which the sample is passed through solvent in order to separate the elements within the sample) and may elute from it in a later experiment where the temperature is set higher *unless the column is cleaned or replaced between experiments* (Dunn, March 2011). To avoid this, the column should ideally be cleaned or replaced whenever an instrument configuration with a low oven temperature is tested. However, usually, cleaning or replacing of instrument parts is done overnight because it incurs additional costs in time and money when done after every analysis run. Hence, using a low oven temperature enforces a temporary commitment to this temperature for the rest of the day, or, equivalently, activates a commitment relaxation ERC; as the application of O'Hagan et al. (2005, 2007) makes use of two ovens, the optimization is subject to two commitment relaxation ERCs. In the particular application of O'Hagan et al. (2005, 2007), however, the experimentalists avoided the ERCs by setting a smaller parameter range of the oven temperature than would have been ideal to optimize this instrument parameter (Dunn, March 2011). We want to investigate how the performance might have been affected in the presence of the two ERCs.

ERCs: To keep things simple, we assume that the maximal number of instrument configurations that can be tested on a day is fixed at $V = 15$, and the total number of days available for the optimization is 150, resulting in $T = 15 \times 150 = 2250$ available time steps or fitness evaluations. We set the first two variables to represent the oven temperatures, and the value 0 to represent the low temperatures. Hence, we have the following two commitment relaxation ERCs: $commRelaxERC(0, 2250, 15, H = (0 * * * ...))$ and $commRelaxERC(0, 2250, 15, H = (*0 * * * ...))$. Little is known of the fitness landscape before optimization begins but, as in (O'Hagan et al., 2007), it would be expected that there is some degree of epistasis in the problem. We would not know which of the two schemata represent good or poor instrument configurations.

Offline testing: As algorithm designers, we are now faced with the challenge to select an optimization algorithm or constraint-handling strategy for the above described ERCOP. The common approach is to first design appropriate problem functions that simulate the problem at hand, and then to test several algorithms offline on these functions and use the best one for the real-world problem. In this

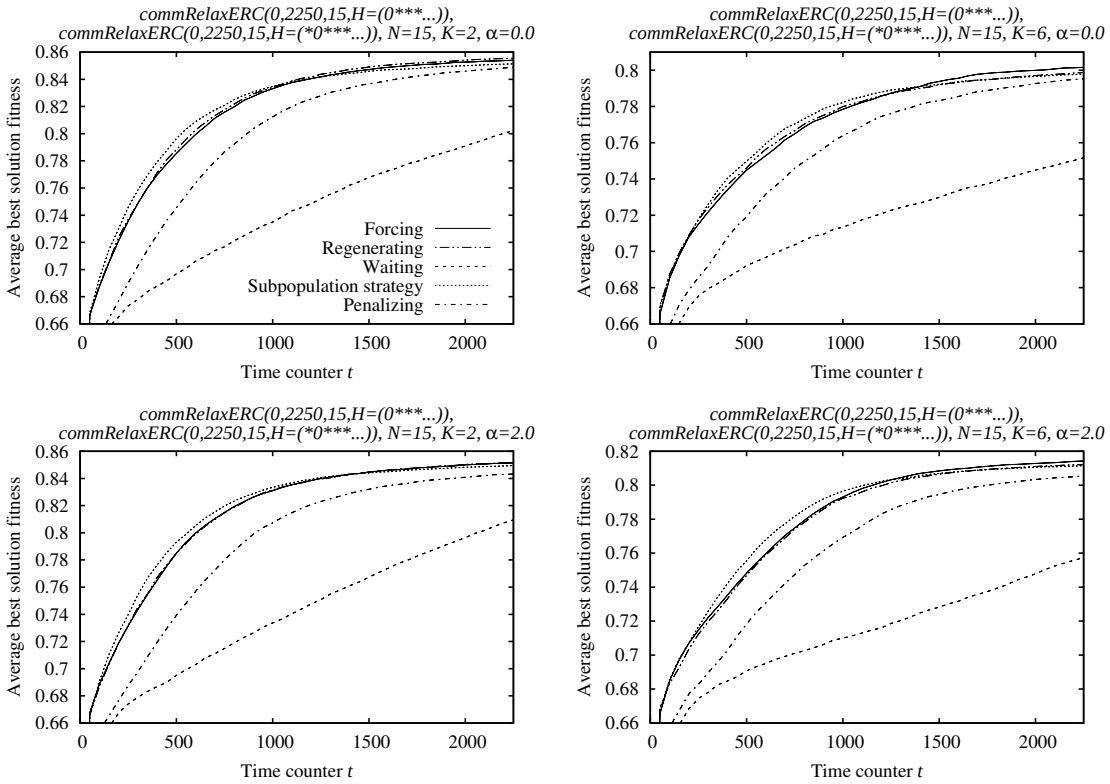

Figure 5.11: Plots showing the average best solution fitness obtained on $NK\alpha$ landscapes with $N = 15$ and $K = 2, \alpha = 0.0$ (top left), $K = 6, \alpha = 0.0$ (top right), $K = 2, \alpha = 2.0$ (bottom left), and $K = 6, \alpha = 2.0$ (bottom right) as a function of the time counter t; results are averaged over 500 independent runs using a different randomly generated problem instance for each run. All instances were subject to the two commitment relaxation ERCs $commRelaxERC(0, 2250, 15, H = (0 * * * ...))$ and $commRelaxERC(0, 2250, 15, H = (*0 * * * ...))$.

case study, we use $NK\alpha$ landscapes as the test problems because they allow us to model different degrees of epistasis. We introduced this problem in Section 5.1.2.2, and also provided the settings of the problem parameters N, K, and α in Table 5.3. The (four) selected settings will give us some insights into how the landscape topology and the degree of epistasis may influence the performance. To cope with the integer representation, we need to modify the mutation operator, which shall now select a random setting from the set of possible ones for the instrument parameter that is to be modified. Otherwise, we can use the same algorithm setup as previously (see Algorithm 5.1).

Let us now analyze how the different constraint-handling strategies perform on the four $NK\alpha$ landscapes. Figure 5.11 shows the average best solution fitness obtained by the different strategies on the landscape models as a function of the

time counter (we do not show the standard error as it was negligible). The plots confirm what we observed in the experimental study that similar patterns are obtained for different landscapes. In fact, we observe that a repairing strategy (forcing, regenerating, or the subpopulation strategy) should be clearly favoured over a waiting or penalizing strategy. More precisely, a trend is apparent that the subpopulation strategy and regenerating perform best in the initial stages of the optimization, while forcing is slightly better in the final part of the optimization. Also, the performance advantage of forcing at $T = 2250$ over the other two repairing strategies tends to increase with K and/or α. The waiting strategy does not perform well because the likelihood that either or both of the ERCs is active is relatively high, causing the optimization to freeze for too long. Although penalizing performs significantly better than waiting, the probability of penalizing many solutions and thus making only little or no progress in the optimization is too high to match the performance of the repairing strategies.

Based on these results, if a single strategy is to be chosen, then we would select the strategy, forcing, for the real-world instrument optimization problem as it performs best after 2250 time steps.

Testing on real-world landscape: To test this choice now on the real problem, we cannot run it on the real closed-loop problem (and certainly we would not be able to compare different policies). However, we are able to do the next best thing. Since we have available the actual fitness values collected during the real experimental trials reported by O'Hagan et al. (2005, 2007) (i.e. the number of peaks detected) for around 315 instrument configurations tested, we can use this data to construct an interpolated fitness landscape using, for example, the Kriging approach (Cressie, 1993).[7] In comparison to the $NK\alpha$ landscapes considered for offline testing, the interpolated landscape is smoother and contains significantly fewer local optima;[8] both aspects are attributed to the low number of data points.

Nevertheless, as it is apparent from Figure 5.12, the performance of the strategies on the interpolated landscapes is largely in alignment with the findings made on the $NK\alpha$ landscapes. In particular, the repairing strategies tend to perform

[7]In essence, Kriging is a technique that interpolates the fitness value of an unobserved data point from observations of values of nearby data points. To generate the fitness landscape we used a Kriging function Krig() from the *fields* package of the statistical software, R.

[8]To obtain this information, we followed the experimental methods employing *adaptive walks* already used by Hebbron et al. (2008).

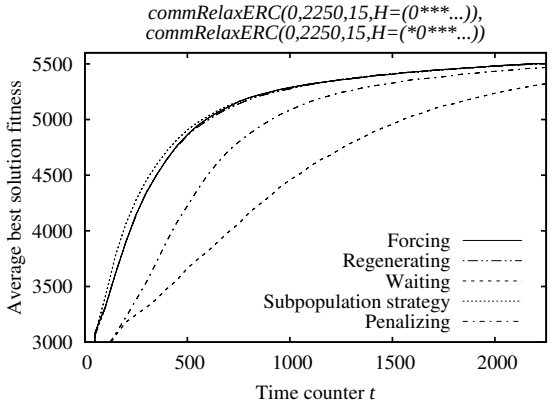

Figure 5.12: A plot showing the average best solution fitness obtained on a fitness landscape interpolated from real-world data as a function of the time counter t. The optimization was subject to the two commitment relaxation ERCs $commRelaxERC(0, 2250, 15, H = (0 * * * ...))$ and $commRelaxERC(0, 2250, 15, H = (*0 * * * ...))$.

better than waiting and penalizing, and, while the subpopulation strategy performs best at the beginning of the optimization, all repairing strategies tend to perform identically at the end of the optimization run. Clearly, in reality one is usually able to perform only a single optimization run meaning that the result might be different from the one we obtained from averaging over many runs. Nevertheless, this case study demonstrates how one can approach and solve an ERCOP beginning with the definition of the ERCs, modelling the simulated environment, selection of appropriate test functions, and finally comparing different optimizers and selecting the most suitable one to be used in the real-world application.

5.1.5 Summary and conclusion

In this study, we have proposed and analyzed various static constraint-handling strategies for dealing with non-evaluable solutions arising in ERCOPs. We considered two types of ERCs, commitment relaxation ERCs and periodic ERCs, and four test functions, OneMax, TwoMax, a MAX-SAT problem instance, and $NK\alpha$ landscapes. In addition, a demonstration of how one may approach and solve a new ERCOP in the common case where knowledge of the fitness landscape is poor has been given in the form of a case study (see Section 5.1.4).

We made several key observations from the experimental analysis that may determine how one should proceed with an ERCOP. Generally, ERCs affect the

performance of an optimizer, and clear patterns emerge relating ERC parameters to performance effects. The later the constraint time frame of an ERC begins, the less disruptive is the impact on search. This could mean that investment in resources at early stages of the optimization should be preferred, where possible. For commitment relaxation constraints, the probability of activating the constraint is dependent on the order and the quality of the genetic material represented by the constraint schemata. Thus, to some degree, we may be able to predict the extent of impact if information about these schemata is available. Although periodic constraints are activated at regular time intervals (independently of the constraint schemata defining them), their impact on search still depends upon the order and quality of the constraint schemata in predictable ways. We also clearly see that the impact on EA performance is modulated by the choice of constraint-handling strategy adopted. Which choice of strategy is best is dependent on the details of the ERC, as we have set out in our results.

Importantly, we also observed that the patterns of performance impact seen on the same ERC type are quite similar *across different search problems with different types of fitness landscape*. Whereas, between the two ERCs, even on the same problem, the impact on performance is quite different. If this pattern turns out to be more generally true, then it is good news because we usually have more knowledge about the ERCs than about the fitness landscape. Therefore we would not need to be 'right' about the fitness landscape in order to choose the right strategy. Nevertheless, as indicated in the case study, an *a priori* analysis of the problem at hand can be beneficial when it comes to selecting a suitable strategy.

With respect to the impact of the individual ERC types, our analysis concluded that with commitment relaxation ERCs, we would tentatively say that repairing strategies should not be used (i.e. the genotype of a solution should not be modified) if non-evaluable solutions are unlikely to be encountered, resources are non-available for only short periods, or much recovery or optimization time is available. With periodic ERCs, resources become non-available at regular time intervals. Because of this, solutions are likely to be non-evaluable during activation periods, which seems to cause strategies that do not repair to perform quite poorly. An exception may be the situation where the available resources are poor because, there, repairing solutions and inserting them into the population may cause an EA to prematurely convergence to a suboptimal population state.

For both ERC types, in situations where it should be repaired, we can tentatively suggest that a strategy that aims at creating fit repaired solutions should be preferred over naïve forcing strategy. An exception may be again the case where the available resources are poor. In the latter situation, the fitter a repaired solution is, the more likely it is to be inserted into the population and thus cause further constraint activations. Ultimately, this may increase the probability of prematurely converging to a suboptimal population state.

Although we are able to draw these conclusions, our study has of course been very limited, and there remains much else to learn about the effects of ERCs and how to handle them. The next section takes a step further and investigates learning-based strategies for switching between the static constraint-handling strategies introduced here during an optimization process.

5.2 Learning-based constraint-handling strategies

The previous section provided evidence that it is possible to select a suitable (static) constraint-handling strategy for an ERCOP offline if the ERCs are known in advance. Inspired by this observation, this section investigates whether an EA that learns offline when to switch between the static constraint-handling strategies during the optimization process is more efficient than the static strategies themselves. Offline learning is performed by a reinforcement learning (RL) agent, and this agent aims at optimizing the average fitness of the final population (reward) by selecting the most suitable static strategy (which serve as the actions) in a given state during the optimization. The next section introduces this offline learning strategy in more detail. We also consider an EA that learns the same switching task as the RL agent but online using a multi-armed bandit algorithm; Section 5.2.2 will introduce this approach. In Section 5.2.3 we experimentally compare the two learning-based strategies against the static constraint-handling strategies themselves for commitment relaxation ERCs. Section 5.2.4 draws together the findings from the experimental analyses.

5.2.1 Offline learning-based strategy

To learn offline when to switch between static constraint-handling strategies during an optimization run we use the tabular RL algorithm, Sarsa(λ) (see Algorithm 3.4 in Section 3.6). We employ Sarsa(λ) in combination with replacing eligibility traces (see Equation (3.7)), and the ϵ-greedy action selection method. Our learning task is episodic with each episode representing a single EA run. We reset the eligibility trace to $e = 0$ at the beginning of each episode, meaning we forget about the state-action pairs visited in the previous episode (see Line 11 of Algorithm 5.2). This modification to Algorithm 3.4 resulted in a slightly better performance in our case. To encourage exploration we use optimistic initial values for the action-value estimates $Q(s, a)$; we set all values to $Q(s, a) = 1.0, \forall s \in S, a \in A(s)$ (1.0 is the maximal expected return in our case).

To characterize a state s we use the *current population average fitness* and the *current time step*. The fitness values are normalized so that they lie in the range $[0; 1]$, while the optimization time is limited by T. Each of the two variables is binned into 5 equally-sized intervals, resulting in the intervals $\left(\frac{j}{5}; \frac{j+1}{5}\right]$ and $\left(\frac{T \cdot j}{5}; \frac{T \cdot (j+1)}{5}\right], j = 0, 1, 2, 3, 4$, respectively. That is, we have 25 states in total, and 5 actions (the static constraint-handling strategies) are available in each state.

Our learning goal is to achieve a high population or individual fitness at the end of the search. Consequently, the only reward we provide is the *average fitness of the population* at the end of an episode; i.e. $r_i = 0$ for $i = 1, ..., T$, and $r_{T+1} \in [0, 1]$. Alternatively, the reward may be the best solution fitness found.

There are some further aspects related to the particular learning problem of switching between constraint-handling strategies. First, note that a non-evaluable solution may be encountered at different stages of the optimization process or not at all in a single EA run. In other words, the number of states visited in an episode, and the number of non-evaluable solutions encountered in a state, may vary between episodes and states, respectively. In the case where a non-evaluable solution is encountered, the first action selected in a particular state is applied to all non-evaluable solutions encountered in this state (see Line 17 and 18 of Algorithm 5.2). This selection approach tends to perform better than allowing an optimizer to reselect actions when in one and the same state, because it is more direct in terms of credit assignment.

5.2.2 Online learning-based strategy

To learn online when to switch between static constraint-handling strategies, we use an adaptive operator selection method known as the dynamic multi-armed bandit (D-MAB) algorithm (Hartland et al., 2006, 2007; Costa et al., 2008).[9] The idea of D-MAB is to consider the learning problem as a multi-armed bandit problem with the static strategies serving as independent arms. D-MAB extends the *upper confidence bound 1* (UCB1) algorithm (Auer et al., 2002) with the statistical Page-Hinkley (PH) test (Page, 1954), which has the purpose to detect changes in the sequence of rewards obtained, and then to restart the multi-armed bandit. The algorithm, D-MAB, is described in detail in Appendix B.

D-MAB requires that the play of an arm is followed by a subsequent reward. We provide a reward immediately after the play of an arm, and it is the (normalized) raw fitness of the resulting solution. Note that if the arm associated with the strategy, *waiting*, is played, then the reward may be available only a few time steps later. Also, in the case of *penalizing*, the reward will always be some poor fitness value c. However, this does not mean that penalizing is never selected because (i) UCB1 maintains always a certain degree of exploration, and (ii) a restart of the multi-armed bandit triggered by the PH test puts all arms in the same initial position. Alternative credit assignment schemes and UCB algorithms were tested but the combination used in this chapter tends to perform, on average, best and most robustly on the test problems considered.

We want to point out here that certain credit assignment schemes cannot be readily applied when dealing with ERCs. For instance, a common scheme is to use a credit based on the fitness improvement of an offspring compared to its parent after applying a variation operator to it. In our scenario the parent would be the individual that is to be repaired and the offspring the repaired individual after applying a constraint-handling strategy to the parent. As we do not know the fitness of the parent because it is non-evaluable, we cannot quantify by how much its fitness differs from the one of the repaired individual.

[9] In (Hartland et al., 2006, 2007), various versions of a D-MAB algorithm are proposed but the version we use here is referred to as γ-restart.

Table 5.4: Parameter settings of static and learning-based constraint-handling strategies.

Strategy	*Parameter*	*Setting*
Regenerating	Number of regeneration trials L	10000
Penalizing	Fitness c assigned to non-evaluable solutions	0
Subpopulation strategy	Maximal size of all subpopulations SP_h, J_h	25
RL-EA	Decay factor λ	1.0
	Discount rate γ	1.0
	Learning rate α	0.1
	Probability ϵ of selecting a random action	0.1
	#Training episodes	5000
	#Testing episodes	100
D-MAB	Threshold parameter λ_{PH}	0.1
	Tolerance parameter δ	0.01
	Scaling factor C	1

5.2.3 Experimental analysis

Experimental setup: Here, we use a slightly different experimental setup than we used in the analysis of the static constraint-handling strategies (see Section 5.1.3). The changes concern the reproduction scheme of the EA on which we augment the constraint-handling strategies, and the maximal size of the subpopulation SP, J, used within the subpopulation strategy. The reason for using a modified setup is that we specifically tuned the EA and constraint-handling strategies to perform well on the test problems considered.

Rather than using an elitist generational reproduction scheme, we employ an EA with a steady state or $(\mu + 1)$-ES reproduction scheme. Algorithm 5.2 shows the interplay between the static and learning-based constraint-handling strategies within this EA. The method $currentState(\overline{Pop}, t)$ (Line 17 and 18) called by the RL-based approach returns the current state s_t based on the state variables $\overline{Pop}$ and t; the method $giveAction(s_t, rand(0, 1))$ (Line 18) returns the action a_t based on the ϵ-greedy method.

The parameter settings of the EA are identical to the settings used in the previous study (see Table 5.1) with the difference that $\lambda = 1$. Table 5.4 shows

Algorithm 5.2 Steady-state EA with static and learning-based constraint-handling strategies

Require: ERC_1,...,ERC_r (set of ERCs), f (objective function), T (time limit), μ (parent population size), *offlineLearning* (boolean variable indicating whether offline learning is applied (true) or online learning (false))

1: $t = 0$ (global time counter), $Pop = \emptyset$ (current population), $s_t = -1$ (variable representing current state), #Strategy = -1 (variable indicating the number of the static constraint-handling strategy currently in use)
2: **while** $|Pop| < \mu \ \wedge \ t < T$ **do**
3: generate solution $\vec{x}$ at random
4: $\vec{x}$ = functionWrapper($\vec{x}, t$)
5: $Pop = Pop \cup \{\vec{x}\}$; t++
6: **while** $t < T$ **do**
7: generate two offspring by selecting two parents from *Pop*, recombining and mutating them; pick one of two offspring $\vec{x}$ at random
8: $\vec{x}$ = functionWrapper($\vec{x}, t$)
9: form new *Pop* by selecting the best μ solutions from the union population $Pop \cup \{\vec{x}\}$, t++
10: **if** *offlineLearning* **then**
11: set reward $r_{T+1} = \overline{Pop}$ and update the function $Q(s, a)$ (see Line 8 of Algorithm 3.4); clear trace $e(s, a) = 0, \forall s \in S, a \in A(s)$, and reset s_t = -1 and #Strategy = -1

12: functionWrapper($\vec{x}, t$){
13: y_t = null
14: **if** $\vec{x}$ satisfies the ERCs ERC_1,...,ERC_r **then**
15: $\vec{x}_t = \vec{x}$; $y_t = f(\vec{x}_t)$
16: **else**
17: **if** *offlineLearning* $\wedge \ s_t \neq$ currentState($\overline{Pop}, t$) **then**
18: s_t = currentState($\overline{Pop}, t$); a_t = #Strategy = giveAction(s_t,rand(0,1)); $r_{t+1} = 0$; update trace e according to Equation (3.7) and the function $Q(s, a)$ (see Line 8 of Algorithm 3.4)
19: **if** !*offlineLearning* **then**
20: #Strategy = giveBestArm() // Select the best arm using the UCB1 algorithm
21: **if** #Strategy = 1 $\vee$ #Strategy = 2 $\vee$ #Strategy = 3 **then**
22: $\vec{x}_t$ = repair($\vec{x}, t$); $y_t = f(\vec{x}_t)$
23: **if** #Strategy = 4 **then**
24: $t = t + \tau$; $\vec{x}_t = \vec{x}$; $y_t = f(\vec{x}_t)$ // τ is the number of time steps we have to wait until $\vec{x}$ is evaluable
25: **if** #Strategy = 5 **then**
26: $\vec{x}_t = \vec{x}$; $y_t = c$ // c is a constant, representing poor fitness
27: **if** !*offlineLearning* **then**
28: normalize y_t and assign it as reward to arm j = #Strategy; update the variables $n_j, \bar{p}_j, m_j$, and M_j, and perform the PH test; if change-point detected, then set $n_j, \bar{p}_j, m_j$, and M_j ($j = 1, ..., J$) to 0
29: return $\vec{x}_t$ and y_t}

the parameter settings of the constraint-handling strategies. For the RL-based strategy, denoted in Table 5.4 by RL-EA, we use a training and testing scheme (similar to Pettinger and Everson (2003)). In the training phase, the RL agent estimates the function $Q(s, a)$, while, in the testing phase, the Q-function is frozen

and the greedy actions a^* are always selected; i.e. during the testing phase we set $\epsilon = 0$. As specified in Table 5.4, the testing phase involves 100 episodes or EA runs, and this is also the number of runs for which we run the other constraint-handling strategies; i.e. any results shown are average results across 100 EA runs. To allow for a fair comparison of the strategies, we use a different seed for the random number generator for each EA run but the same seeds for all strategies. We also use a different seed for each episode of the training phase of RL-EA.

Experimental results: Let us imagine that we are faced with a similar closed-loop scenario as in the case study of Section 5.1.4. Suppose our optimization problem is binary and subject to the following two, *a priori* known, commitment relaxation ERCs: $commRelax\text{-}\ ERC(0, 2000, 20, H = (10101 * * * ...))$ and $commRelaxERC(0, 2000, 20, H = (*... * *101))$. That is, one ERC constrains the first 5 solution bits, while the other the last three bits. Solutions shall be represented by a binary strings of length $l = 30$, and we set the optimization time to $T = 2000$ time steps. As the test function f let us consider the standard NK landscapes or, equivalently, $NK\alpha$ landscapes with $\alpha = 0$ (see Section 5.1.2.2). For the two tunable parameters of NK landscapes, N and K, we consider four different settings for K, namely, $K = 1, 2, 3$, and 4, while N is fixed to $N = l = 30$. These settings will allow us to cover reasonable degrees of epistasis.

Let us now analyze the performance of the static and learning-based constraint-handling strategies on the different NK landscapes. Figure 5.13 shows the population average fitness obtained with the different constraint-handling strategies on the four NK landscapes as a function of the time counter. The average fitness of the best solution in a population is in alignment with the population average fitness. From the plots we can see that the ERCs affect the performance of an EA. Also, the plots confirm what we mentioned before that similar patterns are obtained for all the test functions. In fact, with respect to the static strategies, we observe a trend that the subpopulation strategy performs best in the initial stages of the optimization, forcing and penalizing in the middle part of the optimization, and waiting in the final stages of the optimization. Also, the time step at which waiting outperforms penalizing shifts further to the right as the degree of epistasis increases. The behavior of the static constraint-handling strategies is similar to the one we made in the case study of Section 5.1.4.

For RL-EA, the results in Figure 5.13 were obtained using a training phase involving 5000 different NK landscapes with $N = 30$ and $K = 2$. After that

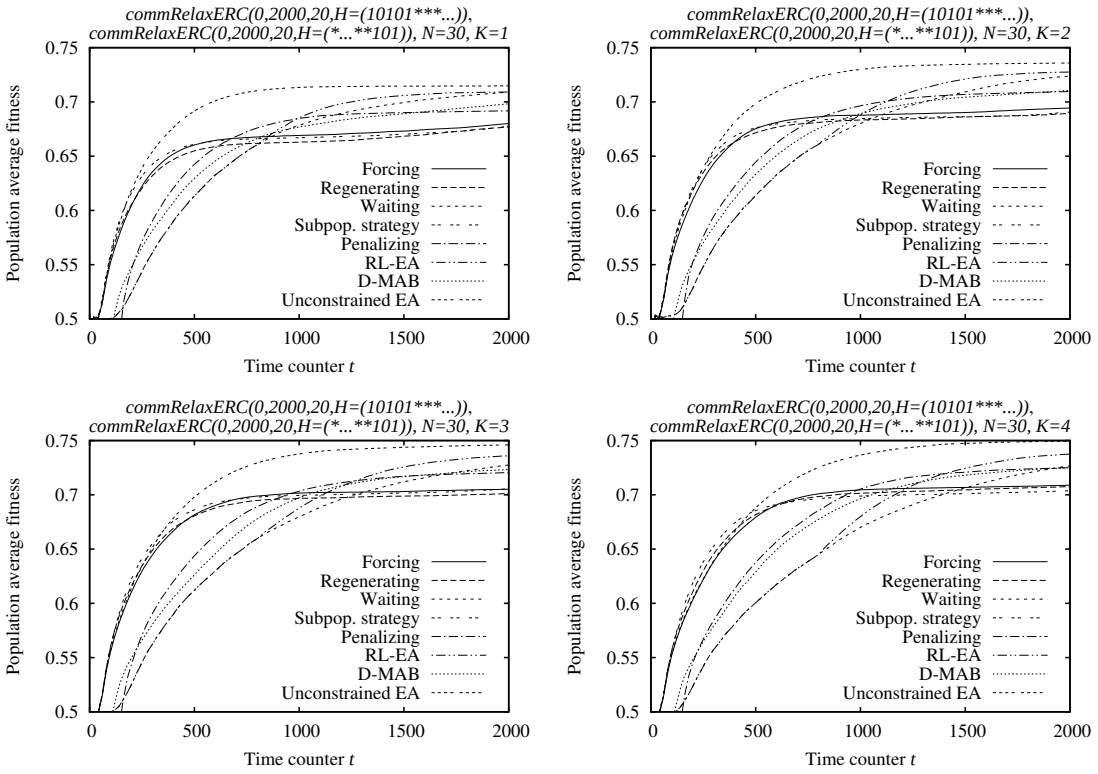

Figure 5.13: Plots showing the population average fitness (we do not show the standard error as it was negligible) obtained by the different constraint-handling strategies on NK landscapes with $N = 30$ and $K = 1$ (top left), $K = 2$ (top right), $K = 3$ (bottom left), and $K = 4$ (bottom right) as a function of the time counter t; results are averaged over 100 independent runs using a different randomly generated NK problem instance for each run. All instances were subject to the commitment relaxation ERCs $commRelaxERC(0, 2000, 20, H = (10101 * * * ...))$ and $commRelaxERC(0, 2000, 20, H = (*...**101))$. The results of 'Unconstrained EA' were obtained by running the EA on the same problem instances but without the ERCs.

training phase, the same frozen Q-function was used in the testing phase of all four NK landscape types. Consequently, this allows us to assess the robustness of the control policy learnt as training and testing are done in environments with different properties (at least for the cases $K \neq 2$). From the figure we can see that RL-EA is the best performing strategy on all four NK landscape types at $T = 2000$. For a low level of epistasis, RL-EA is even able to match almost the performance of an EA optimizing in an ERC-free environment. We observe also that the performance advantage over waiting, the second best strategy, tends to increase with the degree of epistasis; when comparing the two strategies against each other using more than 100 algorithmic runs, then, according to the Kruskal-Wallis test (significance level of 5%), RL-EA is also significantly better than

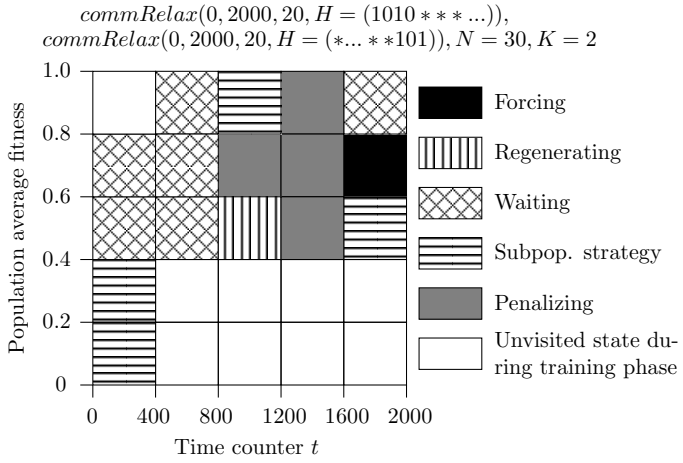

Figure 5.14: A plot showing the greedy actions a^* learnt by the RL agent for each state s. Training was done on NK landscapes with $N = 30$ and $K = 2$.

waiting for $K = 3$ and $K = 4$. This aspect indicates the robustness of RL-EA. On the other hand, although D-MAB is not able to perform as well as RL-EA, it is able to match the performance of waiting for $N = 30, K = 4$. Another benefit of offline learning is that if the optimization time would be shorter, say around $T = 500$, then RL-EA would learn a different control policy, while D-MAB would not. That is, RL-EA adapts to the real-world problem at hand; we have confirmed this experimentally (results not shown).

Figure 5.14 shows the greedy action a^* in each state s learnt by the RL agent. From the plot it is apparent that the agent learnt to use mainly waiting at the beginning of the optimization process, penalizing in the middle part of the optimization, and, depending on the population average fitness, either forcing, waiting, or the subpopulation strategy, in the final part of the optimization. From Figure 5.13 one may conclude that a policy that uses a repairing strategy (forcing, regenerating, or subpopulation strategy) at the beginning of the optimization, and waiting or penalizing towards the end, should also perform well. The agent did not learn this policy because the repairing strategies may lead quickly to a homogeneous population containing many solutions that fall into both or either of the constraint schemata. If the schemata are poor, then one should escape from this population state but this is difficult as diversity needs to be again introduced into the population and this takes too long using waiting or penalizing.

Figure 5.15 illustrates what strategies have been used on average by D-MAB during different periods of the optimization process of $N = 30, K = 4$. From

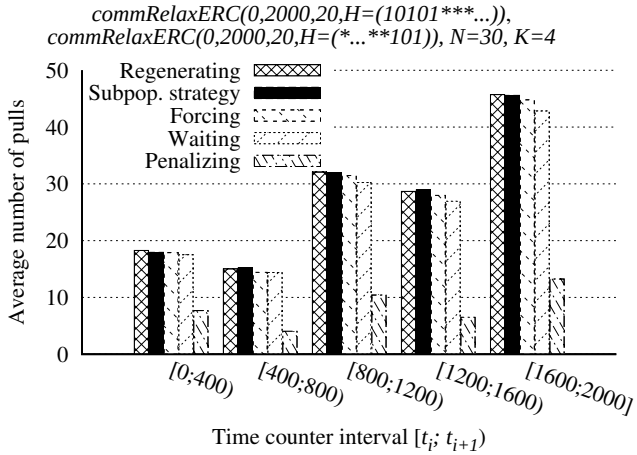

Figure 5.15: A plot showing the average number of times each strategy is played by D-MAB at different periods of the optimization process. Results are shown for $N = 30$ and $K = 4$.

the plot we can see that penalizing is selected least often, which is due to the low fixed reward associated when playing the associated arm. A trend is obvious that waiting is used less often than the repairing strategies at the end of the optimization process, which is in alignment with the policy learnt by the RL agent. The reason that the performance of D-MAB is nevertheless poorer than the one of RL-EA is that the repairing strategies are used too often throughout the optimization process but in particular at the beginning of the optimization. As mentioned before, this may result in a homogeneous population state from which it is difficult to escape; the fact that the total number of plays increases towards the end of the optimization confirms that D-MAB actually ends up in this poor population state. On *NK* instances with lower epistasis, the PH test is triggered less often, and waiting is used almost as often as the repairing strategies throughout the optimization.

Overall, the strong performance of the RL-EA is encouraging, but we want to mention that in order to achieve that performance, some tuning of the agent may be required. This is due to two aspects. First, the stochastic nature of an EA in the sense that: (i) taking the same action in a particular state but in different episodes may cause an EA to end up in different future states, and (ii) visiting the same state-action pairs in different episodes may result in different rewards obtained at the end of an episode. Secondly, since a different problem instance is used in each training episode, the constraint schemata of the two ERCs may represent both good and poor sets of solutions. Hence, as the quality of a schema has an impact

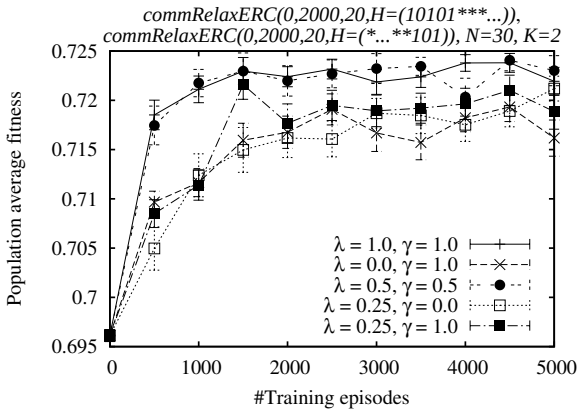

Figure 5.16: A plot showing the population average fitness obtained by RL-EA for different values of the decay factor λ and the discount rate γ as a function of the number of training episodes. The population average fitness values themselves represent average values across 30 learning trials; each trial used different randomly generated problem instances and random number generator seeds for training and testing, but the same instances and seeds for any combination of λ and γ values. Training and testing was done on NK landscapes with $N = 30$ and $K = 2$

on what strategy is most appropriate, the current optimal control policy learnt by the RL agent may change constantly during training. The approach taken here to deal with these two issues was to train the agent for different numbers of training episodes using also different settings of parameters involved in the agent algorithm. The parameter setting combination that performed, on average, best and most robustly on the training or validation problems was used.

Figure 5.16 illustrates how the population average fitness may be affected by the number of training episodes, and different settings of the decay factor λ and the discount rate γ. From the figure it is apparent that the RL agent is able to learn an optimal policy after around 1000 episodes or algorithmic runs. Furthermore, while the agent is relatively robust to the setting of λ and γ, optimal performance tends to be obtained for configurations where both parameters take large values. The meaning of these configurations is that we should not discount any reward obtained during an episode (large γ) nor should action-value estimates $Q(s, a)$ be updated based on other action-value estimates (large λ). These are sensible configurations because the aim of discounting (small γ) is to reach a terminal state more quickly; this is not the case in our scenario because we provide a reward always after visiting the same amount of simulated time. The fact that we obtain better performance by updating visited state-action pairs (s, a) based on the actual rewards obtained (small values of λ) is an indication that the states

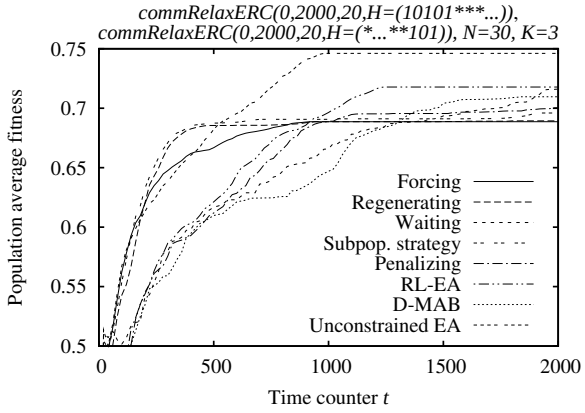

Figure 5.17: A plot showing the population average fitness obtained in a single run on a randomly generated NK landscape with $N = 30$ and $K = 3$ as a function of the time counter t; the instance was subject to the two commitment relaxation ERCs $commRelaxERC(20, H = (10101***...))$ and $commRelaxERC(20, H = (*...**101))$.

we visited in an episode are not significantly dependent on the actions taken; this is true in our scenario because the EA passes through approximately the same state trajectory in an episode. Of course, different settings of γ and λ may be more suitable for more fine-grained state space realizations.

Overall, based on the results shown, we would select RL-EA for the real-world closed-loop problem as it performs best after 2000 time steps. Let us assume that the real-world problem is a NK landscape instance with $N = l = 30$ and $K = 3$. Hence, to verify our selection, we perform one run with all strategies on a single newly generated NK landscape instance with $N = 30$ and $K = 3$; the results are shown in Figure 5.17, and note that RL-EA uses the control policy of Figure 5.14. The plot confirms the findings made on the test functions. In particular, RL-EA, D-MAB, and waiting, perform best at the end of the run while the static repairing strategies perform best at the beginning of the optimization.

5.2.4 Summary and conclusion

In this study we have investigated two learning-based constraint-handling strategies for coping with commitment relaxation ERCs. The learning-based strategies aim at learning when to switch between static constraint-handling strategies, which we introduced in Section 5.1.1, during the optimization process. However, while one strategy learns this task offline using a reinforcement learning (RL) agent, here Sarsa(λ), the other strategy performs online learning using the UCB

algorithm extended with the statistical Page-Hinkley test to detect changes in the sequence of rewards obtained. Offline learning is possible in this optimization scenario because the performance of an EA depends largely on the type of ERC, with similar effects observable for different fitness landscapes; this observation has been indicated by the study conducted in Section 5.1.3.

The experimental analysis compared the learning-based constraint-handling strategies against the static strategies themselves, and it concluded that ERCs affect the performance of an EA but an RL-based strategy is able to get close to the performance of an EA optimizing in an ERC-free environment, particularly on fitness landscapes with little epistasis. The online learning algorithm did not perform as well as the RL-based algorithm, mainly because it does not look ahead in the optimization process and so may end up quickly in a poor population state from which it is difficult to escape; this is a general issue of online learning within an ERCOP scenario and it is yet unclear how to avoid it. Static repairing strategies are well suited if little optimization time is available, while a static waiting or penalizing strategy is more suited for longer optimization times.

To improve the performance of learning-based constraint-handling strategies, in particular, offline learning-based techniques, one may look at the design and tuning of RL agents that also account for the stochasticity present in evolutionary search (e.g. the P-Trace and Q-Trace algorithm of Pendrith (1994), or the Kalman filtering approach of Chang et al. (2004)). Analyzing constraint-handling and learning strategies on different and perhaps more realistic fitness landscapes than *NK* landscapes is another important research avenue. Furthermore, in addition to learning the task of switching between constraint-handling strategies, an agent may be required to arrange its own resources required in the evaluation of solutions. The next section investigates strategies for coping with such scenarios. Whilst these strategies do not rely on RL algorithms, extending them with ideas from RL is certainly an interesting and promising research direction.

5.3 Online resource-purchasing strategies

So far, this chapter has investigated strategies for dealing with non-evaluable solutions arising due to ERCs. In this section, our focus shifts to online resource-purchasing strategies to cope with commitment composite ERCs, which we introduced in Section 4.2.5. Recall that with this type of ERC, some part of the

solution vector defines a complex subpart or *composite* (expressed by a high-level constraint schema $H_{\#}$) that must be purchased online and pre-emptively so that it arrives in time to be used in the evaluation process of a solution. Once the composite arrives (after some lag of TL time steps) it can be stored for a limited period of time only (because composites have a shelf life of SL time steps) and/or re-used in different experiments only a certain number of times (RN denotes the reuse number); we assume $SL \geq RN$. The number of composites available simultaneously is upper bounded by the number of storage cells $\#SC$. We also assume a budget C limiting the usage of the composites and/or limiting time. In the next section we devise three resource-purchasing strategies (for use in a generational EA) to cope with this type of ERCs. Before we deploy and test these strategies empirically over a number of resource-constraint settings and budget regimes in Section 5.3.3, we define the experimental setup in Section 5.3.2. Section 5.3.4 draws together the findings of the experimental study.

5.3.1 Specific online purchasing strategies

This section proposes three online resource-purchasing strategies for coping with commitment composite ERCs: a just-in-time strategy, a just-in-time strategy with repairing, and a sliding window strategy. All strategies are augmented on a generational EA and designed such that the EA has never to deal with solutions that have a null fitness value (although null solutions may need to be submitted purposely to bridge waiting periods between arriving composite orders).

In general, a strategy designed to deal with commitment composite ERCs is composed of three mechanisms, which are concerned with: (i) selecting the composite that is ordered and the point of time at which it is ordered, (ii) selecting the storage cell into which an arrived composite is put, which may mean selecting an existing composite that is to be replaced by a new one, and (iii) selecting an available composite to repair a non-evaluable solution (given it is to be repaired). Each of these mechanisms is described for our strategies in the following. As before, we assume that time steps refer to function evaluations of single solutions.

5.3.1.1 Just-in-time strategy

Without repairing and any composites in storage, the minimum number of time steps for evaluating all solutions of a population Pop (of size μ) is $\mu + TL$ (TL

time steps are needed for the first composite to arrive; this period is bridged by submitting TL null solutions). This is achieved if orders are processed in contiguous groups organized by the composites they require, e.g. *ccbbbadd...*, where a, b, c, and d shall represent different composites required by solutions. Thus, the first main mechanism of the *just-in-time* (JIT) strategy is to arrange solutions of a population *Pop* into these contiguous groups, and then to make purchase orders so that composites arrive just in time for the scheduled experiment time. This mechanism, which is performed by the method *alignOrdersAndSolutions(Pop)* (see below), prolongs the availability of resources.

When composites are already in storage (we call them *old composites*) savings in purchase orders may be made if those composites are used first. For example, suppose the optimizer requires the composites *ccdadcac*, and composite a is available in one of the storage cells and has 3 uses and 5 time steps of its shelf life remaining. Then, by placing a first, the permutation *aaccccdd* will save us a purchase order since only two a composites are needed. Thus, the second main mechanism of the JIT strategy is to efficiently schedule the evaluation of solutions in a population *Pop* using old composites. This is performed by the method *useUpOldComposites(Pop)* (see below). Notice that, at any given time, JIT (and JIT with repairing) maintain non-identical composites in storage.

During the optimization, the JIT strategy checks at each generation whether old composites can be used in the evaluation of solutions of the current population. If so, the method *useUpOldComposites(Pop)* is applied first and then the method *alignOrdersAndSolutions(Pop)*, otherwise we proceed directly with the method *alignOrdersAndSolutions(Pop)*.

alignOrdersAndSolutions(*Pop*): Recalling that solutions are grouped by the composite they require; this method must just choose the order in which these groups of solutions are to be submitted for evaluation. To obtain a permutation of groups we select groups one by one using roulette wheel selection (without replacement) based on the number of solutions in *Pop* associated with them, and then reverse the order so obtained. This strategy increases the chances that left-over composites at the end of the generation will be useful next generation (because the solutions associated with them are over-represented in the population). After aligning composite orders with the evaluation sequence of (groups of) solutions, we know exactly when composite orders need to be submitted for them to arrive just in time for being used in the evaluation of solutions.

After the arrival of a composite we need to decide the storage cell into which we put it. The default is to place them in an empty storage cell. If no cell is empty, then the arriving composite replaces the old composite that can be used in the fewest evaluations within the subsequent generation. That is, if P is the current set of old composites and associated with each $p \in P$ there is the remaining shelf life ψ_p and a remaining number of reuses ρ_p, then an arriving composite replaces the composite $p^* \in P$ given by $\min_{p \in P}(\psi_p - (t_{\text{new generation}} - t), \rho_p)$, where $t_{\text{new generation}}$ is the time step when the subsequent generation begins, and t the current time step. Remember that we are aware of $t_{\text{new generation}}$ because the entire schedule for the evaluation of *Pop* is done beforehand. In the case where there are multiple composites p^*, ties are broken by replacing the composite that has the shortest shelf life remaining; further ties are broken randomly.

useUpOldComposites(*Pop*): Our aim when using up old composites is to save as many composite orders as possible. To achieve this when there are several storage cells to consider, we solve the lexicographical optimization problem $\text{lexmax}_{i \in \pi(Z)}(F_1, F_2)$ over the permutation set of

$$Z = \{\zeta | \zeta \in P \cap C \wedge \theta_\zeta \text{mod} RN \leq \min(\psi_\zeta, \rho_\zeta)\}, \tag{5.1}$$

where P, ψ_p and ρ_p have the same meaning as before, and C is the set of composites required by the optimizer to evaluate *Pop*; associated with each $c \in C$ there is a required number θ_c. The first condition, $\zeta \in P \cap C$, says that a composite needs to be currently in storage and it needs to be required by at least one solution of the population *Pop*. The second condition, $\theta_\zeta \text{ mod} RN \leq \min(\psi_\zeta, \rho_\zeta)$, ensures that the only left-over solutions considered for evaluation with the old composites are ones that do not require the ordering of a new composite.[10]

The objective functions to be maximized are $F_1 = \sum_{j \in 1,\ldots,|Z|} f_1(\pi_{ij}(Z))$ and

[10]Remember that we assume that $SL \geq RN$, meaning that the maximum number of evaluation steps associated with a composite is limited by its reuse number. If the shelf life would be the limiting factor, then the condition would be $\theta_\zeta \text{ mod} SL \leq \min(\psi_\zeta, \rho_\zeta)$.

$F_2 = \sum_{j \in 1,\ldots,|Z|} f_2(\pi_{ij}(Z))$, where π_{ij} is the jth element of permutation i,

$$f_1(\pi_{ij}(Z)) = \begin{cases} 1 & \text{if } j = 1 \\ 1 & \text{if } \psi_{\pi_{ij}} - \left(\sum_{k=1}^{j-1} f_1(\pi_{ik}(Z)) \cdot \theta_{\pi_{ik}} \text{mod} RN\right) \geq \theta_{\pi_{ij}} \text{mod} RN \\ 0 & \text{otherwise,} \end{cases} \tag{5.2}$$

and

$$f_2(\pi_{ij}(Z)) = f_1(\pi_{ij}(Z)) \cdot \theta_{\pi_{ij}} \text{mod} RN. \tag{5.3}$$

F_1 determines the number of purchase orders saved for each permutation of Z, while F_2 breaks ties among the best permutations by selecting the permutation(s) that evaluate the most solutions; further ties are broken randomly. Required composites left in storage after evaluating the last solution of the selected permutation (i.e. after $\max(F_2)$ time steps) are used up further in a random fashion; i.e. we first select a composite at random and then select a random (unscheduled) solution from *Pop* that requires this composite, and we repeat this until no future evaluations can be scheduled. For small sizes of Z, the lexicographical optimization problem $\text{lexmax}_{i \in \pi(Z)}(F_1, F_2)$ can be solved to optimality by exhaustive search (see Figure 5.18); this is the approach taken here.

Having devised a schedule for using up the old composites, and selecting the solutions to be evaluated with these composites, we can now specify at which time step the first new composite order needs to be submitted (to evaluate the remaining solutions in *Pop*) in order to reduce our waiting time for it to arrive. Subsequently, we know when and how many null solutions need to be submitted.

5.3.1.2 Just-in-time strategy with repairing

Although the JIT strategy avoids repairing of solutions and thus allows an optimizer to perform an unconstrained optimization, this may be associated with a waste of up to $\mu \times (RN - 1)$ reuses per generation (if each solution of the population requires a different composite). To reduce wastage, the *JIT with repairing* (JITR) strategy extends the basic JIT strategy with a repairing procedure. The method *performRepairing(Pop)* (see Algorithm 5.3) performs the repairing

1)

p	a	b	d	e	f	r
ψ_p	12	4	6	5	6	19
ρ_p	8	5	6	8	4	9

c	b	d	e	f	g
θ_c	3	14	21	5	7
$\theta_c \bmod RN$	3	4	1	5	7

2) $Z = \{b, d, e\}$ (composite f is not a member of Z because $\theta_f \bmod RN > \min(\psi_f, \rho_f)$)

3)

i	π_{ij}			$f_1(\pi_{ij}(Z))$			F_1	F_2
	$j=1$	$j=2$	$j=3$	$j=1$	$j=2$	$j=3$		
1	b	d	e	1	0	1	2	4
2	b	e	d	1	1	0	2	4
3	d	b	e	1	0	1	2	5
4	d	e	b	1	1	0	2	5
5	e	b	d	1	1	0	2	4
6	e	d	b	1	1	0	2	5

4) Choose at random between permutations $i = 3, 4$, and 6

5) Use up left-over composites from the set P if they are needed in the evaluation of the current population. Regardless of the permutation selected in this example, one evaluation will be left-over with composite d (because $\min(\psi_d, \rho_d) - F_2 = 1$)

Figure 5.18: Visualization of the internal calculations made by the method *useUpOldComposites(Pop)*. We assume a commitment composite ERC with $RN = 10$ and $SL = 20$. We need to perform five steps to process the population *Pop* (containing in this example $\sum_c \theta_c = 50$ individuals): 1) determining the parameters ψ_p, ρ_p, and θ_c; 2) determining the permutation set Z; 3) computing the objective functions F_1 and F_2 for each permutation π; 4) selecting a random permutation in case there are equally good permutations; 5) match unscheduled solutions of *Pop* and composites (from set P) in a random fashion if any composites are left after realizing the selected permutation.

and this method is always called before calling the method *alignOrdersAndSolutions(Pop)* (but after *useUpOldComposites(Pop)*). The idea of the repairing strategy is to repair solutions such that they use a composite that is *nearly* the one required. Solutions to be repaired are identified by first clustering their composites, and then trying to find an assignment of solutions to clusters that minimizes the total Hamming distance of all repairs.

performRepairing(*Pop*): The first step of this method is to filter out all solutions from the population *Pop* that use up a composite entirely (without being repaired); *Pop*′ shall denote the resulting solution. If there are more than RN solutions requiring the same composite type, then we filter out a subset of RN solutions among them at random. The remaining solutions take part in the clustering process and are subject to being repaired. The number of clusters k into which we can partition these solutions (or their required composites), varies between

Algorithm 5.3 Just-in-time strategy with repairing

Require: w (weighting factor), $\#mIter$ (number of shifting rounds)

performRepairing(Pop){
create Pop' by filtering out solutions from Pop that use up composites entirely
for $\left\lceil \frac{|Pop'|}{RN} \right\rceil \leq k \leq \#CompsInPop$ **do**
    cluster solution in Pop' using k-medoids
    if required, shift solutions between clusters (perform $mIter$ rounds of shifting); retain configuration with the smallest Hamming distance loss and record HDL_k
normalize the values HDL_k and k of all configurations, and calculate the scores $HDLN_k \cdot (1-w) + kN \cdot w, \forall k$, where $HDLN_k = \frac{HDL_k - \min_k(HDL_k)}{\max_k(HDL_k) - \min_k(HDL_k)}$ and $kN = \frac{k - \left\lceil \frac{|Pop'|}{RN} \right\rceil}{\#CompsInPop - \left\lceil \frac{|Pop'|}{RN} \right\rceil}$
repair solutions in Pop' according to the configuration associated with the smallest score or $\min_k(HDLN_k \cdot (1-w) + kN \cdot w)$; add the filtered-out solutions back to Pop' to create a new population Pop''.}

$\left\lceil \frac{|Pop'|}{RN} \right\rceil \leq k \leq \#CompsInPop$, where $\lceil . \rceil$ is the ceil function and $\#CompsInPop$ the number of non-identical composites in Pop'. Let us first describe how we perform the clustering for a given k, and then how we select a particular cluster configuration according to which we repair.

To obtain a partition with k clusters we apply k-medoids (Kaufman and Rousseeuw, 1990) on the different composites.[11] The distance between two composites is the Hamming distance between their constraint schemata. The medoid composite of a cluster is the composite that would be used to repair (using forcing) all solutions in that cluster that require a different composite. For diversity reasons, we want to avoid ordering a medoid composite more than once, or, in other words, we want to keep the number of solutions within a cluster smaller than the reuse number RN. A simple way to ensure this is to randomly shift solutions from clusters that are overly large to clusters that can accommodate further solutions. We perform $mIter$ shifting rounds in total and select the configuration with the smallest total Hamming distance of all repairs HDL_k to be the best configuration obtained with k clusters; formally we can define this selection step as:

$$HDL_k = \min_{i=1,\ldots,mIter} \left(\sum_{j=1}^{k} \left\{ \sum_{r \in CL_{j,i}} HD(H_j, H_{j,r,i}) \right\} \right) , \tag{5.4}$$

[11] The k-medoids algorithm divides a set of n data points into k groups (clusters), with k being known *a priori*, in such a way that the sum of pairwise dissimilarities between the data points of a cluster is minimal. The medoid of a cluster is the data point whose average dissimilarity to all other points in that cluster is minimal. In other words, the medoid is the most centrally located data point in a cluster.

where H_j represents the constraint schema of the medoid composite in cluster j, and $H_{j,r,i}$ the rth solution in the set $CL_{j,i}$, which is the cluster j under shifting configuration i; note that while the medoid remains the same for different shifting rounds, the set $CL_{j,i}$ depends on how solutions are shifted between clusters. The function $HD(.,.)$ computes the Hamming distance between two schemata.

The cluster configuration according to which we repair is the one with the smallest score $HDLN_k \cdot (1-w) + kN \cdot w$, where $HDLN_k$ and kN are the normalized values of HDL_k and k (see Line 6 of Algorithm 5.3), and $w \in [0,1]$ a predetermined weighting factor representing the degree of repairing. Roughly speaking, the larger the value w the more solutions are repaired; note, a setting of $w = 0$ would result in the basic JIT strategy because the smallest total Hamming distance of all repairs is achieved by not repairing at all. We can now add the solutions that were filtered out again to Pop' to form a new population Pop'', which is processed further as usual using the method *alignOrdersAndSolutions(Pop'')*.

5.3.1.3 Sliding window strategy

While JIT and JITR operate in a sequential mode in that they devise a schedule of solution evaluations and purchase orders upon receiving a population from the EA first, the *sliding window* (SW) strategy deals with the working of the algorithm and the purchasing of orders in parallel. More precisely, the strategy submits solutions for evaluation in the order they are generated, non-evaluable solutions are always repaired, and it aims for the 'most useful' composites to be kept in storage by (i) replenishing composites so that there can never be an empty storage cell and (ii) maintaining composites that were recently requested by the optimizer.

The first aspect, avoiding empty storage cells, is achieved by making purchase orders to fill all storage cells every RN time steps (because $SL \geq RN$). The second aspect, maintaining recently requested composites, is achieved by ordering composites from the sliding window, which we define here as a set $\kappa(t)$ containing composites that were requested most recently but were unavailable at the time of the request; $\kappa(t)$ is limited in size such that $|\kappa(t)| \leq WS$, where WS is the *window size*. Here, a requested composite means one that was part of a solution that the EA submitted to the function wrapper for evaluation. In this thesis, we simply order the $\#SC$ composites from $\kappa(t)$ that have been added to this set most recently. That is, if we set $WS = \#SC$, then we order always all composites in

$\kappa(t)$; note, this means that all composites in the storage cells are replaced upon the arrival of new composites. If we cannot order a composite for each storage cell because $|\kappa(t)| < \#SC$, then we order random composites for the remaining storage cells; this is, for example, the case at $t = 0$ where the set $\kappa(0)$ is empty.

To repair a non-evaluable solution, we use the composite from the storage cell that has the smallest Hamming distance to the actually required composite; ties between equally distant composites are broken randomly. This repairing step is encoded within the repairing method, $repair(\vec{x}, t)$, of the function wrapper (see Line 36 of Algorithm 5.4). This is different to the JITR strategy, which applies repairing after generating an offspring population (see Line 15 of Algorithm 5.4).

5.3.2 Experimental setup

We augment the three online resource-purchasing strategies on the same elitist generational EA as used in the analysis of the static constraint-handling strategies (see Section 5.1.2.1); we also use the same parameter settings (see Table 5.1). Algorithm 5.4 shows how we incorporate the purchasing strategies into this EA.[12]

As the objective function f we consider again the MAX-SAT problem instance introduced in Section 5.1.2.2 (i.e. solutions are represented as binary strings of length $l = 50$). Any results shown are average results across 100 independent algorithm runs on this instance. Similarly to our previous studies, we choose the order-defining bits of the high-level constraint schema $H_{\#}$ at random at each run but, of course, use the same schemata across the strategies analyzed.

Table 5.5 gives the settings of the online purchasing strategies. JIT is parameter-free and thus not included in this table. For JITR, we anneal the weighting factor w stepwise as a function of the cost counter c as $w = \max(0, 0.1 \times \max(0, 4 - \lfloor c/250 \rfloor))$. This means that the weighting factor is decreasing in the range [0.4,0.1], resulting in a decreasing number of repairs as the optimization proceeds. The factor remains constant at $w = 0.1$ (i.e. only little repairing is applied) for $c \geq 1000$. This setting performed most robustly on the test problem considered here. For SW, we found that ordering a composite from $\kappa(t)$ that first undergoes mutation yields better performance. Preliminary experiments revealed a per-bit mutation rate of $p_\kappa = 0.05$, which will also be used here, to be promising. This modification encourages exploration in case the EA gets stuck at

[12]The purpose of Line 27 and 28 of Algorithm 5.4 is to permit the submission and arrival of composite orders also during time steps where null submissions are submitted.

Algorithm 5.4 Generational EA with online resource-purchasing strategies

Require: *commCompERC*(...) (a commitment composite ERC), f (objective function), T (time limit), C (budget), $c_{\text{time\_step}}$ (cost per time step), μ (parent population size), λ (offspring population size), #Strategy (selected resource purchasing strategy)

1: $t = 0, c = 0$ (global variables representing the current time step and costs), $Pop = \emptyset$ (current population), $OffPop = \emptyset$ (offspring population)
2: **while** $|Pop| < \mu \ \wedge \ t < T \ \wedge \ c < C$ **do**
3: generate solution $\vec{x}$ at random
4: $\vec{x} = \text{functionWrapper}(\vec{x}, t)$
5: $Pop = Pop \cup \{\vec{x}\}$; t++
6: **while** $t < T \ \wedge \ c < C$ **do**
7: $OffPop = \emptyset$
8: **repeat**
9: generate two offspring $\vec{x}^{(1)}$ and $\vec{x}^{(2)}$ by selecting two parents from Pop, and then recombining and mutating them
10: $OffPop = OffPop \cup \{\vec{x}^{(1)}\} \cup \{\vec{x}^{(2)}\}$
11: **until** $|OffPop| = \lambda$
12: **if** $Z \neq \emptyset \ \wedge$ (#Strategy=JIT $\vee$ #Strategy=JITR) **then**
13: useUpOldComposites($OffPop$)
14: **if** #Strategy=JITR **then**
15: performRepairing($OffPop$)
16: **if** #Strategy=JIT $\vee$ #Strategy=JITR) **then**
17: alignOrdersAndSolutions($OffPop$)
18: **for** $i = 0$ to $|OffPop|$ **do**
19: **if** $t < T \ \wedge \ c < C$ **then**
20: $\vec{x}_i = \text{functionWrapper}(\vec{x}_i, t)$ // $\vec{x}_i$ represents the ith solution of $OffPop$
21: t++; c+=$c_{\text{time\_step}}$
22: form new Pop by selecting the best μ solutions from the union population $Pop \cup OffPop$

23: functionWrapper($\vec{x}, t$){ // Notice that Line 32 to 36 can only be entered by the SW strategy
24: $y_t = \text{null}$
25: **if** $\vec{x}$ is a null solution **then**
26: $\vec{x}_t = \vec{x}$;
27: do the same task as in Line 4 of Algorithm 4.5
28: do the same task as in Line 5 of Algorithm 4.5
29: **else**
30: **if** $\vec{x}$ satisfies the ERC *commCompERC*(...) **then**
31: $\vec{x}_t = \vec{x}$; $y_t = f(\vec{x}_t)$
32: **else**
33: **if** $|\kappa(t)| = WS$ **then**
34: remove the element of $\kappa(t)$ that has been added earliest
35: $\kappa(t) \cup \vec{x}_{comp}$ // $\vec{x}_{\text{comp}}$ denotes the composite required by $\vec{x}$
36: $\vec{x}_t$=repair($\vec{x}, t$); $y_t = f(\vec{x}_t)$
37: return $\vec{x}_t$ and y_t}

suboptimal composites. It is likely that the appropriate mutation rate depends on the string length l of solutions, and the number of composite-defining bits; a more thorough investigation is needed to confirm this.

Table 5.5: Parameter settings of online resource-purchasing strategies.

Strategy	*Parameter*	*Setting*
JITR	Number of shifting rounds *mIter*	500
	Weighting factor w	$0.1 \times \max(0, 4 - \lfloor c/250 \rfloor)$, where c is the current cost counter
SW	Per-bit mutation probability for composites from $\kappa(t)$, p_κ	0.05

5.3.3 Experimental results

Our focus in this study is to analyze the impact of commitment composition ERCs in the presence of budgetary constraints rather than time limitations. We thus set the time limit T to an arbitrarily large number and terminate the optimization upon reaching some budget C.

Figure 5.19 shows the probability of SW and JIT of achieving the population average fitness of our base algorithm obtained in an ERC-free environment. For SW the performance impact depends crucially on the number of storage cells $\#SC$ and the reuse number RN. From the left plot we observe that the performance improves the more storage cells are available and the lower the reuse number is. The reason therefore is that for these settings we order a larger number of potentially different composites more frequently. This in turn reduces the probability of needing to repair and if in case it needs to be repaired it increases the diversity of the repaired solutions among the composite-defining bits.

From the right plot of Figure 5.19 we observe that the performance of JIT improves as the time lag decreases and the reuse number increases. As this strategy does not repair and composites are gratis in this experiment, any positive effect on the performance comes from a shorter waiting period between transitions of population generations. While a shorter time lag has a direct positive effect on the waiting period, a greater reuse number allows an optimizer to evaluate more solutions from a new population using old composites and thus to compensate a slightly larger time lag. An increase in the number of storage cells $\#SC$, results not shown here, can have a positive performance effect because one is more likely to have a required composite in storage and thus can reduce the waiting time between population generations.

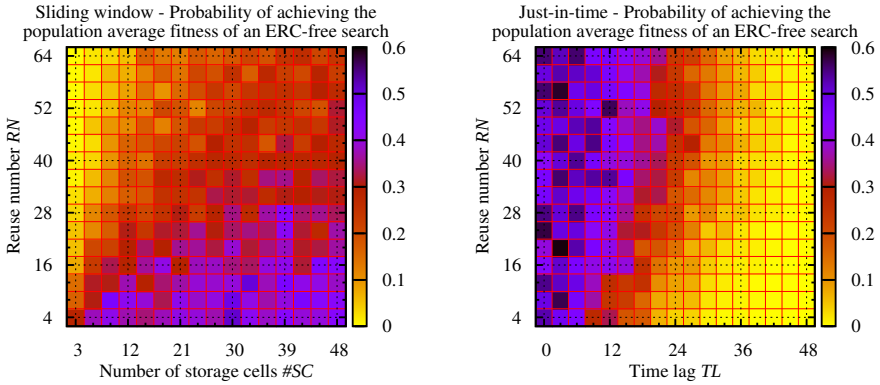

Figure 5.19: Plots showing the probability of SW (left) and JIT (right) of achieving the population average fitness of our base algorithm obtained in an ERC-free environment given a budget and time limit of $C = T = 1500$. For SW this probability is shown as a function of $\#SC$ and RN for the ERC $commCompERC(o(H_{\#}) = 30, \#SC, TL = 10, RN, SL = RN)$, and for JIT it is shown as a function of TL and RN for the ERC $commCompERC(o(H_{\#}) = 10, \#SC = 10, TL, RN, SL = RN)$; cost were set to $c_{\text{order}} = 0$, $c_{\text{time\_step}} = 1$, $C = 1500$.

The performance of JITR obtained at large budgets is not significantly different from the performance of JIT as a function of TL and RN. This is due to the rather conservative cooling scheme of w. Let us now analyze whether there is a performance difference between the two strategies when we vary the budget and associate composite orders with costs $c_{\text{order}} > 0$. Figure 5.20 shows pairwise performance comparisons of the three strategies as a function of the the cost counter c and the cost parameter c_{order}. From the top left plot it is apparent that JITR is able to locate fit solutions at a lower budget than required by JIT (see region $0 < c \leq 600$, $0 \leq c_{\text{time\_step}} \leq 0.5$). That is, repairing yields a more economical search and should be preferred in situations where the number of evaluation steps is limited due to a low budget and/or high costs.

From the top right plot of Figure 5.20 we observe that the weakness of JIT of being too expensive or wasteful at low budgets and/or high costs is also apparent when comparing it with SW. The bottom plot of Figure 5.20 indicates that repairing improves performance significantly in these constraint setting regimes, though the performance of SW cannot be matched. The reason that SW performs so well in the presence of small budgets is that it avoids waiting for composite orders to arrive. However, this is achieved at the cost of many repairs, which, ultimately, may cause a shift in the search direction towards suboptimal composites. An indication that such shifts in the search direction take place is the fact

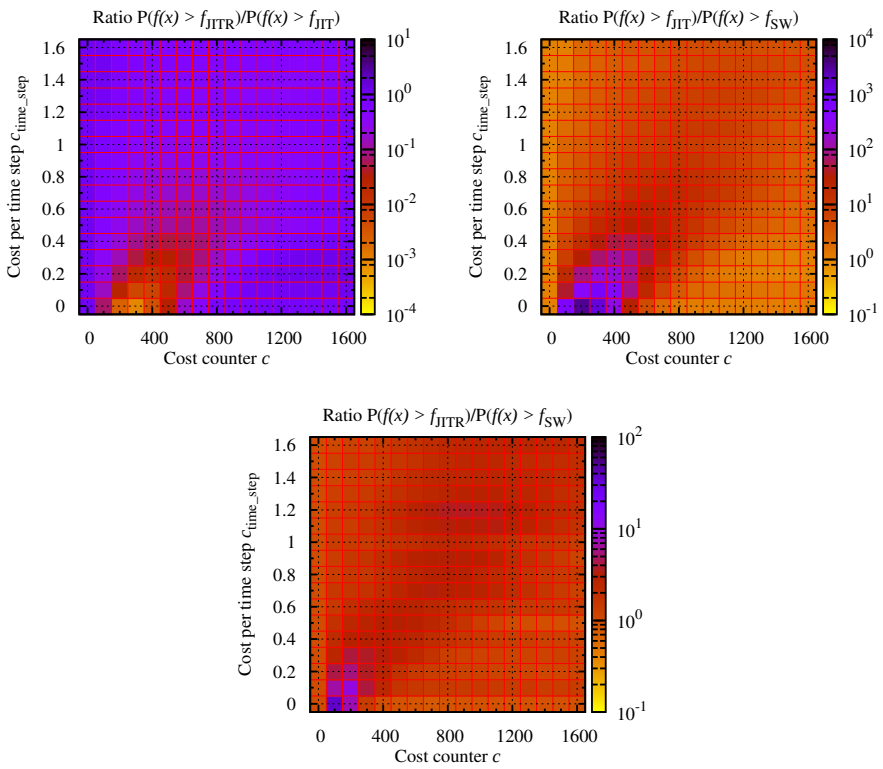

Figure 5.20: Plots showing the ratio $\mathrm{P}(f(x) > f_{\mathrm{JIT}})/\mathrm{P}(f(x) > f_{\mathrm{SW}})$ (left) and $\mathrm{P}(f(x) > f_{\mathrm{JITR}})/\mathrm{P}(f(x) > f_{\mathrm{JIT}})$ (right) as a function of c and $c_{\mathrm{time\_step}}$ for the ERC $commCompERC(o(H_{\#}) = 10, \#SC = 5, TL = 5, RN = 30, SL = 30)$, $c_{\mathrm{order}} = 1$; here, x is a random variable that represents solutions drawn uniformly at random from the search space and f_* the population average fitness obtained with strategy $*$. Here, x is a random variable that represents solutions drawn uniformly at random from the search space and f_* the population average fitness obtained with policy $*$. If $\mathrm{P}(f(x) > f_*)/\mathrm{P}(f(x) > f_{**}) > 1$, then strategy $**$ is able to achieve a higher average best solution fitness than strategy $*$ and a greater advantage of $**$ is indicated by a darker shading in the heat maps; similarly, if $\mathrm{P}(f(x) > f_*)/\mathrm{P}(f(x) > f_{**}) < 1$, then $*$ is better than $**$ and a lighter shading indicates a greater advantage of $*$.

that JIT and JITR can match, and sometimes outperform SW for larger budgets (see region $1000 < c \leq 1600$, $0 \leq c_{\mathrm{time\_step}} \leq 0.5$ in the top right and bottom plot).

The previous experiment looked at composite commitment ERCs featuring a rather small number of storage cells SC, which is in favour of SW. We also used a rather high order $o(H_{\#}) = 30$ in the previous experiment. For SW, a high order $o(H_{\#})$ tends to reduce the risk of having identical composites on storage and thus benefits the diversity in the population with respect to the composite-defining solution bits. Let us now investigate whether the performance advantage

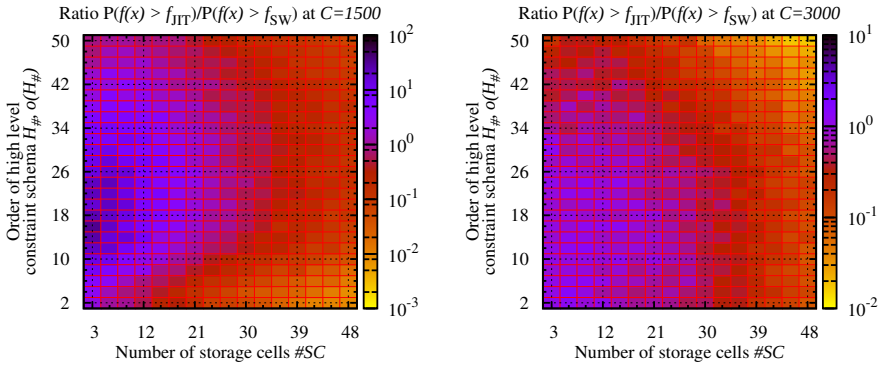

Figure 5.21: Plots showing is the ratio $\mathrm{P}(f(x) > f_{\mathrm{JIT}})/\mathrm{P}(f(x) > f_{\mathrm{SW}})$ for a budget of $C = 1500$ (left) and $C = 3000$ (right) as a function of the number of $\#SC$ and $o(H_{\#})$ for the ERC $commCompERC(o(H_{\#}), \#SC, TL{=}25, RN{=}25, SL{=}25)$, $c_{\mathrm{order}} = c_{\mathrm{time\_step}} = 1$. Here, x is a random variable that represents solutions drawn uniformly at random from the search space and f_* the population average fitness obtained with policy $*$. If $\mathrm{P}(f(x) > f_*)/\mathrm{P}(f(x) > f_{**}) > 1$, then strategy $**$ is able to achieve a higher average best solution fitness than strategy $*$ and a greater advantage of $**$ is indicated by a darker shading in the heat maps; similarly, if $\mathrm{P}(f(x) > f_*)/\mathrm{P}(f(x) > f_{**}) < 1$, then $*$ is better than $**$ and a lighter shading indicates a greater advantage of $*$.

of SW over the other two strategies can still be preserved when we vary $\#SC$ and $o(H_{\#})$. Figure 5.21 compares the performance between JIT and SW as a function of $\#SC$ and $o(H_{\#})$ for a budget of $C = 1500$ (left plot) and $C = 3000$ (right plot); JITR achieved a similar performance to JIT for the considered budget regimes. For the lower budget (left plot), we observe that SW outperforms JIT for rather large orders $o(H_{\#})$ and few storage cells $\#SC$ (see range $8 < o(H_{\#}) < 40$, $\#SC < 15$); SW is too wasteful in the presence of more storage cells. As the budget increases (right plot), JIT is able to match the performance of SW and eventually to overtake it, particularly if both $\#SC$ and $o(H_{\#})$ are large.

5.3.4 Summary and conclusion

In this study we have considered an optimization scenario in which costly and expendable resources (composites) are required in the evaluation of solutions, and these composites had to be ordered in advance, kept in capacity-limited storage, and used within a certain time frame. We proposed three online resource-purchasing strategies (see Section 5.3.1) and analyzed them over a number of resource-constraint settings. A simple just-in-time (JIT) strategy, which can be seen as the minimal strategy, has shown to be generally effective but too expensive at early optimization stages. This drawback has been eliminated by intelligently

repairing solutions during these stages (JITR). Besides these two reactive strategies we also considered a sliding window (SW) strategy, which applied some from of anticipation. This (simple) strategy performs better than JITR and JIT for some constraint settings, although it is costly in the presence of much storage space.

We can try out various modifications to the purchasing strategies to improve their performance. For instance, for the SW strategy one may ensure that distinct composites only are ordered from the sliding window, and also avoid having all storage cells filled at any time. Furthermore, rather than ordering the composites that were added to $\kappa(t)$ most recently, one may order composites that have been requested most frequently, or have proved to be successful in the past. In essence, a sophisticated ordering strategy may trade off exploration of rarely-requested composites (from the sliding window) with exploitation of composites that have proved to be successful and/or were requested frequently. Alternative repairing strategies for the JIT strategy may also prove fruitful.

5.4 Chapter summary

In this chapter we proposed and analyzed a variety of strategies for coping with ERCs. We first looked at static strategies for dealing with non-evaluable solutions arising due to ERCs. Experiments over a number of test functions suggest that the appropriate strategy depends on the constraint parameters of the ERC rather than the fitness landscape it is optimized over. If this turns out to be generally true, then this means we can select a strategy offline given the common situation that the ERCs are known beforehand. Motivated by this observation we then showed that learning offline when to switch between the static strategies during an optimization process can yield better performance than the static strategies themselves. Finally, we proposed several online resource-purchasing strategies for coping with an ERC type that leaves the arrangement of resources to the optimizer. Overall we can tentatively say that the most suitable constraint-handling strategy depends on the constraint settings of the ERCs considered.

The next chapter focuses on the two remaining resourcing issues considered in this thesis: optimization subject to changes of variables and lethal environment.

Chapter 6

Further Resourcing Issues in Closed-Loop Optimization

In this chapter we propose and analyze strategies for coping with two further resourcing issues in closed-loop optimization: problems subject to changes of variables and lethal environments. The first resourcing issue, optimization subject to changes of variables, is motivated by an experimental problem involving the identification of effective drug combinations drawn from a non-static drug library. The second resourcing issue, optimization in lethal environments, causes the population of an EA to decrease in size (permanently) upon evaluating a poor or lethal solution. We motivate this scenario, and devise and validate guidelines for tuning EAs to cope with it.

6.1 Optimization on problems subject to changes of variables

Optimization problems subject to changes of variables belong to the class of dynamic optimization problems, which we introduced in Section 3.4. The dynamic component is related to the fitness landscape rather than the constraints (as was the case with ephemeral resource-constrained problems). The next section motivates optimization problems subject to changes of variables, and in Section 6.1.2 we propose a method, which we call fair mutation, specifically designed for coping with such problems. The test problems we use to compare fair mutation against four standard strategies from dynamic optimization are described Section 6.1.3.

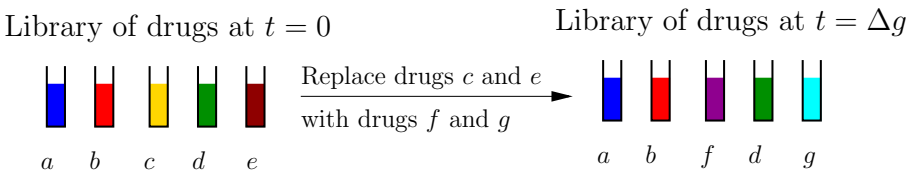

Figure 6.1: An illustration of the effect of changing variables. In the (drug discovery) scenario considered, this corresponds to replacing a number of drugs (here $\#v = 2$) of a library with new ones (every Δg generations during an optimization process).

The experimental setup is outlined in Section 6.1.4, and an empirical study follows in Section 6.1.5. Section 6.1.6 concludes this study.

6.1.1 Motivation

Our motivation for considering optimization problems subject to changes of variables is a completed experimental study (Small et al., 2011), in which we were involved, concerned with the identification of effective drug combinations using EAs. For a detailed description of the experimental setup please refer to Small et al. (2011); here we focus solely on the effect of changing variables. In this study, a human experimentalist needs to arrange a library (of size l) of promising or relevant drugs to be considered by an optimizer. In our case, each drug i corresponds to a single binary variable x_i indicating whether the drug is included into a drug mixture ($x_i = 1$) or not ($x_i = 0$); the drug mixture itself represents an entire solution $\vec{x}$. In applications of this type, the experimental equipment is often set up such that multiple experiments (which are here the mixing of drugs and subsequent testing of the mixtures) can be done in parallel. In our scenario, a robot is taking care of the mixing of drugs, and this robot is able to mix up to 50 combinations in parallel. During the running experimental optimization, which may take several months, the human experimentalist may decide to replace drugs from the library with new ones (as indicated in Figure 6.1) because, for instance, the old drugs have an undesired effect on the efficiency of a drug cocktail or are simply not of interest anymore.[1] Hence, whenever a change of drugs takes place, the corresponding variables and their effects on the fitness are changed as well. Note, a change of drugs does not change the effectivity of cocktails that did not use any of the replaced drugs; i.e. solutions with a 0-bit at any of the replaced

[1]The situation where drugs are only added to or removed from an existing library is realistic too but will not be considered in this study.

variables retain their fitness. Obvious ways of dealing with this are to carry on regardless, or to restart optimization entirely. We wish to see if other strategies fare any better.

Similar to other dynamic optimization problems, problems subject to changes of variables feature a changing fitness function and thus a changing fitness landscape. An example of a conceptually similar dynamic optimization problem is the dynamic traveling salesman (DTSP) or vehicle routing problem (Larsen, 2000), where cities may be replaced with new ones during the optimization. The two main differences between our dynamic combinatorial problem and these existing dynamic problems are that: (i) we do not need to detect changes in the landscape because it is always known when variables are changed (sometimes the exact generation of a change can even be controlled), and (ii) the fitness value remains unchanged for a known subset of solutions (see above). The fact that in our case solutions are evaluated by conducting physical experiments, whose number is limited by resource restrictions, can be considered as another distinction to previous analyses of dynamic problems. The presence of resource limitations requires a quick tracking of new optima. Fortunately, thanks to (i) and (ii), fulfilling this task is usually easier in our scenario than, say, in DTSP.

6.1.2 Fair mutation

In Section 3.4 we introduced three main approaches for tracking optima in a dynamic optimization environment: diversity control, memory-based, and multi-population approaches. The strategy we propose here for dealing with changes of variables, which we call *fair mutation*, can be regarded as a diversity-control approach. Unlike for the majority of strategies of this type, we do not need to answer the question of how to detect environmental changes. Instead, the question is rather how to exploit best the (often expensively obtained and partially intact) information contained in the current population in the generation of a new population after exchanging some variables. Fair mutation introduces diversity into the population only in the initial generations following an environmental change. This is done by using effectively the information stored in the current population and by making use of the fact that only bits with value 1 have potentially an effect. We explain the strategy in detail in the following.

Algorithm 6.1 Fair mutation

Require: p_r (raised mutation rate), Δh (number of generations for which p_r is used)

1: applyFairMutation($Pop, \lambda, V, \Delta t$){
2: **if** $\Delta t = 0$ **then**
3: **for** each variable $k \in V$ **do**
4: **for** $1 \leq i \leq \lfloor \lambda / |V| \rfloor$ **do**
5: generate an offspring $\vec{x}_{off}$ as normal (if the application of selection and variation operators yields two offspring, then select one of them at random); mutate $\vec{x}_{off}$ using a rate of p_r; set variable k of $\vec{x}_{off}$ to 1 and add $\vec{x}_{off}$ to *offPop*; i.e. *offPop* = *offPop* $\cup \vec{x}_{off}$
6: in the situation where $|offPop| < \lambda$, fill *offPop* in the same way as above but instead of setting variable k to 1, set to 1 a variable selected from V at random without replacement
7: **else**
8: **if** $0 < \Delta t \leq \Delta h$ **then**
9: generate offspring population as normal but for any offspring containing a 1-bit at any of the variables in V, mutate the other bits using p_r
10: **else**
11: generate offspring population as normal
12: return *offPop*}

6.1.2.1 Algorithm

The idea of fair mutation is to allow an optimizer both to continue optimization as normal and at the same time to rapidly explore the space of solutions that *use* any of the new variables. In our context the new variables represent a set of new drugs that replaces a set of existing drugs in the drug library. We employ two mechanisms to facilitate the exploration of the search space spanned by the new variables: (i) allowing an optimizer to test each new variable (i.e. its bit being set to 1) within the same number of offspring solutions, and (ii) enabling an optimizer to explore solutions with a 1-bit at any of the new variables by applying a raised mutation rate p_r for Δh generations (p_r and Δh are user-defined parameters). Algorithm 6.1 describes the method *applyFairMutation*($Pop, \lambda, V, \Delta t$), which we call to generate the offspring population *OffPop* at each generation. The method takes as input the current population *Pop*, the number of offspring solutions λ to be generated, the set of changed variables V, and the number of generations Δt passed after the last environmental change (or variable replacement step). We assume that fair mutation is augmented on a generational EA with a $(\mu + \lambda)$-ES reproduction scheme.[2]

From Line 8 and 9 in Algorithm 6.1 one can see that, in the first Δh generations after an environmental change, we apply a raised mutation rate (from which

[2]For a steady state EA or $\lambda = 1$, we would call the method *applyFairMutation*($Pop, 1, V, \Delta t$) multiple times thereby ensuring that each variable is tested an equal number of times.

1-bits among the new variables are protected) if an offspring has one or more 1-bits among any of the new variables. Although this step allows an optimizer to perform exploration among solutions that *use* one or more of the new variables, it also runs the risk that an offspring may be perturbed so much that it turns into a poor one. However, when we embed this strategy within an (elitist) population update scheme it achieves both the continued normal optimization of strings that do not contain any of the new variables, and the fair and rapid exploration of each of the subspaces (schemata) that contain a new variable. Alternatively, one may decrease the mutation rate p_r during the period of Δh generations, or look at other parameter controlling techniques (see Section 3.1.2) for tuning p_r.

6.1.3 Testing Environment

To validate the performance of fair mutation we use a *main-and-joint effects (MJ)* problem, which models the real drug mixture problem we are interested in. We present the *MJ* model in the following section. We also consider variable changes on *NK* landscapes as a comparison.

6.1.3.1 *MJ* model

In the application described in (Small et al., 2011), the experience of the human experimentalist and a previously performed pilot study suggested that only some drugs (bits) of the library have a positive effect while others have a negative side effect or no effect at all. Moreover, it is believed that interactions among the considered drugs can be described by main and joint effects only with higher order interaction effects being negligible.

Let M denote the number of randomly selected bits $x_{h(q)}, q = 1, \ldots, M \leq l$ (l is the total number of bits) which are assigned a main effect; $h(q)$ is the variable's index of the qth main effect. And let J denote the number of randomly selected *distinctive* bit pairs $(x_{i(r)}, x_{j(r)}), r = 1, \ldots, J, i(r) \neq j(r)$, which are assigned a joint effect; $i(r)$ and $j(r)$ are the indexes of the two variables that have the rth joint effect. In our case, the strengths of all main effects $m_{h(q)}$ and joint effects $g_{i(r),j(r)}$ are drawn uniformly from the interval $[-3, 1]$. That is, on average, a quarter of all main and joint effects will be positive; negative effects can model undesired side effects associated with (pairs of) drugs. As the purpose of this

model is to model the real drug mixture problem at hand, we assume that variables or drugs have only an effect if they are included in a drug mixture. Hence, the strength of a main effect $m_{h(q)}$ or joint effect $g_{i(r),j(r)}$ is only considered if the variable $x_{h(q)}$, respectively, both variables $x_{i(r)}$ and $x_{j(r)}$ are 1-bits. To evaluate a candidate solution, we look up all the main and joint effects which are *on* and take the average of the sum of all these values. In other words, the fitness function f to be maximized is defined as

$$f(\vec{x}) = \frac{1}{l}\left(\sum_{q=1}^{M} m_{h(q)} \cdot x_{h(q)} + \sum_{r=1}^{J} g_{i(r),j(r)} \cdot x_{i(r)} \cdot x_{j(r)}\right) . \tag{6.1}$$

6.1.3.2 *NK* landscapes

The second test problem we consider are *NK* landscapes, or, equivalently, *NK*α landscapes with the setting $\alpha = 0$. Please refer to Section 5.1.2.2 for a description of the test problem. Compared to an *MJ* model, an *NK* landscape assigns a positive effect (drawn here from $[0, 1]$) to each of the possible 2^{K+1} bit-wise neighborhood configurations; i.e. 0 and 1-bits are treated equally. For fair mutation this means that (i) offspring solutions devoted to a new variable need to be set to 1 and to 0 in equal number (this concerns the *for*-loop in Line 4 of Algorithm 6.1), and (ii) new variables are always protected from mutation (this concerns the *if*-clause in Line 8 of Algorithm 6.1).

6.1.3.3 Realizing a change of variables

On an *NK* landscape, a change of a variable involves reselecting its K neighbors and reinitializing the corresponding 2^{K+1} effects (of the neighborhood configurations). Similarly, on a *MJ* model, if the replaced variables had a main and/or joint effect, then these effects are reinitialized and the joint bits reselected. Recall that we select the joint bits such that we end up with J distinctive bit pairs only. In this study, a change of variables is performed every Δg generations whereby always $\#v$ randomly selected variables are changed. That is, while Δg controls the frequency of environmental changes, $\#v$ controls the severity of a change.

6.1.4 Experimental setup

We compare the performance of fair mutation (denoted as #Strategy=1 in Algorithm 6.2) on dynamic *NK* landscapes and *MJ* models against four standard strategies from dynamic optimization: triggered hypermutation (#Strategy=2), restart of the optimization (#Strategy=3), re-evaluating the current population after changing variables and then carrying on as normal (#Strategy=4), and a niching algorithm (#Strategy=5). We will also show partial results of a strategy called random immigrants (#Strategy=6).

We introduced the idea of triggered hypermutation (Cobb, 1990) and random immigrant (Grefenstette, 1992) in Section 3.4. Here, we will outline the working procedures of the two diversity-controlled approaches in more detail. The triggered hypermutation approach is typically employed within a generational EA, with crossover and a small per-bit mutation probability $p_{m,b}$ (also referred to as *base mutation rate*). The method monitors the quality of the fitness of the best individual in the population over time, and when this measure declines, it is assumed that the environment has changed and the mutation rate is increased drastically (by some factor HR) for a single generation. The obvious drawback of this approach is that changes in the landscape occurring in regions that are not populated by an EA will not trigger the hypermutation. The random immigrants approach circumvents this drawback by promoting exploration of the search space continuously. In fact, every generation, it replaces a percentage p_{RI} of the population by randomly generated individuals.

As the niching algorithm, we use deterministic niching, whose procedure we already described in Algorithm 3.2 of Section 3.4.

Parameter settings: We augment all dynamic optimization strategies (except the niching algorithm) on an EA with a (μ+λ)-ES reproduction scheme; the niching algorithm is elitist in its own way as offspring solutions are allowed to replace their parents only if they are fitter. The EA is identical to the one described in Section 5.1.2.1 using also the same setup (see Table 5.1). Algorithm 6.2 shows how we integrate the different dynamic optimization strategies as well as variable changes (performed in Line 9) into this EA.

The parameter settings of the different dynamic optimization strategies is given in Table 6.1. Note from the table that the triggered hypermutation is set up such that, once triggered, we use a mutation rate of $p_{m,b} \times HR = 0.5$ for a single generation. The effect of mutating a solution with this probability is

Algorithm 6.2 Generational EA with dynamic optimization strategies for handling problems subject to changes of variables

Require: f (objective function), G (maximal number of generations), μ (parent population size), λ (offspring population size), #Strategy (number of selected dynamic optimization technique)

1: $g = 0$ (generation counter), $\Delta t = 0$ (number of generations past after the last variable change), $MA = -1$ (moving average of the best-of-generation fitness), $Pop = \emptyset$ (current population), $OffPop = \emptyset$ (offspring population)
2: **while** $|Pop| < \mu \wedge g < G$ **do**
3: generate solution $\vec{x}$ at random
4: evaluate $\vec{x}$ using f
5: $Pop = Pop \cup \{\vec{x}\}$; g++; update MA
6: **while** $g < G$ **do**
7: $OffPop = \emptyset$
8: **if** $g \bmod \Delta g = 0$ **then**
9: select $\#v$ variables to be replaced at random, and add their indices to the set of changed variables, V; reinitialize the fitness effects of the new variables in case the replaced variables were previously associated with any (i.e. generate a new objective function f); set $\Delta t = 0$
10: **if** #Strategy=3 **then**
11: reinitialize Pop with random solutions
12: **else**
13: re-evaluate all solutions in Pop using (the new objective function) f
14: **if** #Strategy=1 **then**
15: $OffPop =$ applyFairMutation$(Pop, \lambda, V, \Delta t)$
16: **else**
17: **if** #Strategy=2 $\wedge$ MA declined **then**
18: generate λ solutions at random to form the population $OffPop$
19: **if** #Strategy=5 **then**
20: create new population Pop according to Algorithm 3.2 of Section 3.4
21: **else**
22: **for** $i = 0$ to $\lambda/2$ **do**
23: generate two offspring $\vec{x}^{(1)}$ and $\vec{x}^{(2)}$ by selecting two parents from Pop, and then recombining and mutating them
24: $OffPop = OffPop \cup \{\vec{x}^{(1)}\} \cup \{\vec{x}^{(2)}\}$
25: **if** #Strategy=6 **then**
26: replace $p_{RI} \times \lambda$ solutions of $OffPop$ with random solutions
27: **if** #Strategy$\neq$5 **then**
28: evaluate all solutions of $OffPop$ using f
29: update MA based on the best solution fitness in $OffPop$
30: form new Pop by selecting the best μ solutions from the union population $Pop \cup OffPop$
31: g++; Δt++

identical to generating a solution at random. Hypermutation is triggered when there is a decrease in the moving average of the best-of-generation fitness over five generations. These are standard parameter setting combinations for this strategy, and so is the setting of the replacement rate p_{RI} for the random immigrants

Table 6.1: Parameter settings of dynamic optimization techniques.

Strategy	*Parameter*	*Setting*
Fair mutation	Raised mutation rate p_r	$1/l$
	Number of generations for which p_r is applied, Δh	0
Triggered Hypermutation	Base mutation rate $p_{m,b}$	0.001
	Hypermutation rate HR	500
Random Immigrants	Replacement rate p_{RI}	0.3

approach. Note, triggered hypermutation is typically employed within a non-elitist EA. We integrate it into an elitist EA because the performance obtained with that EA proved to be better on the test problems considered.

We are mainly interested in the dynamics resulting from changing a small number of variables frequently as this is the scenario we are faced with in (Small et al., 2011). On this class of problems, the parameter setting for which fair mutation performed best is $\Delta h = 0$ and $p_r = 1/l$, which will also be used here (see Table 6.1). The interpretation of this parameter setting is that new variables are protected from mutation only at the generation at which an environmental change occurs, and that the raised mutation rate is the same as the standard mutation rate (i.e. $1/l$). Higher values of Δh and p_r are required for more severe environmental changes and different interval ranges of main and joint effects. At the generation of an environmental change, all strategies (apart from the restart strategy, see Line 10 and 11 of Algorithm 6.2) re-evaluate the current population before they generate an offspring population.

Our aim is to understand what effect changes of variables have on evolutionary search for different settings of $\#v$ and Δg. For ease of analysis we therefore fix the problem parameters of the NK landscape and MJ model to $N = l = 30$ and $K = 2$, respectively, $M = 30$ and $J = 30$. That is, in the MJ model all bits have a main effect and 30 bit pairs have a joint effect. The search space size l corresponds also to the size of the drug library considered in our experimental study (see Small et al. (2011)). Table 6.2 summarizes the test problem settings.

Any results shown are average results across 100 independent algorithm runs.

Performance measurement: To measure the performance of the investigated

Table 6.2: Parameter settings of test functions f modelling problems subject to changes of variables.

Test function	*Parameter*	*Setting*
MJ model	Solution parameters l	30
	Number of main effects M	30
	Number of joint effects J	30
NK landscapes	Solution parameters $N = l$	30
	Neighborhood size K	2

strategies we use three measurements: *accuracy*, *adaptability*, and *average best-of-generation fitness*. Accuracy and adaptability have been proposed by Trojanowski and Michalewicz (1999) and rate an algorithm's overall performance with a single number. Both metrics are offline performance measures (De Jong, 1975) and rely on the assumption that the time steps at which the environment changes are known beforehand. The accuracy value represents the fitness difference between the best solution in the population at the generation just before a change and the optimal fitness in that environment, averaged over the entire run. Let S denote the total number of environmental changes during an optimization process, and as before Δg the number of generations between successive environmental changes. We can define the accuracy metric as

$$Acc = \frac{1}{S} \sum_{i=1}^{S} \Delta e_{i,\Delta g-1} \ , \tag{6.2}$$

where $\Delta e_{i,j}$ is the fitness difference between the best solution at the jth generation after the last environmental change ($j \in [0, \Delta g - 1]$), and the optimal fitness of the landscape after the ith environmental change ($i \in [0, S - 1]$).

The adaptability value represents the fitness difference between the best fitness of each generation and the optimal fitness in the current environment, averaged over the entire run. We can formally define this metric as

$$Ada = \frac{1}{S} \sum_{i=1}^{S} \left(\frac{1}{\Delta g} \sum_{j=0}^{\Delta g-1} \Delta e_{i,j} \right) . \tag{6.3}$$

For both metrics the optimal solution in an environment is approximated by using the best solution found from running an elitist generational EA for 100

generations, 100 times independently. It is easy to see that an algorithm is better the smaller its accuracy and adaptability measurements are. As Acc decreases the performance of an algorithm in tracking the optima within the Δg generations improves; a value of $Acc = 0$ means that the algorithm was always able to find the optimum within Δg generations. A small value of Ada indicates that the best solution fitness found during an optimization run was always close to the optimal fitness; a value of $Ada = 0$ means that the optimal fitness has never been lost.

The average best-of-generation fitness is a more standard per-generation measure, suitable for plotting.

6.1.5 Experimental results

Figure 6.2 to 6.7 show the best-of-generation fitness of all dynamic optimization strategies (except random immigrants) on MJ models and NK landscapes for three different settings of $\#v$ and Δg: $\#v = 2$ and $\Delta g = 10$ (Figure 6.2 and 6.3), $\#v = 8$ and $\Delta g = 25$ (Figure 6.4 and 6.5), and $\#v = 15$ and $\Delta g = 40$ (Figure 6.6 and 6.7). The accuracy and adaptability measurements of all strategies (including random immigrants) on these problem instances are shown in Table 6.3. For comparison reasons, the table includes also the results obtained by triggered hypermutation using a base mutation rate of $p_{m,b} = 1/l = 1/30$ and a hypermutation rate of $HR = 15$. This base mutation rate is identical to the per-bit mutation rate p_m employed by the other dynamic optimization strategies; the hypermutation rate is again set such that hypermutated solutions are equivalent to random solutions.

For each of the three problem instances it is apparent that all strategies perform better on the MJ model than on the NK landscape. This is due to the higher degree of variable interactions featured by the NK landscape, which tends to make the problem more difficult to solve. Let us analyze the performance of the strategies on each of the problem instances in turn.

Figure 6.2 and 6.3 investigate the situation where the environment changes only little but frequently. This is the most realistic and for us most interesting case. Here we observe that:

- Environmental changes are easier to detect on NK landscapes (Figure 6.3) than on the MJ model (Figure 6.2) because all variables have an effect regardless of whether they are 0 or 1-bits (as opposed to 1-bits only on the

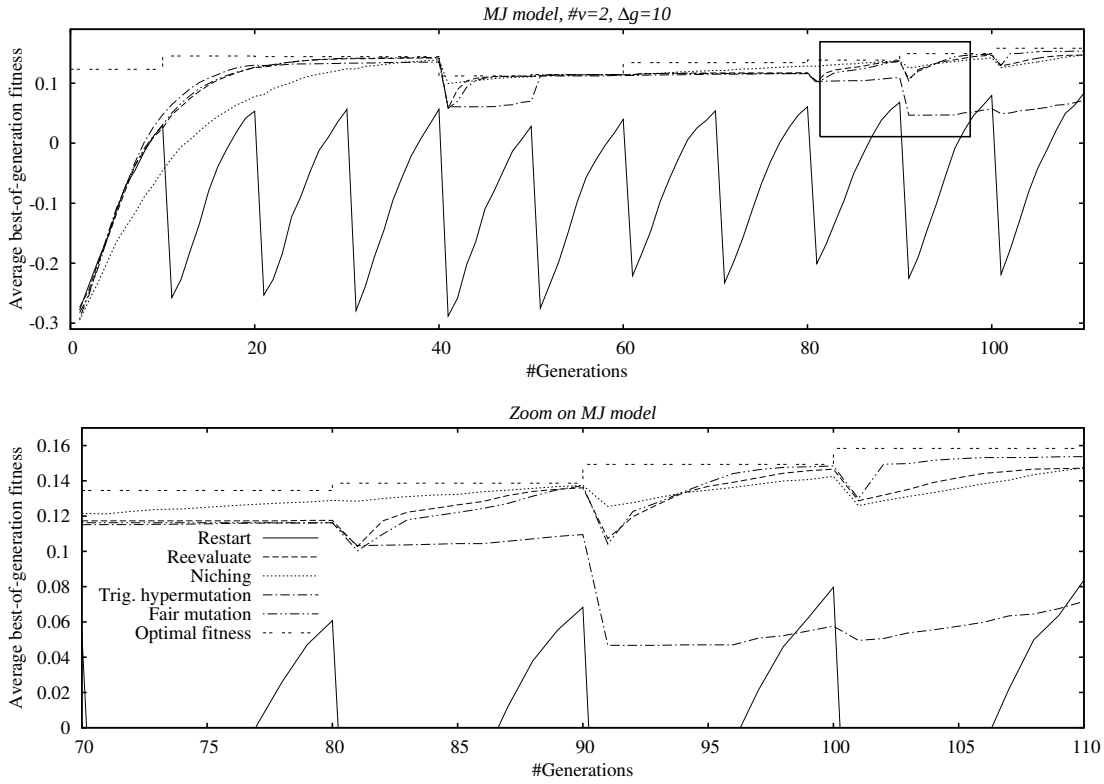

Figure 6.2: Plots showing the average best-of-generation fitness obtained by various dynamic optimization strategies on a *MJ* model with $\#v = 2, \Delta g = 10$. The bottom plot zooms into the area indicated by the box in the top plot.

MJ model).

- Using a restart approach or triggered hypermutation results in a poor performance, especially on *NK* landscapes (Figure 6.3). In the case of the restart strategy, the reasons are the low frequency and severity of environmental changes; there is simply insufficient time for the EA to converge to a good population state before the environment changes again. The poor performance of triggered hypermutation is due to a similar reason: while environmental changes are reliably detected on this problem, the low base mutation rate does not allow for a sufficiently quick convergence to fit regions in the search space. From the accuracy and adaptability measures in Table 6.3 we observe that a higher base mutation rate improves performance significantly.

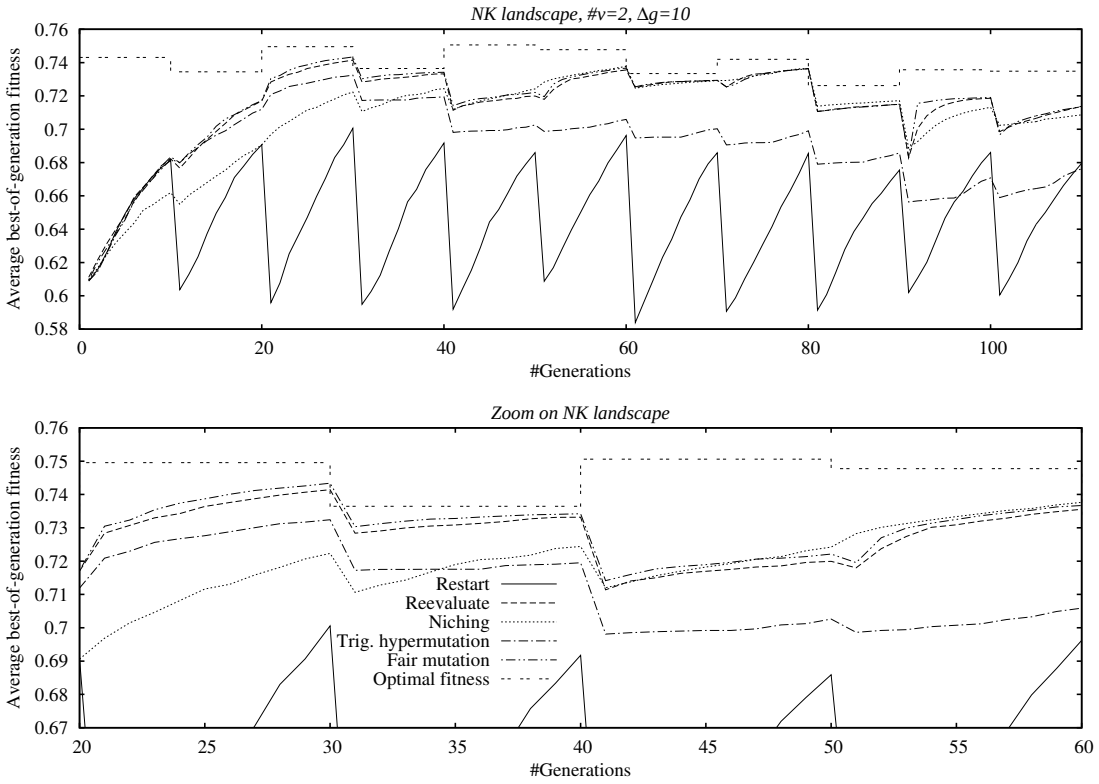

Figure 6.3: Plots showing the average best-of-generation fitness obtained by various dynamic optimization strategies on *NK* landscapes with $\#v = 2, \Delta g = 10$. The bottom plot zooms into the area indicated by the box in the top plot.

- The other three strategies yield similar performances though a slight advantage is apparent for fair mutation, which is closely followed by the reevaluation strategy; this is also confirmed by the accuracy and adaptability measurements. Fair mutation seems to recover significantly faster than the other strategies when there is a severe change in the landscape; see generation 90 and 100 on the *NK* landscape (Figure 6.3) and *MJ* model (Figure 6.2), respectively.

- The niching algorithm maintains a relatively high population diversity throughout the search as it searches within several promising niches simultaneously. This slows down the convergence, as can be seen at the beginning of the optimization, but it also increases the probability of finding new optima quicker after undergoing an environmental change; see e.g. generation 50 and 80 on the *NK* landscape (Figure 6.3) and *MJ* model (Figure 6.2), respectively.

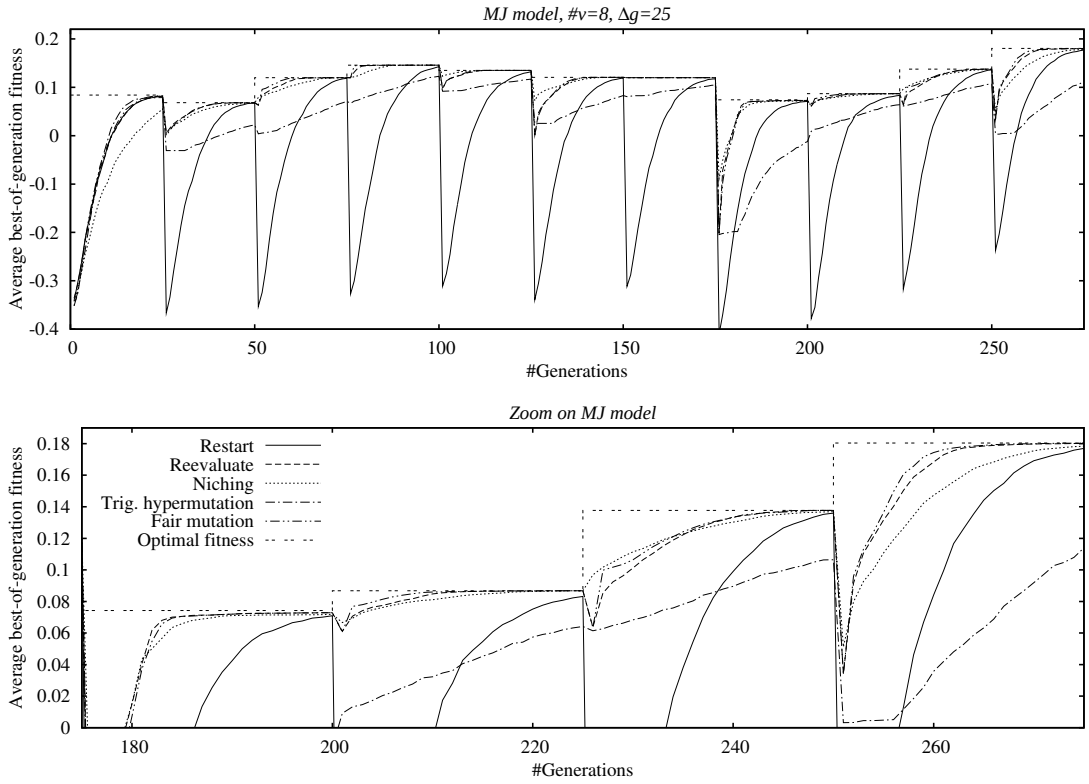

Figure 6.4: Plots showing the average best-of-generation fitness obtained by various dynamic optimization strategies on a *MJ* model with $\#v = 8, \Delta g = 25$. The bottom plot zooms into the area indicated by the box in the top plot.

From Figure 6.4 and 6.5, where more variables are changed less frequently we observe that:

- Changing a larger number of variables alters the fitness landscape more severely and thus simplifies detecting environmental changes.
- The restart approach can sometimes outperform the other strategies on the *NK* landscape (Figure 6.5). This is, for example, the case at generation 50, where the landscape seems to undergo a relatively severe change.
- Fair mutation outperforms the other strategies again slightly. The fact that only a small number of offspring solutions can be devoted to each new variable reduces its ability to explore the search region spanned by the new variables effectively.
- The deficit in the convergence speed of the niching algorithm is now more apparent. This is because a more severe landscape change reduces the probability that any of the currently occupied niches is close to the new optimal

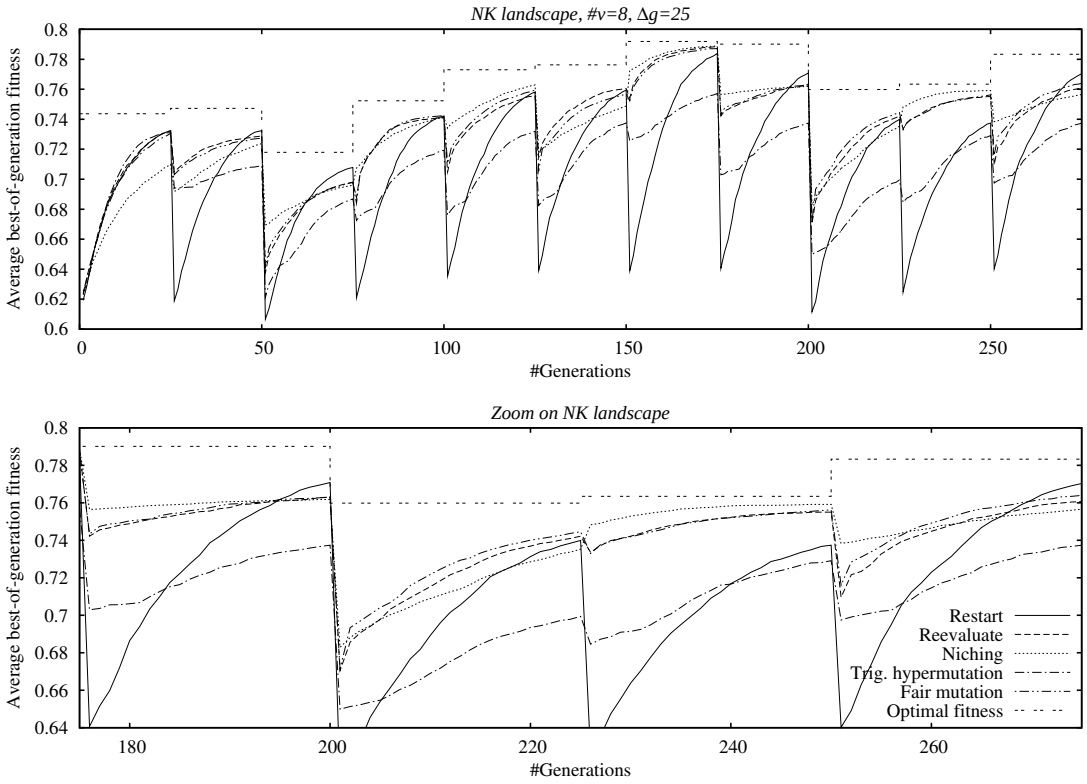

Figure 6.5: Plots showing the average best-of-generation fitness obtained by various dynamic optimization strategies on *NK* landscapes with $\#v = 8, \Delta g = 25$. The bottom plot zooms into the area indicated by the box in the top plot.

search region; i.e. the population needs to shift its search focus more severely and this takes time when trying to maintain several niches simultaneously.

Figure 6.6 and 6.7 consider the case where half of all solution variables are replaced with new ones every $\Delta g = 40$ generation. This setting may simulate the case where a human experimentalist wants to test many different drugs without being able (e.g. due to storage or budget limitations) or wanting to change the drug library size. From Figure 6.7 and the accuracy and adaptability measurements in Table 6.3 it is apparent that a restart approach significantly outperforms the other strategies on the *NK* landscape. On the *MJ* model (Figure 6.6), a restart approach yields best results in terms of the accuracy metric with fair mutation being best in terms of the adaptability metric. That is, a restart approach is able to find, on average, a better solution fitness than fair mutation within the Δg generations available between successive environmental changes. Fair mutation, on the other hand, tends to be on average closer to the optimal fitness at all times than the restart approach; this is due to the quicker convergence of fair

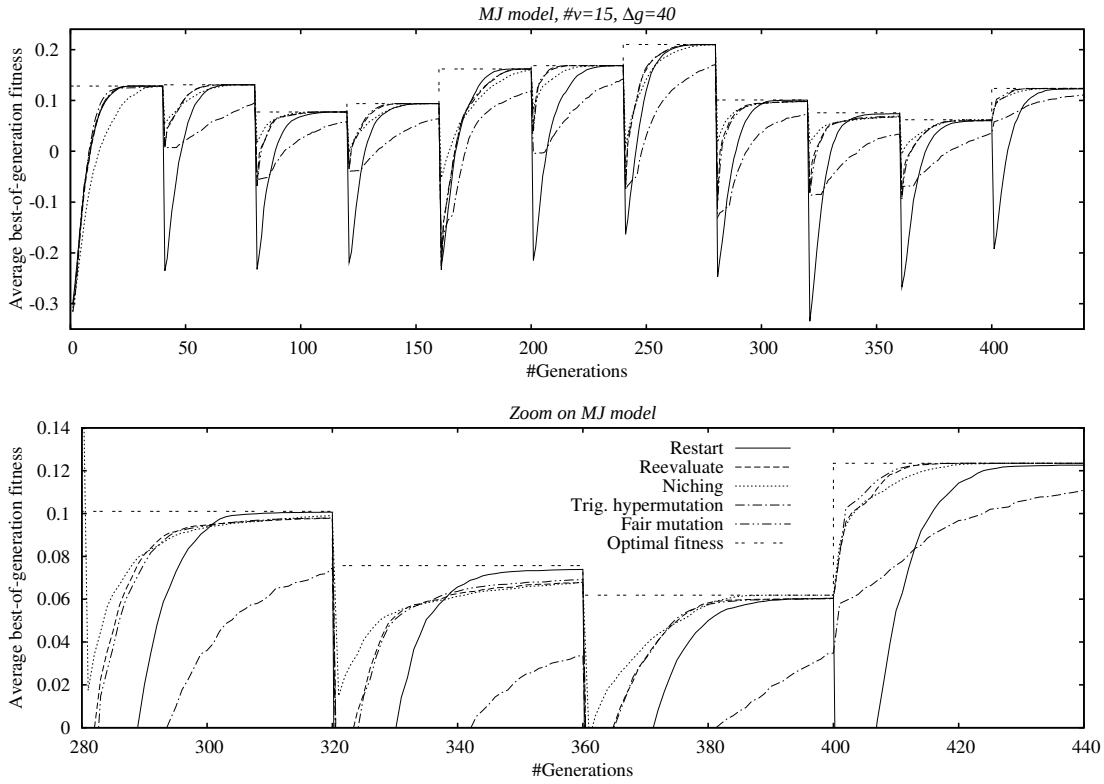

Figure 6.6: Plots showing the average best-of-generation fitness obtained by various dynamic optimization strategies on a *MJ* model with $\#v = 15, \Delta g = 40$. The bottom plot zooms into the area indicated by the box in the top plot.

mutation at the initial phase upon an environmental change (see bottom plot of Figure 6.6). The very small number of offspring solutions devoted to each new variable reduces the *local search* abilities of fair mutation considerably, and this translates into long-term performance effects. The niching algorithm performs worst on the *NK* landscape. The reason is again its deficit in the convergence speed combined with the severity of the environmental changes. On the *NK* landscape, triggered hypermutation performs sometimes quite well, losing often only to the restart approach (see e.g. generation 200, 250, and 440 in Figure 6.7).

6.1.6 Summary and conclusion

In this study we have compared different dynamic strategies for enabling a generational EA to cope with problems that are subject to changes of variables, motivated by a real problem in drug discovery. We proposed a strategy, which we call fair mutation, specifically designed for dealing with such problems, and compared it against five standard strategies applied in dynamic optimization.

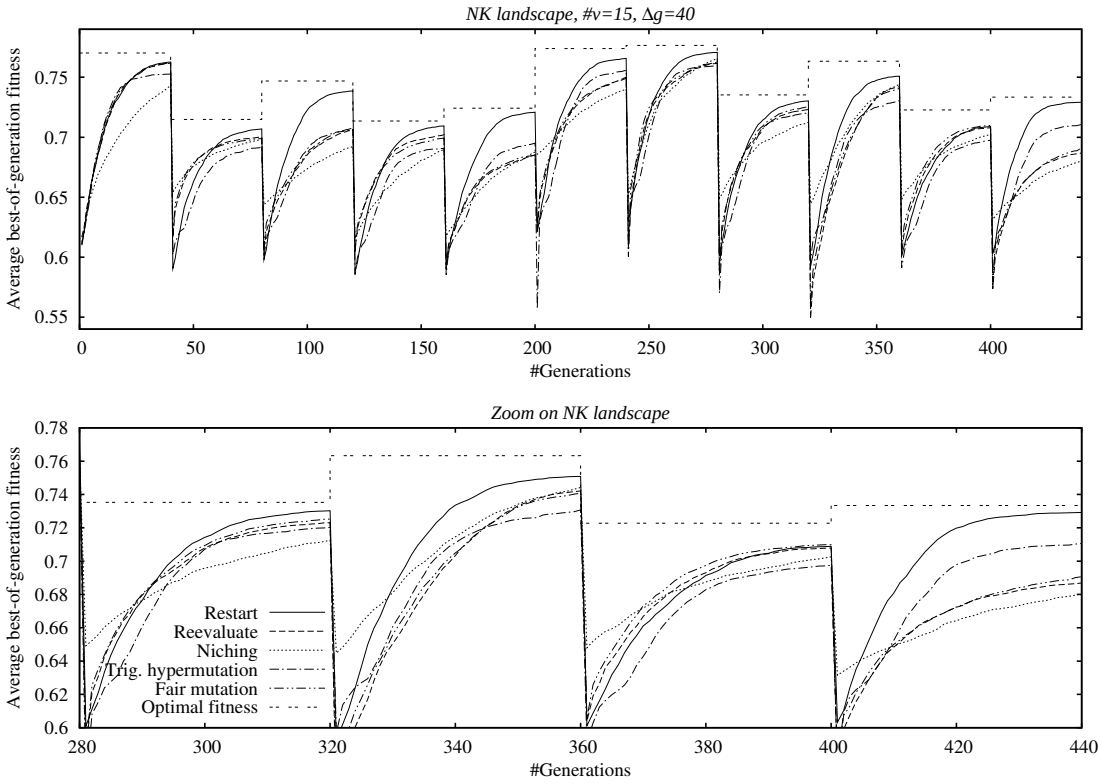

Figure 6.7: Plots showing the average best-of-generation fitness obtained by various dynamic optimization strategies on *NK* landscapes with $\#v = 15, \Delta g = 40$. The bottom plot zooms into the area indicated by the box in the top plot.

The results have shown that only little additional diversity if at all needs to be introduced into the population when changing a few variables frequently, or optimizing a landscape with a low degree of variable interactions. Here, very good results were also obtained using a niching algorithm or a standard elitist EA that simply re-evaluates the population after a variable change and then carries on as normal. When changing many variables (around 10 or more), particularly on a quite epistatic fitness landscape, restarting the optimization from scratch has shown to be often the best choice.

Future work should investigate how to exploit further the correlation existing between the changed variables and the parts of the landscape undergoing a change, featured by optimization problems subject to changes of variables. If the optimizer were allowed to decide the point of time (generation) at which variables are exchanged, then optimizing this aspect might be worth considering in the design of strategies too; this kind of scenario has been considered by Olhofer et al. (2001) for a shape design optimization problem but more work is needed to fully

Table 6.3: Accuracy and adaptability results of various dynamic optimization strategies on different problem instances of dynamic *NK* landscapes and *MJ* models. For each problem instance and metric, we highlighted all strategies in bold face that are not significantly worse than any other strategy. The statistical test applied here is the non-parametric Kruskal-Wallis test with a significance level of 5% (2-sided).

Dynamic optimization strategy	Met-ric	$\#v = 2, \Delta g = 10$		$\#v = 8, \Delta g = 25$		$\#v = 15, \Delta g = 40$	
		NK	*MJ*	*NK*	*MJ*	*NK*	*MJ*
Fair mutation	*Acc*	**0.0163**	**0.0155**	**0.0150**	**0.0002**	0.0193	0.0011
	Ada	**0.0242**	**0.0344**	**0.0298**	**0.0210**	0.0480	**0.0279**
Restart	*Acc*	0.0520	0.0815	**0.0151**	0.0022	**0.0076**	**0.0003**
	Ada	0.0942	0.2133	0.0552	0.1232	**0.0395**	0.0609
Re-evaluation	*Acc*	**0.0170**	**0.0163**	0.0202	**0.0003**	0.0202	0.0012
	Ada	**0.0252**	**0.0350**	**0.0302**	**0.0220**	0.0516	**0.0282**
Niching	*Acc*	0.0242	0.0290	0.0188	0.0032	0.0264	0.0012
	Ada	0.0321	0.0471	**0.0314**	0.0271	0.0560	**0.0296**
Triggered hyperm. $p_{m,b} = 0.001, HR = 500$	*Acc*	0.0387	0.0313	0.0376	0.0334	0.0241	0.0290
	Ada	0.0492	0.0554	0.0586	0.0778	0.0552	0.0833
Triggered hyperm $p_{m,b} = 1/l, HR = 15$	*Acc*	0.0194	0.0195	**0.0163**	**0.0004**	0.0146	0.0015
	Ada	0.0299	0.0442	0.0360	0.0354	0.0487	0.0444
Random immigrants	*Acc*	0.0214	0.0228	0.0209	0.0011	0.0231	0.0019
	Ada	0.0297	0.0434	0.0357	0.0266	0.0537	0.0337

understand how algorithmic choices affect performance. Furthermore, in closed-loop applications like our scenario, variables might not only be associated with resources (e.g. drugs) but their use may also be subject to resource-constraints: e.g. the use of drugs might be associated with costs, or drugs might have to be bought in batches. Here, a good strategy for dealing with changing variables has to account for both fitness gradients and variable experimental costs. For instance, while a restart approach is likely to be quite expensive under these circumstances, a strategy that aims at wasting very little resources might take too long to find fit solutions. Thus, future research into the development of strategies that perform well in the presence of resource-constraints is needed too.

6.2 Optimization in lethal environments

In this section we consider a resourcing issue related to what we term 'lethal environments'. Unlike problems subject to ephemeral resource constraints and

changes of variables, this resourcing issue yields an optimization problem featuring a static feasible region and fitness landscape. However, the population size may reduce over the course of the optimization as a function of poor (lethal) solutions evaluated. We define a *lethal solution* as one with a fitness below a certain threshold. This setup models certain closed-loop evolution scenarios that may be encountered, for example, when evolving nano-technologies or autonomous robots. The next section motivates this optimization scenario with a specific example. We then look at the tuning of some of the basic EA configuration parameters for coping with lethal environments: degree-of-elitism, population size, selection pressure, mutation mode and strength, and crossover rate. We also vary aspects of the fitness landscape we are optimizing to observe which landscape topologies pose a particular challenge when optimizing in a lethal environment. Section 6.2.2 will describe the setup of this study, and the study itself will be conducted in Section 6.2.3. Finally, Section 6.2.4 concludes this study.

6.2.1 Motivation

We consider the use of EAs for optimization in a setting where the notions of an individual and of a population size, are slightly different from standard, which forces us to reconsider how to configure an EA appropriately. The general setting is closed-loop optimization but our concern is for a particular sort of closed-loop setting where the hardware on which individuals are tested are reconfigurable, destructible and non-replaceable.

Consider the following scenario. We are developing, by evolution, the control software for an autonomous endoscopic robot (similar to Moglia et al. (2007); Ciuti et al. (2010)), which is ultimately intended to be swallowed by a patient in capsule form, and then used within the body for screening, diagnosis and therapeutic procedures. Before robotic capsules are used on humans, however, their reliability and effectiveness typically needs to be first validated through *in vivo* animal trials. Imagine we have available μ prototypes of a robotic capsule, and we can "radio in" new control software to each of these individual capsules and evaluate them. However, our control software may cause a capsule to malfunction in a lethal (for it) way, in which case it is no longer available (and falls out of the evolving population). In such a scenario, our population size for the remainder of the evolution is now at most $\mu - 1$, in terms of how many pieces of software can be tested each generation. If we continue to explore too aggressively, we may

upload other pieces of software that cause the loss of individuals, which will cost evolution in terms of the extent of parallelism (and hence efficiency in time) we enjoy, and also perhaps in the loss of expensive hardware. So whilst we wish to innovate by testing new control policies, this must be balanced carefully against considerations of maintaining a reasonable population size.

This setting may seem far-fetched to some readers. However, as we see evolution being applied more and more at the boundary between a full embodied type of evolution (where the hardware individuals may reproduce or exchange information)[3] and a fully *in silico* simulation-only approach, especially as EAs are taken up in the experimental sciences more and more, it is likely that variants of the above scenario will emerge. It need not be robotic capsules; it could equally be nano-machines or drone planes we are dealing with; it could be robot swarms, or software released on the Internet, or anything that is reconfigurable remotely in some way, and also destructible.

Clearly, the principle of "survival of the fittest" is going to be somewhat dampened by the constraint that we wish all designs to survive. Indeed, no innovation at all will be possible if we enforce the evaluation of only safe designs, given the assumption that any unseen design could be lethal. So, as is usual with evolution, we should expect that some rate of loss of individuals is going to be desirable, but set against this is the fact that they cannot be replaced, and hence population size and parallelism will be compromised. So, the question is, how should one control the exploration/exploitation trade-off in this optimization setup.

In more *natural settings* than we consider here, the notion of robustness and the tendency of evolution to build in mutational robustness for free has been previously studied, especially in the context of evolution on neutral plateaux (see e.g. Bullock (2003); Schonfeld and Ashlock (2004); Schonfeld (2007)). Bullock's work in particular shows that certain in-built biases toward mutational robustness can retard innovation (our primary concern here). He proposed methods that avoid the bias and hence explore neutral plateaux more rapidly. Although related, this work has a different purpose to ours, and also involves different assumptions.

6.2.2 Experimental setup

This section describes the search algorithms for which we investigate the impact of a lethal optimization scenario, and the parameter settings as used in the

[3]For concrete examples of embodied evolution applications please refer to Section 2.2.

subsequent experimental analysis.

6.2.2.1 Search Algorithms

We consider three types of search algorithms in this study: a tournament selection based genetic algorithm (TGA), a modified version of it, which we call RBS (standing for *reproduction of best solutions*), and a population of stochastic hill-climbers (PHC). Similar types of algorithms have been considered in the previously mentioned studies related to mutational robustness.

Before we describe each algorithm, we first set out the procedures common to all of them. These are the generation process of the initial population of candidate solutions, the setting of the fitness threshold below which solutions are deemed lethal, the mutation operators, and the handling of duplicate solutions. Regarding the initialization, our aim is to simulate the scenario where the initial population, whose size we denote by μ_0 (the reason for adding a subscript to μ is described below), consists of evolving entities (e.g. robotic capsules, robot swarms or software) that are of a certain (high or state-of-the-art) quality. We achieve this by first generating a sample set of S random and non-identical candidate solutions, and then selecting the fittest μ_0 solutions from this set to form the initial population. The *lethal fitness threshold* f_{LFT} is set to the fitness value of the qth fittest solution of the sample set. This threshold is kept constant throughout an algorithmic run.

For mutation, we will investigate two modes (similar to Barnett (2001)): (i) *Poisson mutation*, where each bit is flipped independently with a mutation rate of p_m, and (ii) *constant mutation*, where exactly d randomly selected bits are flipped. Poisson mutation is the more commonly used mutation mode of the two and also the mode we have used in this thesis so far. Furthermore, all search algorithms ensure that a solution is not evaluated multiple times. That is, all evaluated solutions are cached and compared against a new solution (offspring) before performing an evaluation; preliminary experimentation showed that avoiding duplication improves the performance significantly in the presence of lethal environments. If a solution has been evaluated previously, then the GA iteratively generates new solutions (offspring solutions) until it generates one that is evaluable, i.e. has not been evaluated previously, or until L trials have passed without success. In the latter case, we reset the mutation rate to $p_m = p_m + 0.5/l$ (l is the total number of bits) and $d = d + 1$ in the case of Poisson mutation and

Algorithm 6.3 Tournament selection based GA (TGA)

Require: f (objective function), G (maximal number of generations), μ_0 (initial parent population size), λ_0 (initial offspring population size), L (maximal number of regeneration trials)

1: $g = 0$ (generation counter), $Pop = \emptyset$ (current population), $OffPop = \emptyset$ (offspring population), $AllEvalSols = \emptyset$ (set of solutions evaluated so far), $trials = 0$ (regeneration trials counter), $counter = 0$ (auxiliary variable indicating the number of solutions evaluated during a generation)
2: initialize Pop and set lethal fitness threshold f_{LFT}; copy all solutions of Pop also to $AllEvalSols$, and set $\mu_g = \mu_0, \lambda_g = \lambda_0$
3: **while** $g < G \ \wedge \ \mu_g > 0$ **do**
4: $OffPop = \emptyset$; $trials = 0$; $counter = 0$
5: set mutation rate to initial value
6: **while** $counter < \lambda_g$ **do**
7: generate two offspring $\vec{x}^{(1)}$ and $\vec{x}^{(2)}$ by selecting two parents from Pop, and then recombining and mutating them
8: **for** $i = 1$ **to** 2 **do**
9: **if** $\vec{x}^{(i)} \notin AllEvalSols \ \wedge \ counter < \lambda_g$ **then**
10: evaluate $\vec{x}^{(i)}$ using f; $counter$++; $trials = 0$
11: $AllEvalSols = AllEvalSols \cup \vec{x}^{(i)}$
12: **if** $f(\vec{x}^{(i)}) \geq f_{\mathrm{LFT}}$ **then**
13: $OffPop = OffPop \cup \vec{x}^{(i)}$ // non-lethal solutions only are added to the offspring population
14: **else** $trials$++
15: **if** $trials = L$ **then**
16: depending on the mutation operator, reset mutation rate to $p_m = p_m + 0.5/l$ (Poisson mutation) or $d = d + 1$ (constant mutation)
17: reset new population sizes to $\mu_g = \lambda_g = |OffPop|$, and form new population Pop by selecting the best μ_g solutions from the union population of $Pop \cup OffPop$
18: g++

constant mutation, respectively; note, due to its deterministic nature, it is more likely that this reset step is applied with constant mutation. For TGA and RBS we set the mutation rates back to their initial values at the beginning of each new generation. For PHC it makes more sense to set the mutation rates back whenever a new hill-climber of the current population is considered for mutation because of the independent search of the hill-climbers. The idea of increasing the mutation strength temporarily is to increase an EA's exploration abilities to allow it to create solutions that have not been evaluated in the past but are close to the search region currently under investigation.

Tournament Selection Based GA (TGA): The algorithm is an EA with a $(\mu + \lambda)$-ES reproduction scheme (see Algorithm 6.3), as we have used in previous studies in this thesis. This elitist scheme performed significantly better than a standard generational reproduction scheme in lethal environments. We equip

the EA with tournament selection with replacement for parental selection (TS shall denote the tournament size) and uniform crossover (p_c shall denote the crossover rate). We explained above the two mutation modes to be investigated later. We set the parent and offspring population size to be identical or $\mu = \lambda$. As the population size may decrease during the optimization process (due to evaluated lethal solutions), in future, we will denote the population size at generation g ($0 \leq g \leq G$) by $\mu_g = \lambda_g$ with $g = G$ being the maximum number of generations. Clearly, the maximum number of generations can only be reached if the population does not "die out" beforehand or $\mu_g = 0$ for $g < G$.

Reproduction of best solutions (RBS): This algorithm is based on TGA and motivated by the *netcrawler* of Barnett (2001). The netcrawler is designed to efficiently solve fitness landscapes featuring neutral networks. It reproduces only a single best solution at each generation using mutation only, and the mutation strength adapts to the size of each neutral network of the landscape. Similar to the netcrawler, RBS reproduces only the best solutions. More precisely, it selects parents for reproduction exclusively and at random from the set of solutions of the current population with the highest fitness. Hence, it can be seen as a tournament selection based GA with $TS = \mu_g$, and where the tournament participants are selected *without replacement*. Apart from the selection procedure, RBS will use the same algorithmic setup as TGA.

Population of Hill-Climbers (PHC): This search algorithm maintains a population of stochastic hill-climbers, which explore the landscape independently of each other. Each hill-climber undergoes mutation, and, if the mutant or offspring is not lethal, it replaces the original solution if it is at least as fit. Algorithm 6.4 shows the pseudocode of PHC. In essence, PHC with constant mutation is identical to running a set of random-mutation hill-climbers (Forrest and Mitchell, 1993) in parallel with the difference that, as in the case of TGA, the mutation strength is temporarily increased if a previously unseen solution cannot be generated within L trials. PHC has similarities to Barnett's netcrawler, and also the *plateau crawler* of Bullock (2003). The working procedure of the plateau crawler closely resembles a population of independent hill-climbers. Similar to the netcrawler, it is has been designed to be immune to selection for mutational robustness when optimizing fitness landscapes featuring neutral networks. The property common to all three approaches (netcrawler, plateau crawler and PHC) is that they guarantee that each parent potentially leaves a copy or a slightly

Algorithm 6.4 Population of hill-climbers (PHC)

Require: f (objective function), G (maximal number of generations), μ_0 (initial parent population size), L (maximal number of regeneration trials)

1: $g = 0$ (generation counter), $Pop = \emptyset$ (current population), $OffPop = \emptyset$ (offspring population), $AllEvalSols = \emptyset$ (set of solutions evaluated so far), $trials = 0$ (regeneration trials counter), $lethal$ = false (auxiliary boolean variable indicating whether a hill-climber is lethal)
2: initialize Pop and set lethal fitness threshold f_{LFT}; copy all solutions of Pop also to $AllEvalSols$, and set $\mu_g = \mu_0$
3: **while** $g < G \ \wedge \ \mu_g > 0$ **do**
4: $OffPop = \emptyset$; $trials = 0$
5: **for** $i = 1$ **to** μ_g **do**
6: set mutation rate to initial value and $lethal$ = false
7: **repeat**
8: mutate solution $\vec{x}_i$ of Pop to obtain mutant $\vec{x}'_i$
9: **if** $\vec{x}'_i \notin AllEvalSols$ **then**
10: evaluate $\vec{x}'_i$ using f; $trials = 0$
11: $AllEvalSols = AllEvalSols \cup \vec{x}'_i$
12: **if** $f(\vec{x}'_i) \geq f_{\mathrm{LFT}}$ **then**
13: **if** $f(\vec{x}'_i) \geq f(\vec{x}_i)$ **then**
14: $OffPop = OffPop \cup \vec{x}'_i$
15: **else** $OffPop = OffPop \cup \vec{x}_i$
16: **else** $trials$++
17: **if** $trials = L$ **then**
18: depending on the mutation operator, reset mutation rate to $p_m = p_m + 0.5/l$ (Poisson mutation) or $d = d + 1$ (constant mutation)
19: **until** $\vec{x}'_i$ is evaluable
20: reset new population size to $\mu_g = |OffPop|$, and form new population Pop by copying all solutions from $OffPop$ to Pop
21: g++

modified version of itself in the population. This ensures that offspring, parents, or other ancestors, are equally likely to be selected as future parents. For fitness landscapes with neutral networks, this feature can cause a population to be unaffected by mutational robustness as it allows the population to spend at each point on a neutral network an equal amount of time (Hughes, 1995; Bullock, 2003).

6.2.2.2 Parameter Settings

The experimental study will investigate different settings of the parameters involved in the three search algorithms (TGA, RBS and PHC). However, if not otherwise stated, we use the default settings as given in Table 6.4 with constant mutation being the default mutation operator. Notice that this setup, especially the setting of p_c, TS, and the mutation mode, is significantly different from previous algorithm setups employed in this thesis. Recall that RBS uses the same

Table 6.4: Default parameter settings of search algorithms.

Algorithm	*Parameter*	*Setting*
TGA, RBS, PHC	Initial parent population size μ_0	30
	Constant mutation rate d	1
	Regeneration trials L	1000
	Number of generations G	200
	Sampling size S	1000
	Solution rank q for lethal fitness threshold setting	250
TGA	Initial offspring population size λ_0	30
	Tournament size TS (selection with replacement is applied)	4
	Crossover rate p_c	0.0
RBS	Initial offspring population size λ_0	30
	Tournament size TS (selection without replacement is applied)	$\mu_g, 0 \leq g \leq G$
	Crossover rate p_c	0.0

default parameter settings as TGA with the difference that reproductive selection is done with a tournament size of $TS = \mu_g, 0 \leq g \leq G$, and tournament participants being chosen without replacement.

To investigate the effect of lethal environments on the performance of the algorithms introduced above we consider $NK\alpha$ landscapes (see Section 5.1.2.2 for a description of this problem). We analyze the impact of lethal solutions using different settings of the neighborhood size K and the model parameter α. Recall that the parameter K allows us to control the degree of epistasis or ruggedness of the landscape (larger values of K represent more epistasis), while the model parameter α allows us to specify how influential some variables may be compared to others (larger values of α assign more influence to a minority of variables). We fix the search space size to $N = l = 50$, a setting we have used and justified previously.

Any results shown are average results across 100 independent algorithm runs. As before, we will use a different randomly generated problem instance for each run but, of course, the same instances for all algorithms. To allow for a fair comparison, we also use the same initial population and (hence) lethal fitness threshold for all algorithms in any particular run.

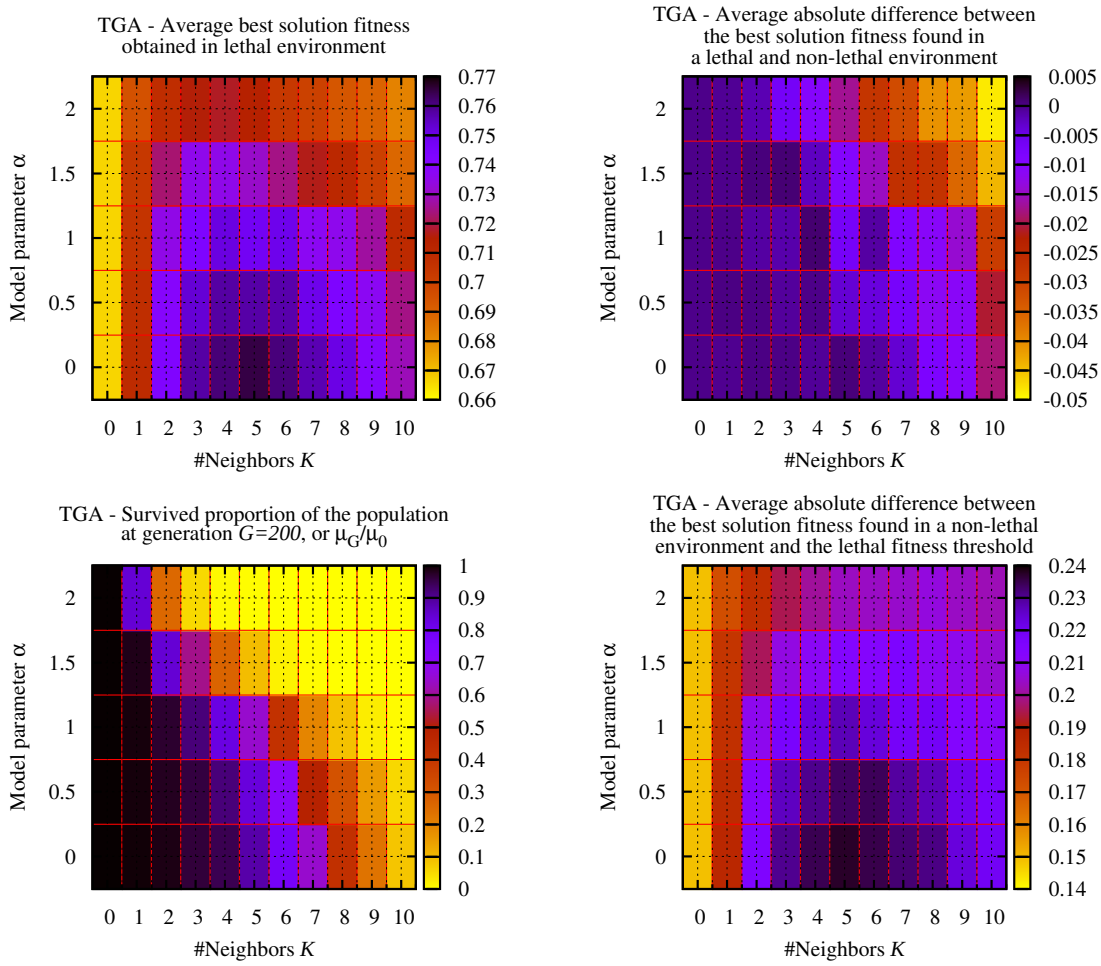

Figure 6.8: Plots showing the average best solution fitness found by TGA (using $TS = 2, p_c = 0.0$ and Poisson mutation with $p_m = 1/N$) in a lethal environment (top left), the average absolute difference between this fitness and the fitness achieved in a non-lethal environment, i.e. $f_{\mathrm{LFT}} = 0$ (top right), the average proportion of the population that survived at generation $G = 200$, or μ_G/μ_0 (bottom left), and the average absolute difference between the best solution fitness found and the lethal fitness threshold in a non-lethal environment (bottom right), on $NK\alpha$ landscapes as a function of the neighborhood size K and the model parameter α.

6.2.3 Experimental results

Let us first analyze the effect of lethal environments on evolutionary search for different topologies of the $NK\alpha$ landscapes. Figure 6.8 shows the effect on the best solution fitness found (top plots) and the (remaining) population size (bottom left plot) for TGA with a rather traditional setup ($TS = 2, p_c = 0.0$ and Poisson mutation with $p_m = 1/l$) as a function of the parameters K and α; the

bottom right plot indicates the scope available for optimization after the initialization of the population. From the top plots we can see that the performance is most affected for fitness landscapes that are rugged (large K) but contain some structure in the sense that some solution bits are more important than others (large α). As is apparent from the bottom left plot, the reason for this pattern is that the population size at the end of the optimization decreases as K and/or α increase, i.e. lethal solutions are here more likely to be encountered. In fact, for large K and α, the optimization terminates on average after only about 25 generations (out of 200); this result is not shown here.

The reason that the population size reduces for large K (bottom left plot of Figure 6.8) is that the fitness landscape becomes more rugged, which increases the risk of evaluating lethal solutions despite them being located close to high-quality solutions. Although an increase in α introduces more structure into the fitness landscape, it has also the effect that solutions which have some of these important bits set incorrectly are more likely to be poor. A plausible alternative explanation for the poor performance at large α may be that the lethal fitness threshold is very close to the best solution fitness that can be found, thus reducing the scope of optimization possible and causing many solutions to be lethal; the bottom right plot rules this explanation out. Regarding the (remaining) population size, a greater selection pressure, i.e. a larger tournament size, can help to maintain a large population size for longer, particularly for large K and small α. On the other hand, being more random in the solution generation process by using, for example, a larger mutation and/or crossover rate, has the opposite effect, i.e. the population size decreases more quickly.

The effect on the (remaining) population size translates into an impact on the average best solution fitness. From Figure 6.9, we can see that, for rugged landscapes with structure (i.e. in the range $K > 5, \alpha > 1$), TGA is outperformed by a GA with a rather traditional setup and a PHC in a non-lethal optimization scenario (right plots) but performs better than both algorithms in a lethal optimization scenario (left plots). In fact, as we will also see later, in the range $K > 5, \alpha > 1$, PHC performs best among all the search algorithms considered in the non-lethal environment but worst in the lethal environment. The poor performance of PHC in this range is due to the fact that the hill-climbers in the population are searching the fitness landscape independently of each other. This may be an advantage in a non-lethal environment because it can help to maintain

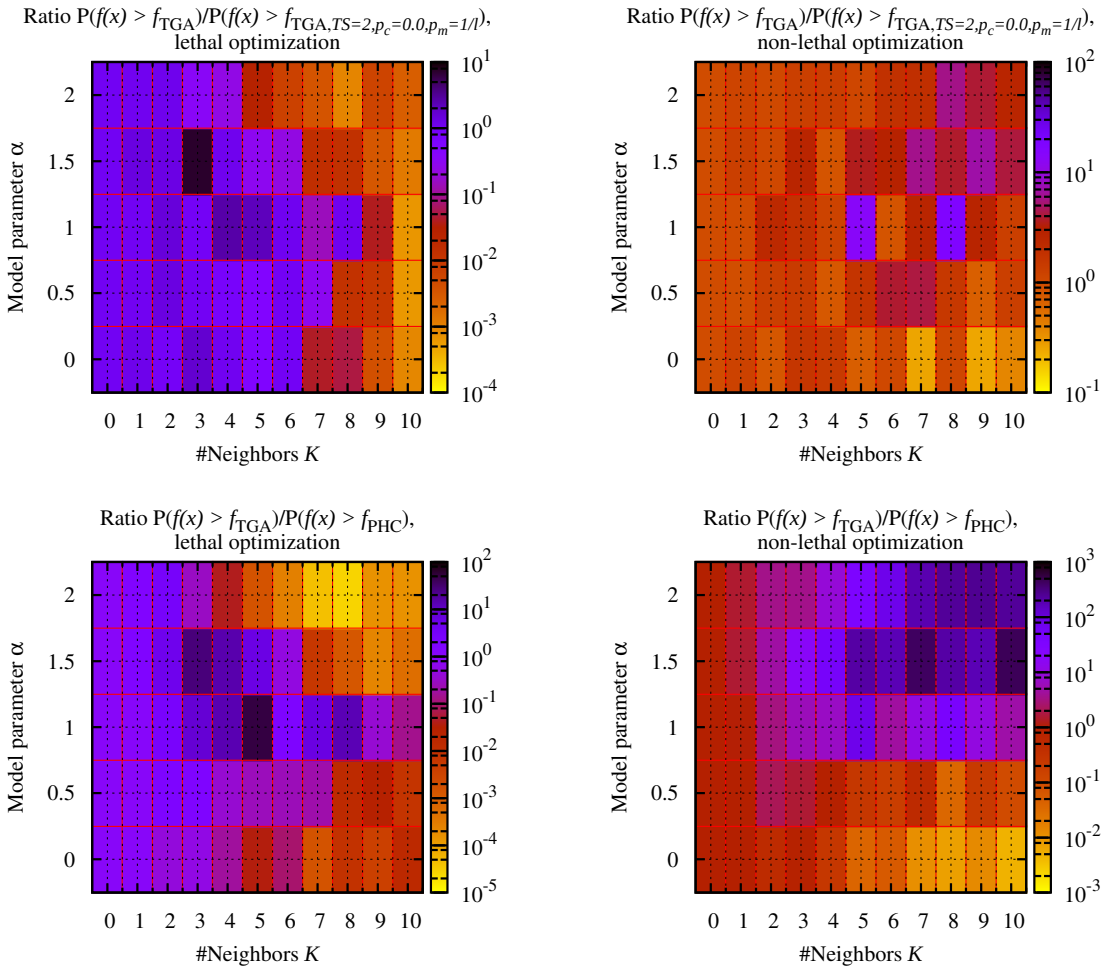

Figure 6.9: Plots showing the ratio $\mathrm{P}(f(x) > f_{\mathrm{TGA}})/\ \mathrm{P}(f(x) > f_{\mathrm{TGA},TS=2,p_c=0.0,p_m=1/l})$ (top plot) and $\mathrm{P}(f(x) > f_{\mathrm{TGA}})/\ \mathrm{P}(f(x) > f_{\mathrm{PHC}})$ (bottom plots) in a lethal (left plots) and non-lethal environment (right plots) on $NK\alpha$ landscapes as a function of K and α. Here, x is a random variable that represents the best solution from a set of solutions drawn uniformly at random from the search space and f_* the average best solution fitness obtained with strategy $*$. If $\mathrm{P}(f(x) > f_*)/\mathrm{P}(f(x) > f_{**}) > 1$, then strategy $**$ is able to achieve a higher average best solution fitness than strategy $*$ and a greater advantage of $**$ is indicated by a darker shading in the heat maps; similarly, if $\mathrm{P}(f(x) > f_*)/\mathrm{P}(f(x) > f_{**}) < 1$, then $*$ is better than $**$ and a lighter shading indicates a greater advantage of $*$.

diversity in the population and prevent premature convergence. However, in a lethal environment, uncontrolled diversity is rather a drawback as it increases the probability of generating solutions with genotypes that are significantly different from the ones in the population, which in turn increases the probability of generating lethal solutions. The less diverse and random optimization of TGA is also the reason that it outperforms a GA with a more traditional setup on rugged

landscapes with structure.

Let us now analyze how the search algorithms fare with different parameter settings. Table 6.5 shows the average best solution fitness obtained by different algorithm setups in a lethal and non-lethal environment (values in parenthesis) on $NK\alpha$ landscapes with $K = 4, \alpha = 2.0$; Table 6.6 shows the same information for $NK\alpha$ landscapes with $K = 10, \alpha = 2.0$. Note that due to the nature of $NK\alpha$ landscapes (particularly due to the absence of plateaus or neutral networks in the landscapes), there was only a single best solution in the population at each generation. For RBS, this has the effect that the performance is independent of the crossover rate because crossover is applied to identical solutions. The two tables largely confirm the observation made above: while a relatively high degree of population diversity and randomness in the solution generation process may be beneficial in a non-lethal environment (which is particularly true for rugged landscape as can be seen from the Table 6.6), it is rather a drawback in a lethal environment as it may increase the risk of generating lethal solutions and subsequently limit the effectiveness of evolutionary search. More precisely, for a lethal environment, the tables indicate that one should use a GA instead of a PHC, and reduce randomness in the solution generation process by avoiding crossover (generally), and using constant mutation as well as a relatively large tournament size. The poor performance of Poisson mutation, which is a typical mutation mode for GAs, is related to situations where an over-averaged number of solution bits is flipped at once. Such variation steps lead to offspring that are located in the search space far away from their non-lethal parents, which can in turn increase the probability of generating lethal solutions.

Finally, Figure 6.10 analyzes whether evolving a large set of entities for a small number of generations yields better performance than evolving only a few entities for many generations; i.e. how is the trade-off between the initial population size μ_0 and the maximum number of generations G. For a non-lethal environment (right plots), we make two observations: (i) best performance is achieved with PHC using a rather small population size of about $\mu_0 = 30$, and (ii) for population sizes $\mu_0 > 50$ one should use a GA whereby the applied tournament size or selection pressure should increase with the population size. As the population size increases, the maximum number of generations G available for optimization is simply insufficient for PHC or a GA with low selection pressure and/or much

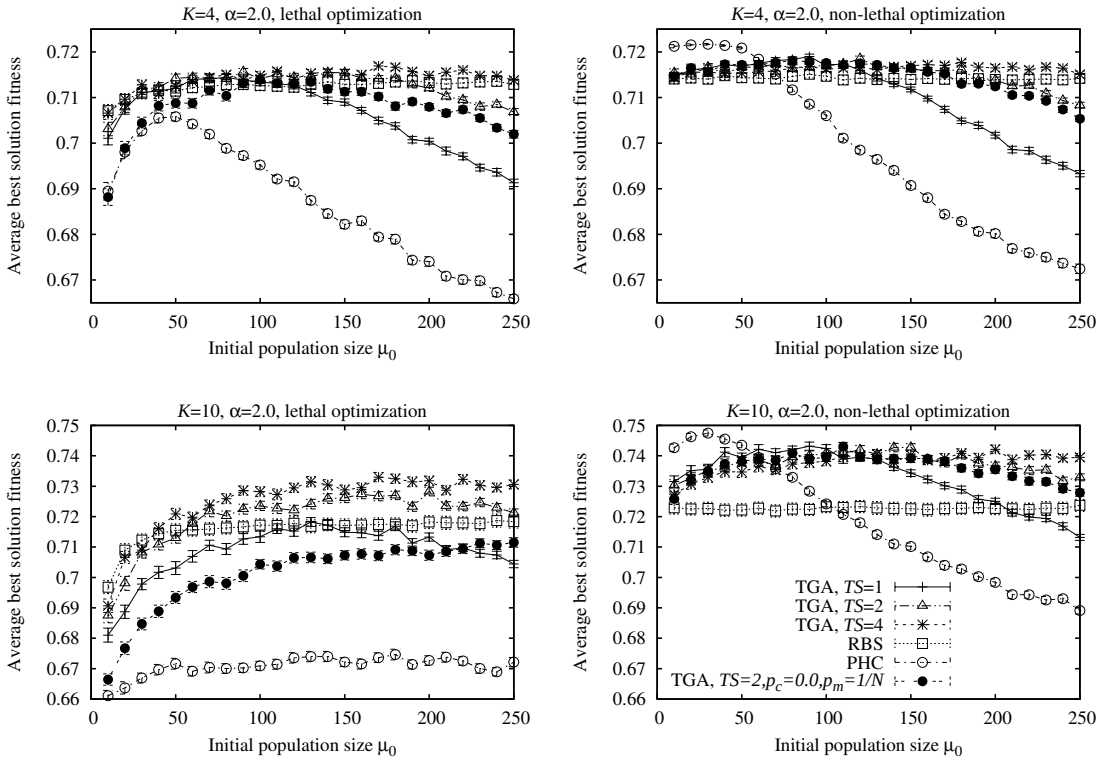

Figure 6.10: Plots showing the average best solution fitness obtained by different search algorithms in a lethal environment (left) and non-lethal environment (right) on $NK\alpha$ landscapes with $K = 4, \alpha = 2.0$ (top) and $K = 10, \alpha = 2.0$ (bottom) as a function of the initial population size μ_0; the maximum number of fitness evaluations available for optimization was fixed at 6000 (corresponding to the default setting $G = 200, \mu_0 = 30$), i.e. $G = \lfloor 6000/\mu_0 \rfloor$. If not otherwise stated, an algorithm used a constant mutation mode with $d = 1$ and no crossover, i.e. $p_c = 0.0$.

randomness in the solution generation process to converge quickly to a high-quality part of the search space. However, on the other hand, a GA with too much selection pressure may cause the population to get stuck at local optima, especially on rugged fitness landscapes (see result of RBS in the bottom right plot). In a lethal optimization scenario (left plots), a GA clearly outperforms PHC (regardless of the population size) due to the reasons mentioned above. Furthermore, unlike to the non-lethal case, we observe that a GA should use an above-average large tournament size already for small population sizes of $\mu_0 > 30$. Similar to the non-lethal case, however, there is a saturation point with RBS, beyond which a further increase in the population size has no effect (see e.g. bottom left plot).

6.2.4 Summary and conclusion

In this study we have conducted an initial investigation of the impact of *lethal* solutions (here modeled as candidate solutions below a certain fitness threshold) on evolutionary search. In essence, the effect of evaluating a lethal solution is that the solution is immediately removed from the population and the population size is reduced by one. This kind of scenario can be found in natural evolution, where the presence of lethal mutants may promote robustness over evolvability, but it is also a characteristic of certain closed-loop evolution applications, for example, in autonomous robots or nano-technologies. When faced with this kind of scenario in optimization, the challenge is to discover innovative and fit candidate solutions without reducing the population too rapidly; i.e. trading off evolvability with robustness.

Our analysis has been focused on the following three main aspects: (i) analyzing the impact of lethal solutions on a standard evolutionary algorithm (EA) framework, (ii) determining challenging fitness landscape topologies in a lethal environment, and (iii) tuning EAs to perform well within a lethal optimization scenario. Generally, the presence of lethal solutions can affect that performance of EAs, but the largest (negative) impact was observed for fitness landscapes that are rugged and possess some structure in the sense that some solution bits are more important than others; in terms of our test functions, which were $NK\alpha$ landscapes, this type of landscape corresponds to large values of K and α. For this fitness landscape topology, we obtained best performance in a non-lethal environment using a small population of stochastic hill-climbers. In a lethal environment, however, significantly better results were obtained using an EA that limits randomness in the solution generation process by employing elitism, a relatively large selection pressure, a constant mutation mode (i.e. flipping exactly one solution bit as opposed to flipping each bit independently with some low probability), and no or a small crossover rate. Also, in a lethal optimization scenario and on the test problems considered, we observed that an EA that evolves a large set of entities (e.g. autonomous robots or software) for a small number of generations performs better than one that evolves a few entities for many generations. The practical implication of this is that, if possible, a larger budget should be allocated for the production of the entities in the first place rather than for attempting to prolong the testing or optimization phase.

As with the other resourcing issues considered in this thesis, there remains

much else to learn about optimization problems subject to lethal environments. The next step could be to design strategies that specifically account for lethal environments (rather than to tune EA operators).

6.3 Chapter summary

In this chapter we have investigated the impact of two resourcing issues on evolutionary search applied to closed-loop optimization: changes of variables, and lethal environments. Problems subject to changes of variables feature a dynamically changing fitness landscapes (but a static feasible region) and thus belong to the class of dynamic optimization problems. However, unlike standard problems in this domain, it is known when changes are applied to the landscape (the point of time may even be controlled), and which part of the search space undergoes a change. We proposed and analyzed a strategy, which we call *fair mutation*, that specifically exploits these two properties. Lethal environments yield optimization problems that limit the number of solutions evaluable below a certain fitness threshold; we refer to these poor solutions as *lethal solutions*. This setup models certain closed-loop evolution scenarios that may be encountered, for example, when hardware (e.g. autonomous robots) on which solutions (e.g a control software) are tested is available in limited quantity only. To cope with lethal environments, we looked at the tuning of some of the basic EA configuration parameters: degree-of-elitism, population size, selection pressure, mutation mode and strength, and crossover rate. The main finding of our study was that lethal environments can be dealt with effectively, even when tuning only some of the basic parameters of EAs, however, the optimal EA configuration depends on the fitness landscape to be optimized.

In the following chapter we will conclude the thesis. We will draw together the findings and their implications, and discuss directions for further research.

Table 6.5: The table shows the average best solution fitness found in a lethal environment after $G = 200$ generations, and in parenthesis, the fitness found in a non-lethal environment, for different algorithm setups on $K = 4, \alpha = 2.0$; the results of *random sampling* were obtained by generating $G \times \mu_0 = 200 \times 30 = 6000$ solutions per run at random and averaging over the best solution fitness values found. The numbers in the subscript indicate the rank of the top five algorithm setups within the respective environment. We highlighted all algorithm setups (among the top five) in bold face that are not significantly worse than any other setup. A Friedman test revealed a significant difference between the search algorithm setups in general, but differences among the individual setups were tested for in a post-hoc analysis using (paired) Wilcoxon tests (significance level of 5%) with Bonferroni correction.

Search algorithm			Constant mutation $d = 1$	Constant mutation $d = 2$	Poisson mutation $p_m = 0.5/N$	Poisson mutation $p_m = 1.0/N$
TGA	$TS = 1$	$p_c = 0.0$	**0.722$_4$** (0.7297$_4$)	0.6998 (0.7286)	0.7174 (0.7272)	0.7141 (0.7285)
		$p_c = 0.25$	0.7188 (0.7272)	0.6969 (0.7296$_5$)	0.7128 (0.7275)	0.7029 (0.7296)
		$p_c = 0.5$	0.713 (0.7279)	0.6901 (0.7287)	0.7091 (0.7276)	0.7078 (0.7277)
	$TS = 2$	$p_c = 0.0$	**0.7223$_3$** (0.7269)	0.7053 (0.7287)	0.7198 (0.7271)	0.7175 (0.7275)
		$p_c = 0.25$	0.7217 (0.7263)	0.7039 (0.7284)	0.7172 (0.7263)	0.7119 (0.7275)
		$p_c = 0.5$	0.7178 (0.7261)	0.7003 (0.728)	0.7176 (0.7268)	0.7135 (0.727)
	$TS = 4$	$p_c = 0.0$	**0.7218$_5$** (0.7266)	0.7107 (0.726)	0.7203 (0.7269)	0.7184 (0.7269)
		$p_c = 0.25$	**0.7238$_1$** (0.7257)	0.7085 (0.7254)	0.7196 (0.7273)	0.7169 (0.7278)
		$p_c = 0.5$	**0.7224$_2$** (0.7251)	0.7101 (0.7256)	0.7172 (0.7263)	0.7167 (0.726)
RBS			0.7179 (0.7226)	0.7129 (0.723)	0.7177 (0.7229)	0.7172 (0.7223)
PHC			0.7117 (**0.7334$_2$**)	0.6843 (0.7271)	0.7033 (**0.7343$_1$**)	0.6912 (**0.7329$_3$**)
Random sampling			0.6197 (0.6358)			

Table 6.6: The table shows the average best solution fitness found in a lethal environment after $G = 200$ generations, and in parenthesis, the fitness found in a non-lethal environment, for different algorithm setups on $K = 10, \alpha = 2.0$; the results of *random sampling* were obtained by generating $G \times \mu_0 = 200 \times 30 = 6000$ solutions per run at random and averaging over the best solution fitness values found. The numbers in the subscript indicate the rank of the top five algorithm setups within the respective environment. We highlighted all algorithm setups (among the top five) in bold face that are not significantly worse than any other setup. A Friedman test revealed a significant difference between the search algorithm setups in general, but differences among the individual setups were tested for in a post-hoc analysis using (paired) Wilcoxon tests (significance level of 5%) with Bonferroni correction.

Search algorithm			Constant mutation $d = 1$	Constant mutation $d = 2$	Poisson mutation $p_m = 0.5/N$	Poisson mutation $p_m = 1/N$
TGA	$TS = 1$	$p_c = 0.0$	0.6918 (0.7317)	0.6637 (0.7383$_{\mathbf{4}}$)	0.6812 (0.7338)	0.6767 (0.7356$_{\mathbf{5}}$)
		$p_c = 0.25$	0.6752 (0.7324)	0.6546 (0.7332)	0.6623 (0.7299)	0.6578 (0.7324)
		$p_c = 0.5$	0.6653 (0.7298)	0.6472 (0.7331)	0.6478 (0.7264)	0.6472 (0.7287)
	$TS = 2$	$p_c = 0.0$	0.702$_{\mathbf{4}}$ (0.7299)	0.6697 (0.7318)	0.6938 (0.7307)	0.6821 (0.7305)
		$p_c = 0.25$	0.6897 (0.7277)	0.662 (0.732)	0.6717 (0.7281)	0.6643 (0.7302)
		$p_c = 0.5$	0.6768 (0.7288)	0.6537 (0.7296)	0.6607 (0.7263)	0.659 (0.7274)
	$TS = 4$	$p_c = 0.0$	**0.7072$_{\mathbf{2}}$** (0.7275)	0.677 (0.7294)	0.6981$_{\mathbf{5}}$ (0.7302)	0.6902 (0.7272)
		$p_c = 0.25$	0.6979 (0.7294)	0.6701 (0.7298)	0.6834 (0.7279)	0.6805 (0.7263)
		$p_c = 0.5$	0.6936 (0.7265)	0.6635 (0.7296)	0.6715 (0.7258)	0.67 (0.7255)
RBS			**0.7102$_{\mathbf{1}}$** (0.7207)	0.6871 (0.7219)	0.7056$_{\mathbf{3}}$ (0.7217)	0.6987 (0.7215)
PHC			0.6651 (**0.7434$_{\mathbf{2}}$**)	0.6476 (0.7345)	0.6568 (**0.7438$_{\mathbf{1}}$**)	0.6517 (**0.7421$_{\mathbf{3}}$**)
Random sampling			0.632 (0.6524)			

Chapter 7

Conclusion

In this thesis we have established an understanding of why and how three resourcing issues — ephemeral resource constraints, changing variables, and lethal environments — affect evolutionary search when applied to closed-loop optimization, and devised search strategies for combating these issues. Generally, we observed that all three resourcing issues may negatively impact the performance of an optimizer, but that the strategies that we devised can be effective against these issues. We will now discuss what we have learnt about each resourcing issue and the practical implications surrounding it.

Optimization subject to ephemeral resource constraints (ERCs): The main resourcing issue we focused on in this thesis (Chapters 4 and 5) were optimization problems subject to *ephemeral resource constraints* (ERCs). These are a type of dynamic constraint that can cause certain solutions to be temporarily non-evaluable during the optimization process. We define solutions as non-evaluable when the resources required in their evaluation are temporarily not available. Clearly, ERCs are less likely to arise if we ensure that all the resources needed to cover all possible search paths are available at any point during an optimization process. However, this strategy is rather costly and inefficient. Instead, we need to develop strategies that interplay with the workings of an evolutionary algorithm (EA) to ensure that promising and relevant resources are available; if a required resource is not available, then a strategy should also be able to efficiently utilize those resources that are available.

Generally, we observed that ERCs may affect the performance of an optimizer in some (but by no means all) cases, and clear patterns emerge that relate ERC parameters to performance effects. For instance, we observed that the later the

constraint time frame of an ERC begins, the less disruptive is the impact on search. This could mean that investment in resources at early stages of the optimization should be preferred, where possible. We also observed that the impact of an ERC on the performance of evolutionary search depends on the order and the quality of the genetic material represented by the constraint schemata of an ERC. Thus, to some degree, we may be able to predict the extent of impact if information about these schemata is available. Consequently, this prediction can serve as the basis for investment decisions related to resources. For an ERC that leaves the arrangement of resources to the optimizer (see Section 5.3), we observed that, at the beginning of an optimization process, it is necessary to limit the number of different resources purchased in the presence of budgetary constraints. This policy is cost-effective and does not impact the performance significantly in the long run.

We have also clearly seen (theoretically and empirically) that the impact on the performance of an evolutionary algorithm (EA) is modulated by the choice of constraint-handling strategy adopted and EA parameters. Which choice of strategy is best is dependent on the details of the ERC, and the time and budgetary constraints present. For summaries of (theoretical and empirical) results relating ERC parameters to suitability of EA configurations and search strategies please refer to Sections 4.3.4, 5.1.5, 5.2.4, and 5.3.4.

Finally, we also observed that knowing about the type of ERC may be sufficient to select an effective search strategy for dealing with it *a priori*, even when knowledge of the fitness landscape is limited. If this observation turns out to be more generally true, then it is good news because we usually have more knowledge about the ERCs than about the fitness landscape. Therefore we would not need to be 'right' about the fitness landscape in order to choose the right search strategy. Nevertheless, as we indicated in the case study of Section 5.1.4, a comprehensive *a priori* analysis of the ERCs at hand can be beneficial when it comes to selecting a suitable policy.

Optimization subject to changes of variables: Motivated by a real closed-loop problem in drug discovery, the second resourcing issue we considered in this thesis (Chapter 6) was concerned with optimizing subject to *changes of variables* (drugs) and thus a changing fitness landscape. The general objective when tackling such dynamic problems is to track the global optimum, which may shift upon a change in the landscape. Quick tracking of global optima is essential

in a closed-loop optimization scenario because evaluations are expensive so their number is typically limited. Clearly, we will not encounter this resourcing issue in the first place if the experimentalist knows in advance all the drugs she wants to test during the optimization process. However, knowing that evolutionary search can cope with changing variables gives an experimentalist the opportunity to interfere in the optimization process if necessary.

Compared to standard dynamic problems, problems subject to changes of variables feature two important advantages for tracking shifting optima quickly: (i) changes in the fitness landscape do not need to be detected explicitly because an optimizer knows and may even be allowed to control when and which variables (drugs) are replaced, and (ii) tracking of optima is simplified because there is a correlation between the variables changed and the parts of the landscape undergoing a change. We proposed a strategy (see Section 6.1.2), which we call *fair mutation*, that specifically exploits these two aspects for tracking shifting optima.

Our analyses showed that the difficulty of tracking optima depends on the number of variables changed and the frequency with which they are changed. Fair mutation has two user-defined parameters that allow the experimentalist to account for these two factors. We have shown in Section 6.1.5 that, when tuned appropriately, fair mutation can track shifting optima more quickly than standard strategies from the dynamic optimization literature. However, as more variables are changed less frequently, restarting the optimization from scratch upon a variable change has often turned out to be the best choice on the test problems considered (see Section 6.1.6).

Optimization in lethal environments: The final resourcing issue we considered in this thesis (Chapter 6) was related to lethal optimization environments. This resourcing issue has an effect on the population size of an optimizer as poor solutions 'die at birth' and are permanently lost from the evolving population. This issue arises because the hardware on which individuals (representing, e.g. control software) are tested is reconfigurable, destructible and non-replaceable. The hardware may be, for instance, prototypes of autonomous robots, drone planes, or nano-machines, all of which are expensive pieces and commonly only available in limit numbers for experimentation. Clearly, the hardware is expensive and we would like to avoid damaging it, but set against this is that we want to improve the control software that runs on the hardware. This conflict requires an optimizer to trade off exploration and exploitation rather carefully.

To cope with lethal environments, we looked (in Section 6.2.3) at the tuning of some of the basic EA configuration parameters: degree-of-elitism, population size, selection pressure, mutation mode and strength, and crossover rate. We also varied aspects of the fitness landscape we are optimizing to observe which landscape topologies pose a particular challenge when optimizing in a lethal environment. The main finding of our investigation was that randomness in the solution generation process should be limited in a lethal optimization environment. That is, elitism and a relatively high selection pressure should be employed and so should be a more deterministic mutation operator; the mutation and crossover rates should be low too. We have also observed that an EA that evolves many individuals (e.g. control software) for a small number of generations performs better than one that evolves a few entities for many generations. The practical implication of this is that, if possible, a larger budget should be allocated for the production of the entities in the first place rather than for attempting to prolong the testing or optimization phase.

7.1 Future work

Although this work allows us to draw valuable conclusions, our study has of course been very limited. There remains much else to learn about the effects of resourcing issues in closed-loop optimization and how to handle them. We now discuss several directions for future research towards achieving this goal.

Gaining a more robust understanding for the search strategies developed. To gain a more robust understanding of the behavior of the search strategies developed, it would be beneficial to consider further and perhaps more realistic fitness landscapes than the ones we employed here. Of course, it would be ideal to validate the search strategies on real-world closed-loop problems featuring real resource constraints. However, this approach is generally not realistic due to the time and/or budgetary requirements. The next best thing we can do is to simulate a fitness landscape based on data obtained from real-world experiments. This is the approach we have taken in the case study of Section 5.1.4, and more studies of this kind are needed.

The test problems considered in this theses were mainly of pseudo-boolean nature; it would also be important to investigate how the search strategies behave for search spaces with real or mixed integer variables. Also, our analysis

has looked at the impact of different resourcing issues isolated from each other, which was necessary to gain an initial understanding of their impact on optimization. However, in real applications it is possible to encounter closed-loop problems that are subject to multiple resourcing issues simultaneously. Hence, further work should investigate how to trade off conflicting properties of different search strategies to cope efficiently with multiple resourcing issues.

Further theoretical analysis of resourcing issues. In Chapter 4 we have used Markov chains to analyze theoretically the effect of a particular ERC type on simple EAs. Although our analysis used a simplified optimization environment (two solution types only), valuable observations were made with respect to the applicability of different selection and reproduction schemes. We also gained some understanding about the impact of ERCs on evolutionary search, which, ultimately, may help us in the design of effective and efficient search strategies for closed-loop optimization. However, our theoretical results were limited in the sense that we did not derive mathematical equations relating, for instance, ERC configurations to optimal EA parameter settings. It remains to be seen whether it is possible to derive such expressions, and how applicable they would be in practice. A number of recent advances in EA theory might present the possibility of understanding ERCs (and other resourcing issues) more deeply, including drift analysis (Auger and Doerr, 2011) and the fitness level method (Chen et al., 2009a; Lehre, 2011).

Understanding the effects of non-homogeneous experimental costs in closed-loop optimization. So far, we have made the assumption that all solution evaluations take equal time or resources. This need not be the case. For instance, when dealing with commitment composite ERCs, it is a very realistic scenario that the composites to be ordered vary in their prices and delivery periods. Under a limited budget, this scenario might cause an optimizer not only to follow fitness gradients but also to account for variable experimental costs. Hence, further work should investigate how to trade off these two aspects effectively. For inspiration, we may look at strategies employed in the Robot Scientist study of King et al. (2004), where, as noted earlier, this scenario has been encountered within an inference problem rather than an optimization problem. Alternatively, we may also cast the problem of trading off fitness with costs as a multi-objective optimization problem and employ multi-objective EAs to tackle this problem.

Once we have gained an understanding of the effect of non-homogeneous costs

on optimization, the next step would be to identify challenging problem instances to investigate whether or not it may be worth developing sophisticated search strategies for dealing with non-homogeneous costs, or whether the same level of performance can be obtained using more straightforward naïve search strategies.

Developing strategies for coping with lethal environments. Our investigation of the impact of lethal environments (see Section 6.2) on evolutionary search was limited to the tuning of simple EA configuration parameters, such as population size, variation operators and their settings. Clearly, it is possible that a better performance may be achieved using strategies specifically designed to cope with lethal environments. From preliminary experiments, we know already that a learning approach, similar to the one proposed in Section 5.2.1, which learns a switching policy offline can yield better performance. Alternatively, we may augment an EA with a strategy that uses assumptions of local fitness correlation to pre-screen the designs and forbid the upload of potential lethals. Such a strategy is similar to brood selection with repair and/or some fitness approximation schemes used in EAs to filter solutions before evaluation (Walters, 1998; Jin and Sendhoff, 2003; Jin, 2005).

Improving and broadening the application of machine learning techniques in closed-loop optimization. We have shown (in Section 5.2) that evolutionary search augmented with machine learning techniques, such as reinforcement learning (RL), can be a powerful optimization tool to cope with ERCs. Similar learning approaches can be applied to other resourcing issues. As mentioned above, we employed a learning-based approach to cope with lethal environments, and the results obtained are promising. However, as mentioned in Section 5.2.3, some tuning of the reinforcement learning agent may be required to achieve this strong performance. The tuning refers to things like parameter settings involved in the RL agent, the state space partitioning, and the reward function. To increase the applicability of learning-based optimizers to different types of optimization problems, one could also try combining offline learning with online learning. For instance, RL can be used to learn offline a policy until some distant point in time, and this policy can then be refined or slightly modified online using the anticipation approach of Bosman (2005). Another avenue worth pursuing is to extend search methods that use surrogate models (see Section 2.6), which are commonly used in real-world problems, to cope with the resourcing issues considered in this thesis.

Casting ERCOPs as RL problems. We indicated in Section 4.1.2 that an ERCOP can be cast as an RL problem rather than a pure optimization problem (perhaps this perspective is more intuitive now that we have seen how RL can be used to learn when to switch between different constraint-handling strategies). When cast as an RL problem, we could use a standard RL technique (such as Sarsa(λ)) to generate solutions to an ERCOP, rather than augment an EA with constraint-handling strategies under RL control. As we know, to make this approach work, we need to define the state space, actions, and the reward function. To define a state, we may now need to consider also the set of evaluable solutions at the current time step (in addition to other indicators, such as the current time step and repairs performed so far). The actions could represent the individual solutions evaluable at the current state, and as the reward function, we could use the best solution fitness obtained at the end of an episode (an algorithm run). Clearly, there are some obvious challenges with this approach including a large state and action space (this depends on the search space size), and the way solutions are created (as trying out all solutions is likely to be infeasible). To cope with large state and action space one may look to RL algorithms employing fitness approximation techniques (Sutton et al., 2000), and, to create solutions, we could employ estimation of distribution algorithms (Larrañaga and Lozano, 2001). It would be interesting to see whether such an RL algorithm could achieve results that are close, or perhaps even better, than the ones we achieved with an EA.

Extending search policies to account not only for resourcing issues but also other aspects of closed-loop optimization. We have considered a small set of resourcing issues in this thesis. However, the list of additional challenges an experimentalist may face in closed-loop optimization is long, as outlined in Section 2.3. In addition to the resourcing issues considered here, one may also need to account for noise, uncertainty, uncontrolled factors, and, as mentioned before, non-homogeneous experimental costs. Moreover, the aim is often to optimize several (conflicting) objectives simultaneously, and user preferences might also be available *a priori* or *progressively* during the optimization process. There is every reason to believe that further studies of these aspects would yield further gains of significant potential value to those employing closed-loop optimization in practice.

Appendix A

Further Results of Static Constraint-Handling Strategies Applied to ERCOPs

We now show additional results of the static constraint-handling strategies obtained on the MAX-SAT problem instance and the TwoMax function. For TwoMax, we also show the probability that the majority of a population climbs up the optimal slope; this measure indicates the robustness of a strategy against drift coming from an ERC because once the majority or all individuals of a population are on a slope, then it is unlikely that individuals on the other slope are generated and thus that slopes are switched. Section A.1 shows the results for commitment composite ERCs, and Section A.2 for periodic ERCs.

A.1 Commitment composite ERCs

We mentioned that the subpopulation strategy tends to generate fitter solutions than the forcing and regenerating strategy. This is again apparent from the results on the MAX-SAT instance (see Figure A.1). On the TwoMax problem (see Figure A.2), however, this effective and independent optimization of the subpopulation strategy over the constrained region leads to a population that is more likely to be on the suboptimal slope (this is also apparent from the probability of climbing up the optimal slope shown in the right plot, second row from the top); regenerating and forcing achieve a lower average best solution fitness than the subpopulation strategy but are slightly more likely to be on the optimal slope.

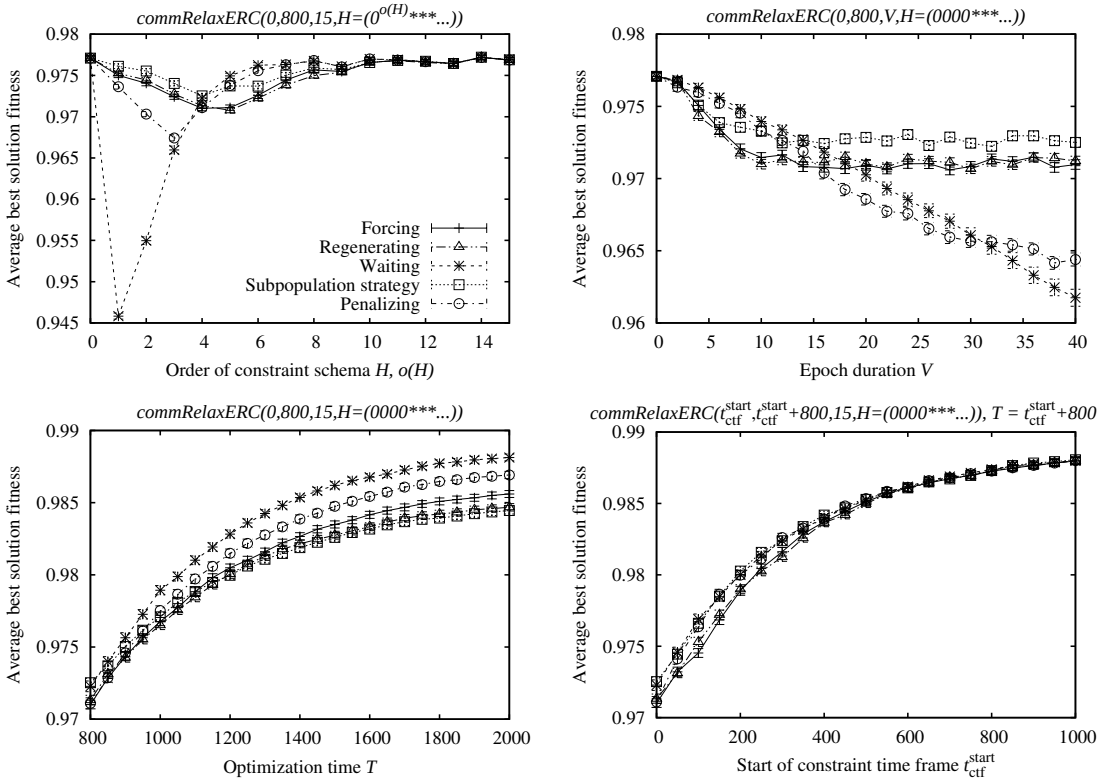

Figure A.1: Plots showing the average best solution fitness found and its standard error on a MAX-SAT problem instance as a function of the order of the constraint schema $o(H)$ (top left), the epoch duration V (top right), the optimization time T (bottom left), and the start of the constraint time frame $t_{\text{ctf}}^{\text{start}}$ (bottom right).

A.2 Periodic ERCs

When optimizing subject to periodic ERCs, we observed that waiting performs poorest for the majority of constraint parameter settings. This is further confirmed by the results obtained on the MAX-SAT problem instance (see Figure A.3). On the TwoMax problem (see Figure A.4), however, waiting (and penalizing) tend to perform significantly better than the repairing strategies for the majority of constraint parameter settings. The reason is that, on this problem, repairing decreases the probability of climbing up the optimal slope significantly and that is already true for low orders $o(H)$; (clearly, the fitter the repaired solutions, the more likely the EA is to climb up the suboptimal slope, causing the subpopulation strategy to perform poorest among the repairing strategies.)

The difference in the behavior of the constraint-handling strategies on the TwoMax problem yields also different results with respect to the performance impact of constraint schemata representing genetic material of different quality.

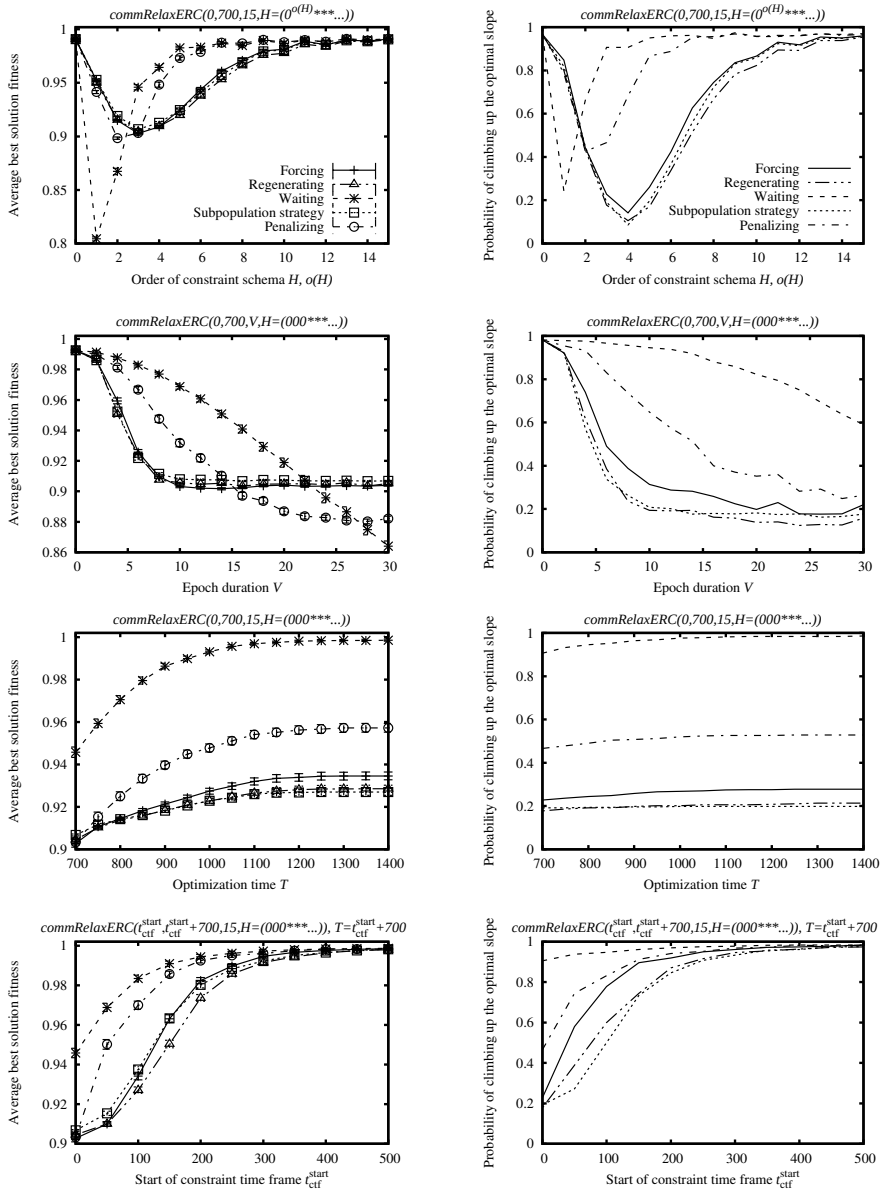

Figure A.2: Plots showing the average best solution fitness found and its standard error (left column) and the probability (measured by the relative number of algorithmic runs) that the majority of the individuals in a population ends up on the optimal slope at the end of the optimization (right column) on TwoMax as a function of the order of the constraint schema $o(H)$ (top row), the epoch duration V (second row from the top), the optimization time T (second row from the bottom), and the start of the constraint time frame $t_{\mathrm{ctf}}^{\mathrm{start}}$ (bottom row).

In fact, with any of the three repairing strategies one tends to obtain a higher average best solution fitness for schemata that represent very poor genetic material than for schemata that represent slightly better genetic material; Figure A.5

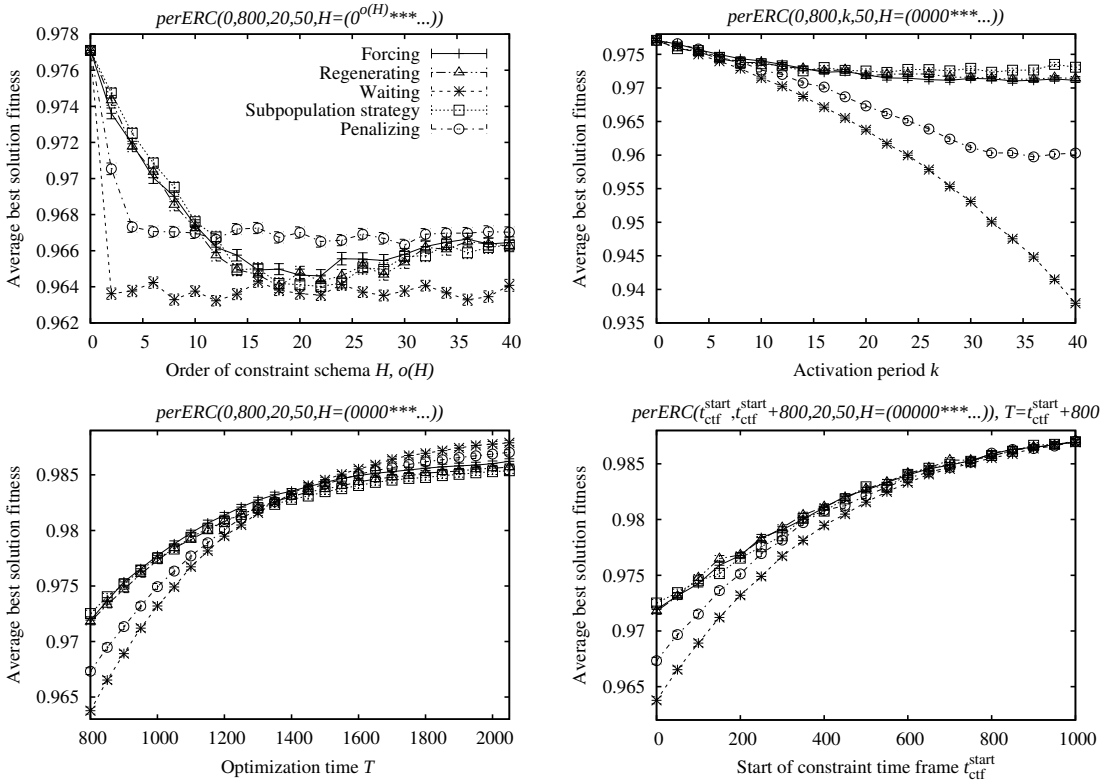

Figure A.3: Plots showing the average best solution fitness found and its standard error on a MAX-SAT problem instance as a function of the order of the constraint schema $o(H)$ (top left), the activation period k (top right), the optimization time T (bottom left), and the start of the constraint time frame $t_{\text{ctf}}^{\text{start}}$ (bottom right).

shows this for the subpopulation strategy. The reason for this pattern is that very poor schemata cause a population to climb up the suboptimal quickly, while slightly better schemata (represented by the light patch just below the straight line) have a weaker bias and thus cause a population to remain diverse for a longer period of time before climbing up the suboptimal slope. From Figure A.6 it is clear that waiting outperforms the subpopulation strategy for exactly the schemata where the subpopulation strategy hesitates for too long before climbing up a slope.

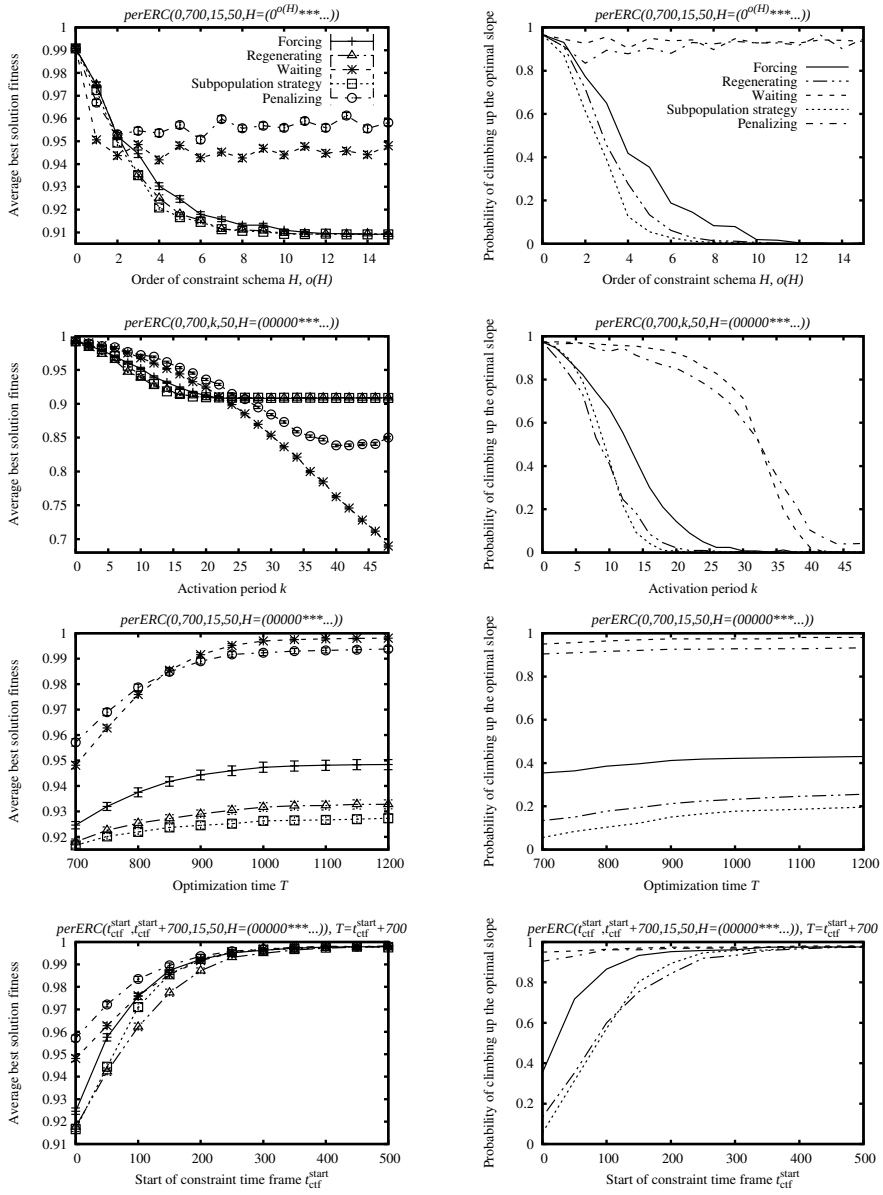

Figure A.4: Plots showing the average best solution fitness found and its standard error (left column) and the probability (measured by the relative number of algorithmic runs) that the majority of the individuals in a population ends up on the optimal slope at the end of the optimization (right column) on TwoMax as a function of the order of the constraint schema $o(H)$ (top row), the activation period k (second row from the top), the optimization time T (second row from the bottom), and the start of the constraint time frame $t_{\mathrm{ctf}}^{\mathrm{start}}$ (bottom row).

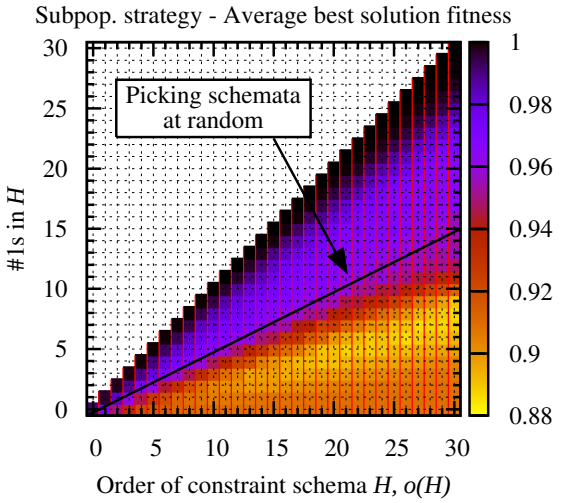

Figure A.5: A plot showing the average best solution fitness obtained by the subpopulation strategy on TwoMax as a function of the order of the constraint $o(H)$, and the number of order-defining bits in H with value 1 for the ERC $perERC(0, 700, 15, 50, H)$. The straight line represents the expected performance when picking a schema (i.e. the order-defining bits and their values) with a particular order at random.

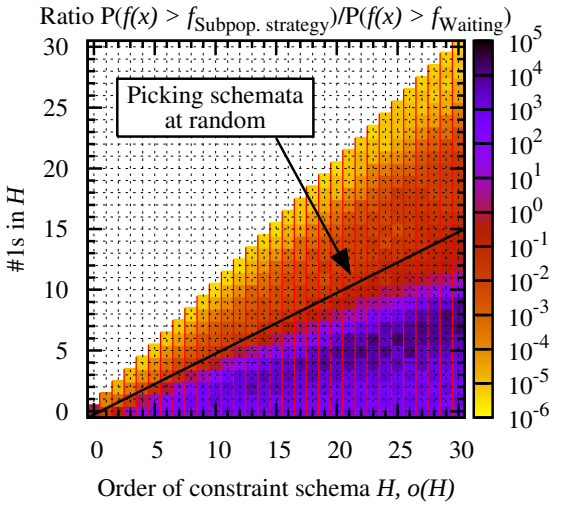

Figure A.6: A plot showing the ratio $\mathrm{P}(f(x) > f_{\text{Subpop. strategy}})/\mathrm{P}(f(x) > f_{\text{Waiting}})$ on TwoMax as a function of the order of the constraint schema $o(H)$, and the number of order-defining bits in H with value 1 for the ERC $perERC(0, 700, 15, 50, H)$; here, x is a random variable that represents the best solution from a set of solutions drawn uniformly at random from the search space and f_* the average best solution fitness obtained with strategy $*$. If $\mathrm{P}(f(x) > f_*)/\mathrm{P}(f(x) > f_{**}) > 1$, then strategy $**$ is able to achieve a higher average best solution fitness than strategy $*$ and a greater advantage of $**$ is indicated by a darker shading in the heat maps; similarly, if $\mathrm{P}(f(x) > f_*)/\mathrm{P}(f(x) > f_{**}) < 1$, then $*$ is better than $**$ and a lighter shading indicates a greater advantage of $*$.

Appendix B

The D-MAB Algorithm

The D-MAB algorithm (Hartland et al., 2006, 2007; Costa et al., 2008) is an adaptive operator selection method (see Section 3.1.2) that extends the *upper confidence bound 1* (UCB1) algorithm (Auer et al., 2002) with the statistical Page-Hinkley (PH) test (Page, 1954). Let us first describe the UCB1 algorithm, and then the PH test.

D-MAB uses the UCB1 algorithm, which is a popular bandit algorithm, to realize the operator selection scheme. After playing each arm during the initialization, UCB1 plays the arm j that maximizes $\bar{p}_j + \sqrt{\frac{2 \ln n}{n_j}}$, where $\bar{p}_j = \sum_{i=1}^{n_j} r_i / n_j$ is the average reward (credit) obtained from arm j and n_j the number of times this arm has been played so far, and $n = \sum_{j=1}^{J} n_j$ (J is the total number of arms) is the overall number of plays so far.

A drawback of UCB1 is its lack to account for dynamic changes in the quality of arms played (Costa et al., 2008). This has the effect that if the currently best arm becomes dominated by another arm, then it might take many plays before that arm overtakes the current best arm. To account for dynamic changes in the quality of arms played, D-MAB employs the statistical PH test. This test has the purpose to detect changes in the sequence of rewards obtained, and then to restart the multi-armed bandit. More precisely, when $r_1, ..., r_U$ is the sequence of rewards collected from playing arm i in the last U plays, the question is whether this sequence can be attributed to a single statistical law (hypothesis); if not, then there is a change in the statistical law and we can conclude that a change-point is detected (Hartland et al., 2006). The PH statistics is a standard criterion for testing this change-point hypothesis. The PH test maintains a random variable $m_U = \sum_{u=1}^{U} (r_u - \bar{r}_u + \delta)$, defined as the sum of the differences between each

reward r_u obtained so far and the average reward $\bar{r}_u = \sum_{i=1}^{u} r_i/u$; the parameter δ is a tolerance parameter related to the robustness of the test when dealing with slowly changing environments (Hartland et al., 2007). Two further variables are maintained: $M_U = \max\{m_u, u = 1, ..., U\}$, the maximum of the random variables $m_u, u = 1, ..., U$, and $PH_U = M_U - m_U$. When the difference PH_U is greater than some given threshold λ_{PH}, which specifies the desired false alarm rate, then we can conclude that a change point in the reward sequence has been detected (i.e. the null hypothesis can be rejected); formally we can define this procedure as.

$$\begin{aligned} \bar{r}_u &= \frac{1}{u}\sum_{i=1}^{u} r_i \ , \qquad \text{(B.1)} \\ m_U &= \sum_{u=1}^{U}(r_u - \bar{r}_u + \delta) \ , \\ M_U &= \max\{m_u, u = 1, ..., U\} \ , \\ PH_U &= M_U - m_U \ , \\ \text{Return } &(PH_U > \lambda_{\text{PH}}) \ . \end{aligned}$$

Increasing the value of λ_{PH} decreases the probability of encountering a false alarm but may mean that some change-points remain undetected. Increasing the value of the tolerance parameter δ reduces the value PH_U and thus has the effect that changes in the reward sequence have to be larger in order to be considered as a change point. In D-MAB, the PH test is implemented for each arm $j = 1, ..., J$, independently. Upon detecting a change point for any of the arms, D-MAB restarts the multi-armed bandit from scratch by setting the parameter values of $n_j, \bar{p}_j, m_j$, and M_j $(j = 1, ..., J)$ to 0.

Bibliography

H. Adeli and N.-T. Cheng. Augmented Lagrangian genetic algorithm for structural optimization. *Journal of Aerospace Engineering*, 7(1):104–118, 1994.

A. N. Aizawa and B. W. Wah. Scheduling of genetic algorithms in a noisy environment. *Evolutionary Computation*, 2(2):97–122, 1994.

R. Allmendinger and J. Knowles. Ephemeral resource constraints in optimization and their effects on evolutionary search. Technical Report MLO-20042010, University of Manchester, 2010.

L. Altenberg. The schema theorem and Price's theorem. *Foundations of Genetic Algorithms*, pages 23–49, 1995.

L. Altenberg. Fitness distance correlation analysis: An instructive counterexample. In *Proceedings of the International Conference on Genetic Algorithms*, pages 57–64, 1997.

P. J. Angeline. Tracking extrema in dynamic environments. In *Proceedings of the International Conference on Evolutionary Programming*, pages 335–345, 1997.

D. V. Arnold. Evolution strategies with adaptively rescaled mutation vectors. In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 2592–2599, 2005.

F. H. Arnold. Design by directed evolution. *Accounts of Chemical Research*, 31 (3):125–131, 1998.

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002.

A. Auger and B. Doerr. *Theory of Randomized Seach Heuristics*. World Scientific, 2011.

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

T. Bäck. On the behavior of evolutionary algorithms in dynamic environments. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 446–451, 1998.

T. Bäck and U. Hammel. Evolution strategies applied to perturbed objective functions. In *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 40–45, 1994.

T. Bäck, F. Hoffmeister, and H.-P. Schwefel. A survey of evolution strategies. In *Proceedings of the International Conference on Genetic Algorithms*, pages 2–9, 1991.

T. Bäck, D. B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

T. Bäck, U. Hammel H.-P., and Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17, 1997.

T. Bäck, J. Knowles, and O. M. Shir. Experimental optimization by evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (Companion)*, pages 2897–2916, 2010.

C. J. Bardeen, V. V. Yakovlev, K. R. Wilson, S. D. Carpenter, P. M. Weber, and W. S. Warren. Feedback quantum control of molecular electronic population transfer. *Chemical Physics Letters*, 280(1-2):151–158, 1997.

L. Barnett. Netcrawling-optimal evolutionary search with neutral networks. In *Proceedings of the Congress on Evolutionary Computation*, pages 30–37, 2001.

T. Bartz-Beielstein, K. E. Parsopoulos, and M. N. Vrahatis. Design and analysis of optimization algorithms using computational statistics. *Applied Numerical Analysis & Computational Mathematics*, 1(2):413–433, 2004.

T. Baumert, T. Brixner, V. Seyfried, M. Strehle, and G. Gerber. Femtosecond pulse shaping by an evolutionary algorithm with feedback. *Applied Physics B*, 65(6):779–782, 1997.

J. C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6(2):154–160, 1994.

J. C. Bean and A. B. Hadj-Alouane. A dual genetic algorithm for bounded integer programs. Technical Report TR 92-53, Department of Industrial and Operations Engineering, University of Michigan, 1992.

M. A. Bedau. Coping with complexity: Machine learning optimization of highly synergistic biological and biochemical systems. Keynote Talk at the International Conference on Genetic and Evolutionary Computation, 2010.

R. E. Bellman. *Dynamic programming*. Princeton University Press, 2003.

H.-G. Beyer. Toward a theory of evolution strategies: Some asymptotical results from the (1,+ λ)-theory. *Evolutionary computation*, 1(2):165–188, 1993.

H.-G. Beyer. Mutate large, but inherit small! on the analysis of rescaled mutations in $(\tilde{1}, \tilde{\lambda})$-es with noisy fitness data. In *Proceedings of Parallel Problem Solving from Nature*, pages 109–118, 1998.

H.-G. Beyer. Evolutionary algorithms in noisy environments: Theoretical issues and guidelines for practice. *Computer Methods in Applied Mechanics and Engineering*, 186:239–267, 2000.

H.-G. Beyer and H.-P. Schwefel. Evolution strategies – A comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

H.-G. Beyer and B. Sendhoff. Robust optimization – A comprehensive survey. *Computer Methods in Applied Mechanics and Engineering*, 196(33-34):3190–3218, 2007.

G. Bilchev and I. Parmee. Constrained optimization with an ant colony search model. In *Proceedings of the International Conference on Adaptive Computing in Engineering Design and Control*, pages 145–151, 1996.

M. Birattari. *Tuning Metaheuristics*. Springer, 2005.

M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 11–18, 2002.

C. M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, 1995.

C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

A. J. Booker, J. E. Dennis, P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural and Multidisciplinary Optimization*, 17(1):1–13, 1999.

A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

P. A. N. Bosman. Learning, anticipation and time-deception in evolutionary online dynamic optimization. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 39–47, 2005.

P. A. N. Bosman. Learning and anticipation in online dynamic optimization. In *Evolutionary Computation in Dynamic and Uncertain Environments*, pages 129–152, 2007.

P. A. N. Bosman and H. La Poutré. Learning and anticipation in online dynamic optimization with evolutionary algorithms: The stochastic case. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 1165–1172, 2007.

G. E. P. Box. Evolutionary operation: A method for increasing industrial productivity. *Applied Statistics*, 6(2):81–101, 1957.

G. E. P. Box and K. B. Wilson. On the experimental attainment of optimum conditions. *Journal of the Royal Statistical Society. Series B*, 13(1):1–45, 1951.

G. E. P Box, J. S. Hunter, and W. G. Hunter. *Statistics for Experimenters: Design, Innovation, and Discovery*. Wiley, 2nd edition, 2005.

J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1875–1882, 1999.

J. Branke. *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, 2001.

J. Branke and J. Elomari. Simultaneous tuning of metaheuristic parameters for various computing budgets. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 263–264, 2011.

J. Branke and D. Mattfeld. Anticipation and flexibility in dynamic scheduling. *International Journal of Production Research*, 43(15):3103–3129, 2005.

J. Branke and C. Schmidt. Sequential sampling in noisy environments. In *Proceedings of Parallel Problem Solving from Nature*, pages 202–211, 2004.

J. Branke, T. Kaußler, C. Schmidt, and H. Schmeck. A multi-population approach to dynamic optimization problems. *Adaptive Computing in Design and Manufacturing*, 2000:299–308, 2000.

H. J. Bremermann. Optimization through evolution and recombination. *Self-organizing systems*, pages 93–106, 1962.

S. H. Brooks. A discussion of random methods for seeking maxima. *Operations Research*, 6(2):244–251, 1958.

S. H. Brooks. A comparison of maximum-seeking methods. *Operations Research*, 7(4):430–457, 1959.

D. Büche, P. Stoll, R. Dornberger, and P. Koumoutsakos. Multiobjective evolutionary algorithm for the optimization of noisy combustion processes. *IEEE Transactions on Systems, Man, and Cybernetics C*, 32(4):460–473, 2002.

S. Bullock. Will selection for mutational robustness significantly retard evolutionary innovation on neutral networks? In *Artificial Life VIII*, pages 192–201, 2003.

E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu. A survey of hyper-heuristics. Technical Report NOTTCS-TR-SUB-0906241418-2747, School of Computer Science and Information Technology, University of Nottingham, 2009.

E. Byrne. Optimising the flow of experiments to a robot scientist with multi-objective evolutionary algorithms. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 2429–2436, 2007.

D. Calzolari, S. Bruschi, L. Coquin, J. Schofield, J. D. Feala, J. C. Reed, A. D. McCulloch, and G. Paternostro. Search algorithms as a framework for the optimization of drug combinations. *PLoS Computational Biology*, 4:1, 2008.

A. Carlisle and G. Dozier. Adapting particle swarm optimization to dynamic environments. In *Proceedings of International Conference on Artificial Intelligence*, pages 429–434, 2000.

S. E. Carlson and R. Shonkwiler. Annealing a genetic algorithm over constraints. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pages 3931–3936, 1998.

F. Caschera, G. Gazzola, M. A. Bedau, C. B. Moreno, A. Buchanan, J. Cawse, N. Packard, and M. M. Hanczyc. Automated discovery of novel drug formulations using predictive iterated high throughput experimentation. *PloS ONE*, 5 (1):e8546, 2010.

Y. H. Chang, T. Ho, and L. P. Kaelbling. All learning is local: Multi-agent learning in global reward games. *Proceedings of the Conference on Advances in Neural Information Processing Systems*, pages 807–814, 2004.

T. Chen, J. He, G. Sun, G. Chen, and X. Yao. A new approach for analyzing average time complexity of population-based evolutionary algorithms on unimodal problems. *IEEE Transactions on Systems, Man, and Cybernetics B*, 39 (5):1092–1106, 2009a.

Y. Chen, D. Oliver, and D. Zhang. Efficient ensemble-based closed-loop production optimization. *SPE Journal*, 14(4):634–645, 2009b.

G. Ciuti, R. Donlin, P. Valdastri, A. Arezzo, A. Menciassi, M. Morino, and P. Dario. Robotic versus manual control in magnetic steering of an endoscopic capsule. *Endoscopy*, 42(2):148–152, 2010.

H. G. Cobb. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Technical Report 6760, Naval Research Lab Washigton DC, 1990.

H. G. Cobb and J. J. Grefenstette. Genetic algorithms for tracking changing environments. In *Proceedings of the International Conference on Genetic Algorithms*, pages 523–530, 1993.

C. A. Coello Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11-12):1245–1287, 2002.

D. Coppersmith, D. Gamarnik, M. T. Hajiaghayi, and G. B. Sorkin. Random MAX SAT, random MAX CUT, and their phase transitions. *Random Structures & Algorithms*, 24(4):502–545, 2004.

D. W. Corne, M. Oates, and D. B. Kell. Landscape state machines: Tools for evolutionary algorithm performance analyses and landscape / algorithm mapping. In *Applications of Evolutionary Computing, EvoWorkshop*, pages 187–198, 2003.

L. Da Costa, A. Fialho, M. Schoenauer, and M. Sebag. Adaptive operator selection with dynamic multi-armed bandits. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 913–920, 2008.

P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In *Practice and Theory of Automated Timetabling III*, pages 176–190, 2000.

S. P. Coy, B. L. Golden, G. C. Runger, and E. A. Wasil. Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7(1):77–97, 2001.

N. Cressie. *Statistics for Spatial Data*. Wiley, 1993.

A. Czarn, C. MacNish, K. Vijayan, and B. Turlach. Statistical exploratory analysis of genetic algorithms: The importance of interaction. *IEEE Transactions on Evolutionary Computation*, 8(4):405–421, 2004.

Y. Davidor. A genetic algorithm applied to robot trajectory generation. *Handbook of Genetic Algorithms*, pages 144–165, 1991.

Z. S. Davies, R. J. Gilbert, R. J. Merry, D. B. Kell, M. K. Theodorou, and G. W. Griffith. Efficient improvement of silage additives by using genetic algorithms. *Applied and Environmental Microbiology*, 66(4):1435–1443, 2000.

L. Davis. Adapting operator probabilities in genetic algorithms. In *Proceedings of the International Conference on Genetic Algorithms*, pages 61–69, 1989.

L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

T. E. Davis and J. C. Principe. A Markov chain framework for the simple genetic algorithm. *Evolutionary Computation*, 1(3):269–288, 1993.

K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975.

K. A. De Jong and W. M. Spears. Using genetic algorithms to solve NP-complete problems. In *Proceedings of the International Conference on Genetic Algorithms*, pages 124–132, 1989.

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

K. Deb, S. Gupta, D. Daum, J. Branke, A. K. Mall, and D. Padmanabhan. Reliability-based optimization using evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 13(5):1054–1074, 2009.

R. Dechter. *Constraint processing*. Morgan Kaufmann, 2003.

B. Doerr, D. Johannsen, and M. Schmidt. Runtime analysis of the (1+1) evolutionary algorithm on strings over finite alphabets. In *Foundations of Genetic Algorithms*, pages 119–126, 2011.

J. L. Doob. *Stochastic Processes*. Wiley, 1953.

J. Dréo. Using performance fronts for parameter setting of stochastic metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 2197–2200, 2009.

W. B. Dunn. Email discussion with W. B. Dunn, March 2011.

A. E. Eiben and S. K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, pages 19–31, 2011.

A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer Verlag, 2003.

A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2): 124–141, 1999.

A. E. Eiben, M. Horvath, W. Kowalczyk, and M. Schut. Reinforcement learning for online control of evolutionary algorithms. In *Engineering Self-Organising Systems*, pages 151–160, 2007.

J. R. G. Evans, M. J. Edirisinghe, P. V. Coveney, and J. Eames. Combinatorial searches of inorganic materials using the ink-jet printer: Science, philosophy and technology. *Journal of the European Ceramic Society*, 21(13):2291–2299, 2001.

X. Feng, E. Shea-Brown, B. Greenwald, H. Rabitz, and R. Kosut. Toward closed-loop optimization of deep brain stimulation for Parkinson's disease: Concepts and lessons from a computational model. *Journal of Neural Engineering*, 4(2): L14–L21, 2007.

V. Ferreira, N. Ortín, and J. F. Cacho. Optimization of a procedure for the selective isolation of some powerful aroma thiols: Development and validation of a quantitative method for their determination in wine. *Journal of Chromatography A*, 1143(1-2):190–198, 2007.

J. G. Ferrier and D. E. Block. Neural-network-assisted optimization of wine blending based on sensory analysis. *American Journal of Enology and Viticulture*, 52(4):386, 2001.

A. Fialho. *Adaptive Operator Selection for Optimization*. PhD thesis, Université Paris-Sud, 2010.

S. G. Ficici, R. A. Watson, and J. B. Pollack. Embodied evolution: A response to challenges in evolutionary robotics. In *Proceedings of the European Workshop on Learning Robots*, pages 14–22, 1999.

J. M. Fitzpatrick and J. J. Grefenstette. Genetic algorithms in noisy environments. *Machine Learning*, 3(2):101–120, 1988.

D. Floreano and F. Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In *From Animals to Animats 3*, pages 421–430, 1994.

T. C. Fogarty. Varying the probability of mutation in the genetic algorithm. In *Proceedings of the International Conference on Genetic Algorithms*, pages 104–109, 1989.

D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14, 1994.

D. B. Fogel. *Evolutionary Computation. The Fossil Record. Selected Readings on the History of Evolutionary Algorithms*. IEEE Press, 1998.

L. J. Fogel. Autonomous automata. *Industrial Research*, 4(2):14–19, 1962.

L. J. Fogel. *On the Organization of Intellect*. PhD thesis, University of California, 1964.

S. Forrest and M. Mitchell. Relative building-block fitness and the building-block hypothesis. In *Foundations of Genetic Algorithms*, pages 109–126, 1993.

O. François and C. Lavergne. Design of evolutionary algorithms – A statistical perspective. *IEEE Transactions on Evolutionary Computation*, 5(2):129–148, 2001.

A. S. Fraser. Simulation of genetic systems by automatic digital computers – VI. Epistasis. *Australian Journal of Biological Sciences*, 13(2):150–162, 1957.

R. M. Friedberg. A learning machine: Part I. *IBM Journal of Research and Development*, 2(1):2–13, 1958.

R. M. Friedberg, B. Dunham, and J. H. North. A learning machine: Part II. *IBM Journal of Research and Development*, 3(7):282–287, 1959.

T. Friedrich, P. S. Oliveto, D. Sudholt, and C. Witt. Theoretical analysis of diversity mechanisms for global exploration. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 945–952, 2008.

F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, 1977.

O. Gobin, A. M. Joaristi, and F. Schüth. Multi-objective optimization in combinatorial chemistry applied to the selective catalytic reduction of NO with C_3H_6. *Journal of Catalysis*, 252(2):205 – 214, 2007.

D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison Wesley, Reading, 1989.

D. E. Goldberg. Probability matching, the magnitude of reinforcement, and classifier system bidding. *Machine Learning*, 5(4):407–425, 1990.

D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the International Conference on Genetic Algorithms*, pages 41–49, 1987.

D. E. Goldberg and P. Segrest. Finite Markov chain analysis of genetic algorithms. In *Proceedings of the International Conference on Genetic Algorithms*, pages 1–8, 1987.

D. E. Goldberg and R. Smith. Nonstationary function optimization using genetic algorithm with dominance and diploidy. In *Proceedings of the International Conference on Genetic Algorithms*, pages 59–68, 1987.

D. E. Goldberg, K. Deb, and J. H. Clark. Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, 6:333–362, 1992.

G. Greenwood and A. Tyrrell. *Introduction to Evolvable Hardware: A Practical Guide for Designing Self-Adaptive System*. IEEE Press, 2006.

J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128, 1986.

J. J. Grefenstette. Genetic algorithms for changing environments. *Proceedings of Parallel Problem Solving from Nature*, 2:137–144, 1992.

B. Hadad and C. Eick. Supporting polyploidy in genetic algorithms using dominance vectors. In *Proceedings of the International Conference on Evolutionary Programming*, pages 223–234, 1997.

B. Hajek. Hitting-time and occupation-time bounds implied by drift analysis with applications. *Advances in Applied Probability*, 13(3):502–525, 1982.

U. Hammel and T. Bäck. Evolution strategies on noisy functions – How to improve convergence properties. In *Proceedings of Parallel Problem Solving from Nature*, pages 159–168, 1994.

N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

N. Hansen, A. S. P. Niederberger, L. Guzzella, and P. Koumoutsakos. A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *IEEE Transactions on Evolutionary Computation*, 13(1):180–197, 2009.

C. Hartland, S. Gelly, N. Baskiotis, O Teytaud, and M. Sebag. Multi-armed bandits, dynamic environments and meta-bandits. In NIPS Workshop "Online Trading of Exploration and Exploitation", 2006.

C. Hartland, N. Baskiotis, S. Gelly, M. Sebag, and O. Teytaud. Change point detection and meta-bandits for online learning in dynamic environments. In *Proceedings of CAp*, pages 237–250, 2007.

I. Harvey, P. Husbands, D. Cliff, A. Thompson, and N. Jakobi. Evolutionary robotics: The Sussex approach. *Robotics and Autonomous Systems*, 20(2-4): 205–224, 1996.

J. He and X. Yao. Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence*, 127(1):57–85, 2001.

J. He and X. Yao. From an individual to a population: An analysis of the first hitting time of population-based evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):495–511, 2002.

T. Hebbron, S. Bullock, and D. Cliff. $NK\alpha$: Non-uniform epistatic interations in an extended *NK* model. In *Artificial Life XI*, pages 234–241, 2008.

M. Herdy. Evolutionary optimization based on subjective selection — evolving blends of coffee. In *European Congress on Intelligent Techniques and Soft Computing*, pages 640–644, 1997.

J. H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.

A. Homaifar, S. H. Y. Lai, and X. Qi. Constrained optimization via genetic algorithms. *Simulation*, 62(4):242–254, 1994.

J. Horn. Finite Markov chain analysis of genetic algorithms with niching. In *Proceedings of the International Conference on Genetic Algorithms*, pages 110–117, 1993.

T. Hornung, M. Motzkus, and R. de Vivie-Riedle. Adapting optimal control theory and using learning loops to provide experimentally feasible shaping mask patterns. *The Journal of Chemical Physics*, 115:3105, 2001.

C. Van Hoyweghen, D. E. Goldberg, and B. Naudts. From TwoMax to the Ising model: Easy and hard symmetrical problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 626–633, 2002.

B. D. Hughes. *Random walks and random environments*. Clarendon Press, 1995.

E. Hughes. Evolutionary multi-objective ranking with uncertainty and noise. In *Proceedings of the International Conference on Evolutionary Multi-Criterion Optimization*, pages 329–343, 2001.

W. G. Hunter and J. R. Kittrell. Evolutionary operation: A review. *Technometrics*, 8(3):389–397, 1966.

P. Husbands and J. A. Meyer. *Evolutionary Robotics*. Springer, 1998.

A. K. Hutzschenreuter, P. A. N. Bosman, and H. La Poutré. Evolutionary multi-objective optimization for dynamic hospital resource management. In *Proceedings of Evolutionary Multi-Criterion Optimization*, pages 320–334, 2009.

A. K. Hutzschenreuter, P. A. N. Bosman, and H. La Poutré. Enhanced hospital resource management using anticipatory policies in online dynamic multi-objective optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 541–542, 2010.

C. Igel and M. Toussaint. A no-free-lunch theorem for non-uniform distributions of target functions. *Journal of Mathematical Modelling and Algorithms*, 3(4): 313–322, 2004.

R. Jarvis, W. Rowe, N. Yaffe, R. O'Connor, J. Knowles, E. Blanch, and R. Goodacre. Multiobjective evolutionary optimisation for surface-enhanced Raman scattering. *Analytical and Bioanalytical Chemistry*, 397(5):1893–1901, 2010.

Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, 9(1):3–12, 2005.

Y. Jin and J. Branke. Evolutionary optimization in uncertain environments – A survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, 2005.

Y. Jin and B. Sendhoff. Trade-off between performance and robustness: An evolutionary multiobjective approach. In *Evolutionary Multi-Criterion Optimization*, pages 237–252, 2003.

Y. Jin, M. Olhofer, and B. Sendhoff. A framework for evolutionary optimization with approximate fitness functions. *IEEE Transactions on Evolutionary Computation*, 6(5):481–494, 2002.

Y. Jin, S. Oh, and M. Jeon. Incremental approximation of nonlinear constraint functions for evolutionary constrained optimization. In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.

J. Joines and C. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 579–584, 1994.

D. R. Jones. A Taxonomy of Global Optimization Methods Based on Response Surfaces. *Journal of Global Optimization*, 21(4):345–383, 2001.

T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the International Conference on Genetic Algorithms*, pages 184–192, 1995.

R. S. Judson and H. Rabitz. Teaching lasers to control molecules. *Physical Review Letters*, 68(10):1500–1503, 1992.

B. A. Julstrom. What have you done for me lately? Adapting operator probabilities in a steady-state genetic algorithm. In *Proceedings of the International Conference on Genetic Algorithms*, pages 81–87, 1995.

L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

S. Kauffman. Adaptation on rugged fitness landscapes. In *Lecture Notes in the Sciences of Complexity*, pages 527–618, 1989.

L. Kaufman and P. J. Rousseeuw. *Finding groups in data: An introduction to cluster analysis*. Wiley, 1990.

S. Kazarlis and V. Petridis. Varying fitness functions in genetic algorithms: Studying the rate of increase of the dynamic penalty terms. In *Proceedings of Parallel Problem Solving from Nature*, pages 211–220, 1998.

M. D. Kidwell and D. J. Cook. Genetic allgorithm for dynamic task scheduling. In *Proceedings of the IEEE International Phoenix Conference on Computers and Communications*, pages 61–67, 1994.

D. G. Kime and P. Husbands. Riemann mapping constraint handling method for genetic algorithms. Technical Report CSRP 469, COGS, University of Sussex, 1997.

R. D. King, K. E. Whelan, F. M. Jones, P. G. K. Reiser, C. H. Bryant, S. H. Muggleton, D. B. Kell, and S. G. Oliver. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427:247–252, 2004.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

J. Klockgether and H.-P. Schwefel. Two-phase nozzle and hollow core jet experiments. In *Engineering Aspects of Magnetohydrodynamics*, pages 141–148, 1970.

C. G. Knight, M. Platt, W. Rowe, D. C. Wedge, F. Khan, P. J. R. Day, A. McShea, J. Knowles, and D. B. Kell. Array-based evolution of DNA aptamers allows modelling of an explicit sequence-fitness landscape. *Nucleic Acids Research*, 37 (1):e6, 2009.

J. Knowles. Closed-loop evolutionary multiobjective optimization. *IEEE Computational Intelligence Magazine*, 4(3):77–91, 2009.

S. Koziel and Z. Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7 (1):19–44, 1999.

P. Larrañaga and J. A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2001.

A. Larsen. *The Dynamic Vehicle Routing Problem*. PhD thesis, Technical University of Denmark, 2000.

J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011.

P. K. Lehre. Fitness-levels for non-elitist populations. In *Proceedings of the Conference on Genetic and Evolutionary Computation*, pages 2075–2082, 2011.

B. Levitan and S. Kauffman. Adaptive walks with noisy fitness measurements. *Molecular Diversity*, 1(1):53–68, 1995.

G. E. Liepins and W. D. Potter. A genetic algorithm approach to multiple-fault diagnosis. In *Handbook of Genetic Algorithms*, pages 237–250, 1991.

G. E. Liepins and M. D. Vose. Representational issues in genetic optimization. *Journal of Experimental and Theoretical Computer Science*, 2(2):4–30, 1990.

H. Lipson and J. B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974–978, 2000.

F. G. Lobo and D. E. Goldberg. Decision making in a hybrid genetic algorithm. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 121–125, 1997.

F. G. Lobo, C. F. Lima, and Z. Michalewicz. *Parameter setting in evolutionary algorithms*. Springer, 2007.

W. Lubeigt, G. J. Valentine, J. M. Girkin, E. Bente, and D. Burns. Active transverse mode control and optimization of an all-solid-state laser using an intracavity adaptive-optic mirror. *Optics Express*, 10(13):550–555, 2002.

A. Mahajan and D. Teneketzis. Multi-armed bandit problems. *Foundations and Applications of Sensor Management*, pages 121–151, 2008.

S. W. Mahfoud. Finite Markov chain models of an alternative selection strategy for the genetic algorithm. *Complex Systems*, 7:155–170, 1991.

S. W. Mahfoud. *Niching methods for genetic algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, 1995.

S. Markon, D. V. Arnold, T. Bäck, T. Beielstein, and H.-G. Beyer. Thresholding – A selection operator for noisy ES. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 465–472, 2001.

O. Maron and A. W. Moore. The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Review*, 11(1):193–225, 1997.

P. Marsh, D. Burns, and G. J. Girkin. Practical implementation of adaptive optics in multiphoton microscopy. *Optics Express*, 11(10):1123–1130, 2003.

R. L. Mason, R. F. Gunst, and J. L. Hess. *Statistical Design and Analysis of Experiments: With Applications to Engineering and Science*. Wiley, 1989.

M. Matarić and D. Cliff. Challenges in evolving controllers for physical robots. *Robotics and Autonomous Systems*, 19(1):67–83, 1996.

G. Matheron. Principles of geostatistics. *Economic Geology*, 58(8):1246–1266, 1963.

K. McClymont and E. C. Keedwell. Markov chain hyper-heuristic (MCHH): An online selective hyper-heuristic for multi-objective continuous problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 2003–2010, 2011.

D. Meignan, A. Koukam, and J.-C. Créput. Coalition-based metaheuristic: A self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, 16(6):859–879, 2010.

D. Meshulach and Y. Silberberg. Coherent quantum control of two-photon transitions by a femtosecond laser pulse. *Nature*, 396:239–242, 1998.

E. Mezura-Montes. *Constraint-Handling in Evolutionary Optimization*. Springer Verlag, 2009.

Z. Michalewicz. Genetic algorithms, numerical optimization, and constraints. In *Proceedings of the International Conference on Genetic Algorithms*, pages 151–158, 1995.

Z. Michalewicz. Some thoughts on wine production. Keynote Talk at the International Conference on Parallel Problem Solving from Nature, 2010.

Z. Michalewicz and N. F. Attia. Evolutionary optimization of constrained problems. In *Proceedings of the International Conference on Evolutionary Programming*, pages 98–108, 1994.

Z. Michalewicz and G. Nazhiyath. Genocop III: A co-evolutionary algorithm for numerical optimization with nonlinear constraints. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 647–651, 1995.

Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.

B. L. Miller. *Noise, sampling, and efficient genetic algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, 1997.

M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1996.

A. Moglia, A. Menciassi, M. Schurr, and P. Dario. Wireless capsule endoscopy: From diagnostic devices to multipurpose robotic systems. *Biomedical Microdevices*, 9(2):235–243, 2007.

F. Mondada and D. Floreano. Evolution of neural control structures: Some experiments on mobile robots. *Robotics and Autonomous Systems*, 16(2-4): 183–195, 1995.

E. M. Montes. *Alternative Techniques to Handle Constraints in Evolutionary Optimization*. PhD thesis, CINVESTAV-IPN, 2004.

D. C. Montgomery. *Design and Analysis of Experiments*. Wiley, 1976.

S. Montserrat, R. Iñaki, O. François, G. Francesc, and C. Carles. Application of factorial design to the optimization of medium composition in batch cultures of streptomyces lividans TK21 producing a hybrid antibiotic. *Biotechnology Letters*, 15(6):559–564, 1993.

H. Mühlenbein. Evolution in time and space-the parallel genetic algorithm. In *Foundations of Genetic Algorithms*, pages 316–338, 1991.

H. Mühlenbein. How genetic algorithms really work: Mutation and hillclimbing. In *Proceedings of Parallel Problem Solving from Nature*, pages 15–25, 1992.

H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions – I. Binary parameters. In *Proceedings of Parallel Problem Solving from Nature*, pages 178–187, 1996.

H. Mühlenbein and D. Schlierkamp-Voosen. Optimal interaction of mutation and crossover in the breeder genetic algorithm. In *Proceedings of the International Conference on Genetic Algorithms*, page 648, 1993.

H. Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. *Parallel computing*, 17(6-7):619–632, 1991.

S. D. Müller, N. N. Schraudolph, and P. D. Koumoutsakos. Step size adaptation in evolution strategies using reinforcement learning. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 151–156, 2002.

R. Myers and E. R. Hancock. Empirical modelling of genetic algorithms. *Evolutionary Computation*, 9(4):461–493, 2001.

R. H. Myers, D. C. Montgomery, and C. M. Anderson-Cook. *Response surface methodology: Process and product optimization using designed experiments*. Wiley, 2009.

T. Nakama. Theoretical analysis of genetic algorithms in noisy environments based on a Markov model. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1001–1008, 2008.

R. Nakano. Conventional genetic algorithm for job shop problems. In *Proceedings of the International Conference on Genetic Algorithms*, pages 474–479, 1991.

R. Nakano and T. Yamada. Conventional genetic algorithm for job shop problems. In *Proceedings of the International Conference on Genetic Algorithms*, pages 474–479, 1991.

V. Nannen and A. E. Eiben. A method for parameter calibration and relevance estimation in evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 183–190, 2006.

A. Nareyek. Choosing search heuristics by non-stationary reinforcement learning. In *Metaheuristics: Computer decision-making*, pages 523–544, 2003.

A. L. Nelson, G. J. Barlow, and L. Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4): 345–370, 2009.

K. P. Ng and K. C. Wong. A new diploid scheme and dominance change mechanism for non-stationary function optimization. In *Proceedings of the International Conference on Genetic Algorithms*, pages 159–166, 1995.

T. T. Nguyen. *Continuous dynamic optimisation using evolutionary algorithms*. PhD thesis, University of Birmingham, 2011.

T. T. Nguyen and X. Yao. Benchmarking and solving dynamic constrained problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 690–697, 2009a.

T. T. Nguyen and X. Yao. Dynamic time-linkage problems revisited. *Proceedings of European Workshops on Applications of Evolutionary Computation*, pages 735–744, 2009b.

A. Nix and M. D. Vose. Modeling genetic algorithms with Markov chains. *Annals of Mathematics and Artificial Intelligence*, 5:79–88, 1992.

S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press / Bradford Book, 2004.

S. Nolfi, D. Floreano, O. Miglino, and F. Mondada. How to evolve autonomous robots: Different approaches in evolutionary robotics. In *Artificial Life IV*, pages 190–197, 1994.

J. R. Norris. *Markov Chains (Cambridge Series in Statistical and Probabilistic Mathematics)*. Cambridge University Press, 1998.

G. Ochoa and M. Schoenauer. Self-* Search – New frontier track. Track at the International Conference on Genetic and Evolutionary Computation, 2011.

G. Ochoa, I. Harvey, and H. Buxton. Optimal mutation rates and selection pressure in genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 315–322, 2000.

G. Ochoa, M. Schoenauer, and D. Whitley. Workshop on self-tuning, self-configuring and self-generating search heuristics (self* 2010). Workshop at the International Conference on Parallel Problem Solving from Nature, 2010.

S. Oh, Y. Jin, and M. Jeon. Approximate models for constraint functions in evolutionary constrained optimization. *International Journal of Innovative Computing, Information and Control*, (to appear), 2011.

S. O'Hagan, W. B. Dunn, M. Brown, J. Knowles, and D. B. Kell. Closed-loop, multiobjective optimization of analytical instrumentation: Gas chromatography / time-of-flight mass spectrometry of the metabolomes of human serum and of yeast fermentations. *Analytical Chemistry*, 77(1):290–303, 2005.

S. O'Hagan, W. B. Dunn, J. Knowles, D. Broadhurst, R. Williams, J. J. Ashworth, M. Cameron, and D. B. Kell. Closed-loop, multiobjective optimization of two-dimensional gas chromatography / mass spectrometry for serum metabolomics. *Analytical Chemistry*, 79(2):464–476, 2007.

M. Olhofer, Y. Jin, and B. Sendhoff. Adaptive encoding for aerodynamic shape optimization using evolution strategies. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 576–583, 2001.

M. Olhofer, D. Yankulova, and B. Sendhoff. Autonomous experimental design optimization of a flapping wing. *Genetic Programming and Evolvable Machines*, 12:23–47, 2011.

P. S. Oliveto and C. Witt. Simplified drift analysis for proving lower bounds in evolutionary computation. *Algorithmica*, 59(3):369–386, 2011.

Y. S. Ong, P. B. Nair, A. J. Keane, and K. W. Wong. Surrogate-assisted evolutionary optimization frameworks for high-fidelity engineering design problems. *Knowledge Incorporation in Evolutionary Computation*, pages 307–332, 2004.

F. Oppacher and M. Wineberg. The shifting balance genetic algorithm: Improving the GA in a dynamic environment. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 504–510, 1999.

D. Orvosh and L. Davis. Using a genetic algorithm to optimize problems with feasability constraints. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 548–553, 1994.

E. S. Page. Continuous inspection schemes. *Biometrika*, 41:100–115, 1954.

J. Paredis. Co-evolutionary constraint satisfaction. In *Proceedings of Parallel Problem Solving from Nature*, pages 46–55, 1994.

C. O. Paschereit, B. Schuermans, and D. Büche. Combustion process optimization using evolutionary algorithm. In *Proceedings of ASME Turbo Expo*, pages 281–291, 2003.

G. S. Peace. *Taguchi methods: A hands-on approach*. Addison-Wesley, 1993.

B. J. Pearson, J. L. White, T. C. Weinacht, and P. H. Bucksbaum. Coherent control using adaptive learning algorithms. *Physical Review A*, 63(6):063412, 2001.

M. Pelikan and D. E. Goldberg. Genetic algorithms, clustering, and the breaking of symmetry. In *Proceedings of Parallel Problem Solving from Nature*, pages 385–394, 2000.

M. D. Pendrith. On reinforcement learning of control actions in noisy and non-Markovian domains. Technical report, University of New South Wales, 1994.

J. E. Pettinger. Using reinforcement learning techniques to improve the performance of genetic algorithms. Master's thesis, The University of Exeter, 2002.

J. E. Pettinger and R. M. Everson. Controlling genetic algorithms with reinforcement learning. Technical report, The University of Exeter, 2003.

M. Platt, W. Rowe, D. C. Wedge, D. B. Kell, J. Knowles, and P. J. R. Day. Aptamer evolution for array-based diagnostics. *Analytical Biochemistry*, 390 (2):203–205, 2009.

J. B. Pollack, H. Lipson, P. Funes, S. G. Ficici, and G. Hornby. Coevolutionary robotics. In *Proceedings of the NASA / DoD Workshop*, pages 208–216, 1999.

J. B. Pollack, H. Lipson, G. Hornby, and P. Funes. Three generations of automatically designed robots. *Artificial Life*, 7(3):215–223, 2001.

V. Poorna and P. R. Kulkarni. A study of inulinase production in aspergillus niger using fractional factorial design. *Bioresource Technology*, 54(3):315–320, 1995.

D. Powell and M. M. Skolnick. Using genetic algorithms in engineering design optimization with non-linear constraints. In *Proceedings of the International Conference on Genetic Algorithms*, pages 424–431, 1993.

D. A. Preece. R. A. Fisher and experimental design: A Review. *Biometrics*, 46 (4):925–935, 1990.

G. R. Price. Selection and covariance. *Nature*, 227:520–521, 1970.

A. Prügel-Bennett. Symmetry breaking in population-based optimization. *IEEE Transactions on Evolutionary Computation*, 8(1):63–79, 2004.

A. Prügel-Bennett and J. L. Shapiro. Analysis of genetic algorithms using statistical mechanics. *Physical Review Letters*, 72(9):1305–1309, 1994.

H. N. Psaraftis. Dynamic vehicle routing problems. *Vehicle routing: Methods and studies*, pages 223–248, 1988.

M. Qasem and A. Prügel-Bennett. Learning the large-scale structure of the MAX-SAT landscape using populations. *IEEE Transactions on Evolutionary Computation*, 14(4):518–529, 2010.

N. J. Radcliffe. Schema processing. In *Handbook of Evolutionary Computation*, pages B2.5–1–10, 1997.

N. Raman and F. B. Talbot. The job shop tardiness problem: A decomposition approach. *European Journal of Operational Research*, 69(2):187–199, 1993.

S. Rana, D. Whitley, and R. Cogswell. Searching in the presence of noise. In *Proceedings of Parallel Problem Solving from Nature*, pages 198–207, 1996.

M. Rattray. *Modelling the Dynamics of Genetic Algorithms using Statistical Mechanics*. PhD thesis, University of Manchester, 1996.

I. Rechenberg. Cybernetic solution path of an experimental problem. *Library translation*, 1964.

I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973.

I. Rechenberg. *Evolutionsstrategie '94*. Friedrich Frommann Verlag, 1994.

I. Rechenberg. Case studies in evolutionary experimentation and computation. *Computer Methods in Applied Mechanics and Engineering*, 2-4(186):125–140, 2000.

C. R. Reeves and J. E. Rowe. *Genetic algorithms – Principles and Perspectives: A guide to GA theory*. Kluwer Academic Publishers, 2003.

C. R. Reeves and C. Wright. An experimental design perspective on genetic algorithms. In *Foundations of Genetic Algorithms*, pages 7–22, 1995.

C. R. Reeves and C. Wright. Genetic algorithms and the design of experiments. In *Evolutionary Algorithms: IMA Volumes in Mathematics and its Applications*, 1996.

A. Reynolds and D. W. Corne. Conversation with A. Reynolds and D. W. Corne at the International Conference on Parallel Problem Solving from Nature, September 2010.

R. Le Riche, C. Knopf-Lenior, and R. T. Hafka. A segregated genetic algorithm for constrained structural optimization. In *Proceedings of the International Conference on Genetic Algorithms*, pages 558–565, 1995.

H. Richter. Memory design for constrained dynamic optimization problems. In *Proceedings of the European Conference on Applications of Evolutionary Computation*, pages 552–561, 2010.

H. Richter and F. Dietel. Solving dynamic constrained optimization problems with asynchronous change pattern. In *Proceedings of the European Conference on the Applications of Evolutionary Computation*, pages 334–343, 2011.

J. Rincon, J. Fuertes, A. Moya, J. M. Monteagudo, and L. Rodriguez. Optimization of the fermentation of whey by *Lactobacillus casei*. *Acta Biotechnologica*, 13(4):323–331, 1993.

H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–35, 1952a.

H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952b.

J. Roslund, O. M. Shir, T. Bäck, and H. Rabitz. Accelerated optimization and automated discovery with covariance matrix adaptation for experimental quantum control. *Physical Review A*, 80(4):043415, 2009.

F. Rothlauf. *Representations for genetic and evolutionary algorithms*. Springer, 2006.

W. Rowe, D. W. Corne, and J. Knowles. Predicting stochastic search algorithm performance using landscape state machines. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 2944–2951, 2006.

W. Rowe, D. C. Wedge, M. Platt, D. B. Kell, and J. Knowles. Predictive models for population performance on real biological fitness landscapes. *Bioinformatics*, 26(17):2145–2152, 2010.

G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report CUED / F-INFENG / TR 166, Cambridge University Engineering Department, 1994.

T. P. Runarsson. Constrained evolutionary optimization by approximate ranking and surrogate models. In *Proceedings of Parallel Problem Solving from Nature*, pages 401–410, 2004.

T. P. Runarsson and X. Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, 2000.

J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4(4):409–423, 1989.

S. Salcedo-Sanz. A survey of repair methods used as constraint handling techniques in evolutionary algorithms. *Computer Science Review*, 3(3):175–192, 2009.

Y. Sano and H. Kita. Optimization of noisy fitness functions by means of genetic algorithms using history of search. In *Proceedings of Parallel Problem Solving from Nature*, pages 571–580, 2000.

R. Schlögl. Combinatorial chemistry in heterogeneous catalysis: A new scientific approach or the king's new clothes? *Angewandte Chemie International Edition*, 37(17):2333–2336, 1998.

J. Schmidhuber. Curious model-building control systems. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1458–1463, 1991.

M. Schoenauer and Z. Michalewicz. Evolutionary computation at the edge of feasibility. In *Proceedings of Parallel Problem Solving from Nature*, pages 245–254, 1996.

M. Schoenauer and S. Xanthakis. Constrained GA optimization. In *Proceedings of the International Conference on Genetic Algorithms*, pages 573–580, 1993.

J. Schonfeld. A study of mutational robustness as the product of evolutionary computation. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 1404–1411, 2007.

J. Schonfeld and D. Ashlock. Comparison of robustness of solutions located by evolutionary computation and other search algorithms. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 250–257, 2004.

C. Schumacher, M. D. Vose, and D. Whitley. The no free lunch and problem description length. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 565–570, 2001.

M. Schütz and H.-P. Schwefel. Evolutionary approaches to solve three challenging engineering tasks. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4):141–170, 2000.

H.-P. Schwefel. Experimentelle optimierung einer Zweiphasendüse. Bericht 35 des AEG-Forschungsinstituts Berlin zum Projekt MHD-Staustrahlrohr, 1968.

H.-P. Schwefel. *Evolutionsstrategie und numerische Optimierung*. PhD thesis, Technical University of Berlin, 1975.

H.-P. Schwefel. *Numerische optimierung von computer-modellen mittels der evolutionsstrategie*. Birkhäuser, 1977.

H.-P. Schwefel. Email discussion with H.-P. Schwefel, April 2009 - January 2010.

B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisability problems. In *Proceedings of the National Conference on Artificial Intelligence*, pages 440–446, 1992.

R. Shah and P. T. Ward. Lean manufacturing: Context, practice bundles, and performance. *Journal of Operations Management*, 21(2):129–149, 2003.

J. L. Shapiro. Statistical mechanics theory of genetic algorithms. In *Theoretical Aspects of Evolutionary Computing*, pages 87–108, 2001.

L. Sherman, J. Y. Ye, O. Albert, and T. B. Norris. Adaptive correction of depth-induced aberrations in multiphoton scanning microscopy using a deformable mirror. *Journal of Microscopy*, 206(1):65–71, 2002.

O. Shir and T. Bäck. Experimental optimization by evolutionary algorithms. Tutorial at the International Conference on Genetic and Evolutionary Computation, 2009.

O. M. Shir. *Niching in Derandomized Evolution Strategies and Its Applications in Quantum Control: A Journey from Organic Diversity to Conceptual Quantum Designs*. PhD thesis, University of Leiden, 2008.

O. M. Shir, M. Emmerich, T. Bäck, and M. J. J. Vrakking. The application of evolutionary multi-criteria optimization to dynamic molecular alignment. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 4108–4115, 2007.

O. M. Shir, J. Roslund, and H. Rabitz. Evolutionary multi-objective quantum control experiments with the covariance matrix adaptation. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 659–666, 2009.

H. K. Singh, A. Isaacs, T. T. Nguyen, T. Ray, and X. Yao. Performance of infeasibility driven evolutionary algorithm (IDEA) on constrained dynamic single objective optimization problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 3127–3134, 2009.

J. Singh, M. A. Ator, E. P. Jaeger, M. P. Allen, D. A. Whipple, J. E. Soloweij, S. Chowdhary, and A. M. Treasurywala. Application of genetic algorithms to combinatorial synthesis: A computational approach to lead identification and lead optimization. *Journal of the American Chemical Society*, 118(7):1669–1676, 1996.

D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

B. G. Small, B. W. McColl, R. Allmendinger, J. Pahle, G. López-Castejón, N. J. Rothwell, J. Knowles, P. Mendes, D. Brough, and D. B. Kell. Efficient discovery of anti-inflammatory small molecule combinations using evolutionary computing. *Nature Chemical Biology*, 7:902–908, 2011.

S. K. Smit, A. E. Eiben, and Z. Szlávik. An MOEA-based method to tune EA parameters on multiple objective functions. In *Proceedings of the International Joint Conference on Computational Intelligence*, pages 261–268, 2010.

P. Stagge. Averaging efficiently in the presence of noise. In *Proceedings of Parallel Problem Solving from Nature*, pages 188–197, 1998.

Y. Sugimori, K. Kusunoki, F. Cho, and S. Uchikawa. Toyota production system and kanban system materialization of just-in-time and respect-for-human system. *International journal of production research*, 15(6):553–564, 1977.

R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Proceedings of the Conference on Advances in Neural Information Processing Systems*, pages 1057–1063, 2000.

G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the International Conference on Genetic Algorithms*, pages 2–9, 1989.

G. Syswerda. A study of reproduction in generational and steady state genetic algorithms. In *Foundations of Genetic Algorithms*, pages 94–101, 1991.

G. Taguchi. *Introduction to Quality Engineering*. American Supplier Institute, 1989.

D. Thierens. An adaptive pursuit strategy for allocating operator probabilities. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 1539–1546, 2005.

A. Thompson. An evolved circuit, intrinsic in silicon, entwined with physics. In *Proceedings of the International Conference on Evolvable Systems: From Biology to Hardware*, pages 390–405, 1996a.

A. Thompson. *Hardware Evolution: Automatic design of electronic circuits in reconfigurable hardware by artificial evolution*. PhD thesis, University of Sussex, 1996b.

A. Thompson. Evolving inherently fault-tolerant systems. *Proceedings of the Institution of Mechanical Engineers, Part I*, 211(5):365–371, 1997.

A. Thompson and P. Layzell. Analysis of unconventional evolved electronics. *Communications of the ACM*, 42(4):71–79, 1999.

A. Thompson, I. Harvey, and P. Husbands. Unconstrained evolution and hard consequences. In *Towards Evolvable Hardware: The evolutionary engineering approach*, volume 1062, pages 136–165, 1996.

K. Trojanowski and Z. Michalewicz. Searching for optima in non-stationary environments. In *Proceedings of the Congress on Evolutionary Computation*, pages 1843–1850, 1999.

K. Trojanowski, Z. Michalewicz, and J. Xiao. Adding memory to the evolutionary planner / navigator. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 483–487, 1997.

A. Tuson and P. Ross. Adapting operator settings in genetic algorithms. *Evolutionary Computation*, 6(2):161–184, 1998.

S. Vaidyanathan, D. I Broadhurst, D. B. Kell, and R. Goodacre. Explanatory optimization of protein mass spectrometry via genetic search. *Analytical Chemistry*, 75(23):6679–6686, 2003.

L. Vanneschi, M. Tomassini, P. Collard, and S. Vérel. Negative slope coefficient: A measure to characterize genetic programming fitness landscapes. In *Proceedings of the European Conference on Genetic Programming*, pages 178–189, 2006.

F. Vavak, T. Fogarty, and K. Jukes. A genetic algorithm with variable range of local search for tracking changing environments. In *Proceedings of Parallel Problem Solving from Nature*, pages 376–385, 1996.

K. Veeramachaneni, K. Vladislavleva, M. Burland, J. Parcon, and U. M. O'Reilly. Evolutionary optimization of flavors. In *Proceedings of the Conference on Genetic and Evolutionary Computation*, pages 1291–1298, 2010.

M. D. Vose. Modeling simple genetic algorithms. *Evolutionary Computation*, 3 (4):453–472, 1995.

M. D. Vose. *The simple genetic algorithm: Foundations and theory*. MIT Press, 1999.

M. D. Vose and G. E. Liepins. Punctuated equilibria in genetic search. *Complex Systems*, 5:31–44, 1991.

W. E. Wallace, C. M. Guttman, K. M. Flynn, and A. J. Kearsley. Numerical optimization of matrix-assisted laser desorption / ionization time-of-flight mass spectrometry: Application to synthetic polymer molecular mass distribution measurement. *Analytica Chimica Acta*, 604(1):62–68, 2007.

T. Walters. Repair and brood selection in the traveling salesman problem. In *Proceedings of Parallel Problem Solving from Nature*, pages 813–822, 1998.

C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge University, 1989.

R. A. Watson, S. G. Ficici, and J. B. Pollack. Embodied evolution: Embodying an evolutionary algorithm in a population of robots. In *Proceedings of the Concress on Evolutionary Computation*, pages 335–342, 1999.

R. A. Watson, S. G. Ficici, and J. B. Pollack. Embodied Evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, 39(1):1–18, 2002.

D. Weuster-Botz. Experimental design for fermentation media development: Statistical design or global random search? *Journal of Bioscience and Bioengineering*, 90(5):473–483, 2000.

D. Weuster-Botz and C. Wandrey. Medium optimization by genetic algorithm for continuous production of formate dehydrogenase. *Process Biochemistry*, 30 (6):563–571, 1995.

D. Weuster-Botz, V. Pramatarova, G. Spassov, and C. Wandrey. Use of a genetic algorithm in the development of a synthetic growth medium for arthrobacter simplex with high hydrocortisone Δ^1- dehydrogenase activity. *Journal of Chemical Technology & Biotechnology*, 64(4):386–392, 1995.

D. Whitley. An executable model of a simple genetic algorithm. *Foundations of genetic algorithms*, 2:45–62, 1993.

D. Wolf, O. V. Buyevskaya, and M. Baerns. An evolutionary approach in the combinatorial selection and optimization of catalytic materials. *Applied Catalysis A: General*, 200(1-2):63–77, 2000.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1:67–82, 1997.

P. K. Wong, F. Yu, A. Shahangian, G. Cheng, R. Sun, and C.-M. Ho. Closed-loop control of cellular functions using combinatory drugs guided by a stochastic search algorithm. *Proceedings of the National Academy of Sciences*, 105:5105–5110, 2008.

A. J. Wright, D. Burns, B. A. Patterson, S. P. Poland, G. J. Valentine, and J. M. Girkin. Exploration of the optimisation algorithms used in the implementation of adaptive optics in confocal and multiphoton microscopy. *Microscopy Research and Technique*, 67(1):36–44, 2005a.

A. J. Wright, D. Burns, B. A. Patterson, S. P. Poland, G. J. Valentine, and J. M. Girkin. Exploration of the optimisation algorithms used in the implementation of adaptive optics in confocal and multiphoton microscopy. *Microscopy Research and Technique*, 67(1):36–44, 2005b.

S. Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution. In *Proceedings of the International Congress of Genetics*, pages 356–366, 1932.

B. Yang and Z. Vickers. Optimization of cheddar cheese taste in model cheese systems. *Journal of Food Science*, 69(6):S229–S236, 2004.

X. Yao and T. Higuchi. Promises and challenges of evolvable hardware. *IEEE Transactions on Systems, Man, and Cybernetics C*, 29(1):87–97, 1999.

D. Zeidler, S. Frey, K.-L. Kompa, and M. Motzkus. Evolutionary algorithms and their application to optimal control studies. *Physical Review A*, 64(2):023420, 2001.

W. Zhang. *Reinforcement Learning for Job-Shop Scheduling*. PhD thesis, Oregon State University, 1996.

W. Zhang. Phase transitions and backbones of 3-SAT and maximum 3-SAT. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, pages 153–167, 2001.

W. Zhang and T. G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1114–1120, 1995.

W. Zhang and T. G. Dietterich. High-performance job-shop scheduling with a time-delay TD(λ) network. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*, pages 1024–1030, 1996.

V. Zykov, E. Mytilinaios, B. Adams, and H. Lipson. Self-reproducing machines. *Nature*, 435(7039):163–164, 2005.

J. M. Zytkow, J. Zhu, and A. Hussam. Automated discovery in a chemistry laboratory. In *Proceedings of the National Conference on Artificial Intelligence*, pages 889–894, 1990.