On Handling Ephemeral Resource Constraints in Evolutionary Search

R. Allmendinger r.allmendinger@ucl.ac.uk
Department of Biochemical Engineering, University College London, London, WC1E 7JE, UK

J. Knowles j.knowles@manchester.ac.uk
School of Computer Science, University of Manchester, Manchester, M13 9PL, UK

Abstract

We consider optimization problems where the set of solutions available for evaluation at any given time t during optimization is some subset of the feasible space. This model is appropriate to describe many closed-loop optimization settings (i.e. where physical processes or experiments are used to evaluate solutions) where, due to resource limitations, it may be impossible to evaluate particular solutions at particular times (despite the solutions being part of the feasible space). We call the constraints determining which solutions are non-evaluable *ephemeral resource constraints* (ERCs). In this paper, we investigate two specific types of ERC: one encodes periodic resource availabilities, the other models 'commitment' constraints that make the evaluable part of the space a function of earlier evaluations conducted. In an experimental study, both types of constraint are seen to impact the performance of an evolutionary algorithm significantly. To deal with the effects of the ERCs, we propose and test five different constraint-handling policies (adapted from those used to handle 'standard' constraints), using a number of different test functions including a fitness landscape from a real closed-loop problem. We show that knowing information about the type of resource constraint in advance may be sufficient to select an effective policy for dealing with it, even when advance knowledge of the fitness landscape is limited.

Keywords

Closed-loop optimization, constrained optimization, dynamic optimization, evolutionary computation, instrument setup optimization, optimization.

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 illustrates the setup employed. Schwefel was using an early form of *evolutionary algorithm* (EA) and evaluating designs, not through simulation, but by conducting real (physical) experiments. Although 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 paper considers problems featuring experimental setups of very similar character to that of Schwefel's, nowadays commonly referred to as *closed-loop optimization problems* (Knowles, 2009). In these, genotypes to a problem (e.g. set of parameter values specifying nozzle shapes) are *planned on a computer*, but their phenotypes (e.g. an actual flashing nozzle) are *realized or prototyped and evaluated ex-silico* (e.g. relying on a physical experiment of some sort). The process of measuring the fitness

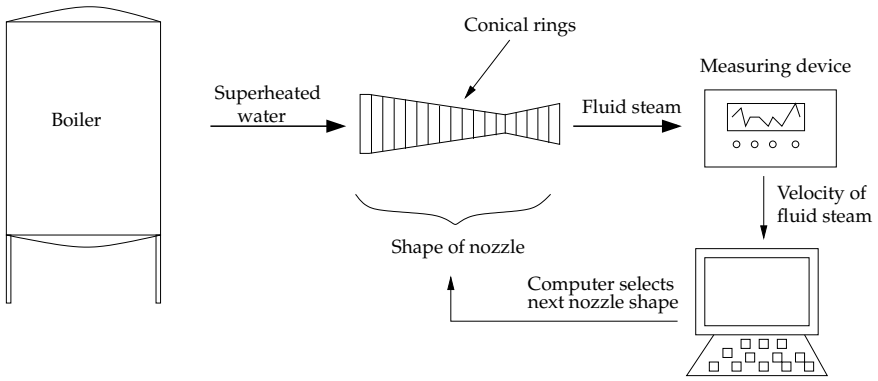

Figure 1: Schematic of the experimental setup as used by Schwefel (1968) in the shape optimization of a flashing nozzle. The nozzle consisted of a series of conical brass rings, each having its own diameter. The quality or fitness of the nozzle was measured by injecting superheated water into one side 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.

of the phenotype involves conducting a physical experiment too, as was also the case in Schwefel's setup. Applications of closed-loop problems have included shape design optimization (Rechenberg, 1973; Schwefel, 1975; Rechenberg, 2000), optimization of running industrial processes (Box, 1957), quantum control (Judson and Rabitz, 1992; Shir, 2008), drug discovery (Caschera et al., 2010; Small et al., 2011), analytical biochemistry (Vaidyanathan et al., 2003; O'Hagan et al., 2005, 2007), evolvable hardware (Thompson, 1996), food science (Herdy, 1997; Knowles, 2009), and other sciences. Many of these have used an EA approach following (Box, 1957; Schwefel, 1975; Rechenberg, 2000).[1] Interest in the use of closed-loop methods seems to be healthy; see e.g. Knowles (2009), Shir and Bäck (2009), Shir et al. (2009), Caschera et al. (2010), Bäck et al. (2010), Bedau (2010), Michalewicz (2010).

While closed-loop optimization frequently produces satisfying results, there are various unexplored resourcing issues an experimentalist may face during an iterative experimental loop. The aim of this paper is to understand how evolutionary search is affected by, and can be extended to combat, a particular *resourcing issue* in a closed-loop optimization scenario: 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 (dynamic) 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).

What is the motivation for studying ERCs? Although not reported explicitly in the literature, Schwefel and others (Heckler and Schwefel, 1978; Booker et al., 1999; Büche et al., 2002) have experienced these ERCs in practice. For instance, Schwefel had to stop experiments when brass rings he needed were not available. In another

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

problem encountered by Schwefel, the fitness of a single solution was measured by running a time-consuming simulation on a computer. During some simulations, the process ended prematurely (i.e. an execution error or exception occurred) and no fitness was returned. Finkel and Kelley (2009) give eight further references where "failure of the function evaluation has been observed in practice", indicating that this difficulty still prevails today. We are also aware of several other different types of ERCs as reported in (Allmendinger and Knowles, 2010a). Briefly, these come about in the following cases: (1) staff/operators/equipment needed for specific experiments/evaluations have limited availability, (2) consumable physical resources required to build or evaluate specific solutions may have "run out", and may have time lags between ordering and receiving them, (3) "relaxation" of a physical instrument setting is costly or takes time, so the instrument should be reused at the same setting (the setting is described by a parameter in the solution vector being optimized), (4) random machine/component breakdowns (the use of a specific machine/component is described by a parameter in the solution vector being optimized). We have come across (1) in the problem of optimizing cocoa roasting (see brief discussion in Knowles, 2009), (2) in a drug discovery problem (Small et al., 2011), and in the flashing nozzle problem encountered by Schwefel (see above), (3) in the domain of instrument setup optimization (O'Hagan et al., 2005, 2007; Jarvis et al., 2010),[2] and (4) we have not seen directly, but one can easily imagine scenarios where in say hardware evolution, a component under evolutionary selection (e.g. a transistor) fails, so that that component cannot be used in further solutions until it is replaced (or similar scenarios).

We have also investigated ERC scenarios where several different ERCs were present at the same time. For example, in a real-world problem in design optimization, resources needed to test new designs had to be ordered in advance, kept in limited (refrigerated) storage, and used within a certain time frame. This type of problem, in abstract form, has been dealt with in detail in (Allmendinger and Knowles, 2010b). When wastage of resources or time is important, we found it is necessary to schedule the resources involved in the application dynamically. A modified just-in-time policy, and another policy that predicts what the EA may wish to evaluate in the next generation, were found to perform well, with the different constraint regimes determining which of the two should be preferred.

From the above examples it is apparent that ERCOPs are static optimization problems, which is to say they have a static fitness function and a static feasible region. They are only dynamic in the sense that some candidate solutions are non-evaluable for certain periods of time (i.e. they cannot be prototyped and/or their fitness cannot be measured) due to resource limitations. In the course of this study we will give various other examples indicating how ERCs may arise in closed-loop optimization, and why they cannot reasonably be avoided if an efficient or budget-limited optimization is to be conducted. We have found that the variety of ERC types is quite rich: in addition to a temporal lack of raw materials or a sudden breakdown of a simulation program, one may also face, for example, periodic non-availabilities of skilled engineers needed to conduct experiments, or random events like machine breakdowns may also cause parts of the search space to 'disappear' temporarily.

Unlike our previous work (Allmendinger and Knowles, 2010b), here we tackle two different (and simpler) but perhaps more common types of ERCs, and investigate some

[2]Note that ERCs are not discussed specifically in (O'Hagan et al., 2005, 2007; Jarvis et al., 2010), but they existed and were circumvented using waiting or other resource-wasteful strategies. In Section 8 we will look at the studies reported in O'Hagan et al. (2005, 2007) in more detail.

general policies for dealing with them.

The rest of this paper is organized as follows. In the next section, for completeness reasons we briefly recall the general problem definition of an ERCOP. Section 3 discusses the relationship between ERCOPs and other types of optimization problems to set this study in proper context. The two real-world ERC types on which we test our policies are outlined in Section 4, and the policies themselves are described in Section 5. Before we proceed with the experimental analysis in Section 7, we describe in Section 6 the choice of test functions, the base algorithm on which we augment the policies, and give all parameter settings. In Section 8 we then present a case study that illustrates one way in which one might select a suitable policy for an ERCOP with largely unknown search space properties, which is a common situation in the real world. Finally, in the concluding section we draw together the findings from the experimental analyses and discuss directions for further research.

2 Ephemeral Resource-Constrained Optimization Problems (ERCOPs)

Ephemeral resource constraints (ERCs) are temporary limitations on the set of solutions that are available for evaluation *during* an optimization procedure. To define them formally we begin with a standard optimization problem, and add to it a 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

$$\begin{aligned} &\text{maximize} \;\; y = f(\mathbf{x}) \\ &\text{subject to} \;\; \mathbf{x} \in X, \end{aligned} \tag{1}$$

where $\mathbf{x} = (x_1, ..., x_l)$ 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, $a : \phi(t) \mapsto \mathbf{x}_t$, where the *search history* $\phi(t) = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_{t-1}, y_{t-1})\}$ denotes the time-ordered set of solutions visited until time step $t-1$, and $\mathbf{x}_i$ 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 in his flashing nozzle problem (see Section 1).

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

$$y_t = \begin{cases} f(\mathbf{x}_t) & \text{if } \mathbf{x}_t \in E(\sigma_t) \subseteq X \\ \text{null} & \text{otherwise,} \end{cases} \tag{2}$$

where $E(\sigma_t)$ represents the *set of evaluable solutions* (or *evaluable search region*) at time step t. In our case, $E(\sigma_t)$ is defined by a set of *schemata* into which solutions have to fall in order to be evaluable.[3] The set $E(\sigma_t)$ may *change over time* depending on a set

[3]We employ the classical notion of schemata, as used in the context of binary-coded genetic algorithms (Holland, 1975), to describe availability of resources. Section 4.1 will introduce the notion of schemata

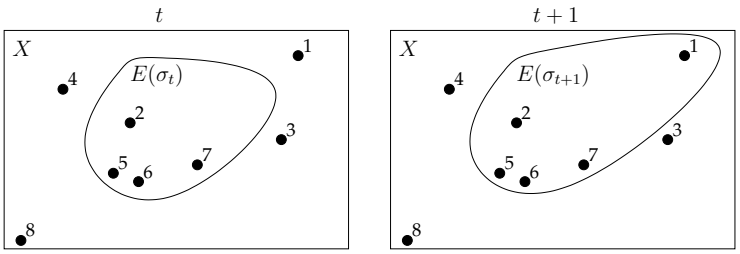

Figure 2: A schematic diagram showing how a population consisting of the solutions 1 to 8 might be distributed across the feasible search space X and the evaluable search space $E(\sigma_t)$. At time step t, only the solutions 2,5,6, and 7 can be evaluated, while the solutions 1,3,4, and 8 must be repaired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

of problem-specific and time-evolving parameters σ_t. The availability of resources required for the evaluation of solutions depends on these parameters. Hence, 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. The ERCs specify how exactly the set $E(\sigma_t)$ changes depending on the parameter set σ_t.

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, $E(\sigma_t)$, that may change. In this context, *repairing a solution* means to modify the genotype of a solution $\mathbf{x} \in X$ that is not in $E(\sigma_t)$ such that it is forced into $E(\sigma_t)$; i.e. the outcome of a repairing step is a solution that falls into all schemata that define $E(\sigma_t)$ (assuming that the schemata are *non-contradictory*; i.e. $E(\sigma_t) \cap X \neq \emptyset$).

Compared to standard (dynamic) constraints, the meaning of ERCs is different: a solution $\mathbf{x}$ that violates an ERC at time t, or $\mathbf{x} \notin E(\sigma_t)$, is not *infeasible* but *non-evaluable* at time step t. That is, the experiment that is associated with $\mathbf{x}$ cannot be conducted, thus causing the fitness of solution $\mathbf{x}$ at time t to be undefined (or null).[4] Figure 2 illustrates the interaction between $E(\sigma_t)$ and X commonly present in an ERCOP.

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. 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.

and its relationship to ERCs in more detail, and also indicate how this notion may be applied to non-binary search spaces.

[4]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.

3 Relationship of ERCOPs to other types of optimization problems

As mentioned in the introductory section, dynamic resource constraints in the sense meant here have not to date been raised much in the literature. In fact, apart from discussions with Schwefel as well as our own collaborative work (Knowles, 2009; Allmendinger and Knowles, 2010a,b), ERCs have not been considered in published work to the best of our knowledge.

However, of course, much other related work informs our research, and we find that ERCOPs and closed-loop optimization are related to several other areas. Traditionally, closed-loop optimization problems are dealt with using statistical methods referred to as *experimental design* or *design of experiments* (DoE) (Montgomery, 1976; Box et al., 2005). The focus of DoE is rather on low-dimensional search spaces with the aim to obtain statistically robust results in as few evaluation steps as possible and to explain them in terms of a model. ERCOPs, by contrast, feature often higher-dimensional search spaces and one is ultimately interested in finding a single optimal solution. Nevertheless, we believe that closed-loop evolution methods should draw on DoE, particularly in areas like noise-handling, replication, blocking and so on. 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 will see later in the commitment relaxation ERCs, ERCOPs can have a time-linkage aspect to them in the sense that ERCs arise due to previously made decisions. We find an interesting parallel in some work on *online (dynamic) optimization problems* (Borodin and El-Yaniv, 1998; Bosman and Poutré, 2007), which exhibits time-linkage too. Nevertheless, there are clear and important differences between our problem formulation and those considered in these studies: their aim is to optimize a cumulative score over some period of time, whereas ours is to find a single optimal (and ultimate) solution. Similarly, we find ERCOPs to be materially different to traditional *dynamic optimization problems* (Branke, 2001) because the objective space in ERCOPs does not change over time and thus the optimal solution does not need to be tracked. Despite this core difference, we believe that some policies, as using memory, can carry over from dynamic optimization into our work considering dynamic resource constraints.

Traditional *constrained optimization* (Michalewicz and Schoenauer, 1996; Nocedal and Wright, 1999; Coello, 2002) is also an important related area, which can inspire some methods for handling resource constraints (e.g. penalty methods). However, the fact that ERCs prevent the evaluation of solutions that are otherwise feasible makes them materially different from standard constraints, including dynamic constraints (Nguyen, 2011). A practical consequence of the difference between ERCs and dynamic constraints is that while an algorithm optimizing subject to ERCs may, during the optimization process, report a currently non-evaluable solution as its best-so-far solution, an algorithm optimizing subject to dynamic constraints should not report an infeasible solution as its current best-so-far solution. Also, while the optimal solution does not change in the presence of ERCs, it is likely to do so when optimizing subject to standard dynamic constraints. The implication of this is that one could terminate the optimization in the presence of ERCs once a solution of desired quality is found, while this should not be done with standard dynamic constraints.

4 Two Specific ERC Types

In this section we introduce two ERC types that we encountered in our own collaborative work and that seem to be common in real-world applications: commitment

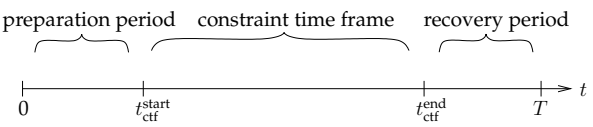

Figure 3: An illustration of how the available optimization time T can be divided into the preparation period $0 \leq t < t^{\text{start}}_{\text{ctf}}$, the constraint time frame $t^{\text{start}}_{\text{ctf}} \leq t < t^{\text{end}}_{\text{ctf}}$, and the recovery period $t^{\text{end}}_{\text{ctf}} \leq t \leq T$.

relaxation ERCs and periodic ERCs.[5] Before we define both ERC types, we introduce three elements that are common to both (and other) ERC types. These elements are the activation period, the constraint time frame and the constraint schema.

4.1 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), or something else. Here, 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.[6] 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 $E(\sigma_t) \subseteq X, \forall t \in$ ctf, and $E(\sigma_t) = X, \forall t \notin$ 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 the *total optimization time*) is the *preparation period* and *recovery period*, respectively (see Figure 3). 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. A schema H represents a particular subset of solutions that share some common properties. For instance, consider solution vectors to be binary strings of length $l = 5$ with each solution bit representing two resource choices (0 and 1 or A and B) to be optimized over. Now, for example, if we assume that only resources 1 and 0 are available for bit position 2 and 5, respectively, whilst all resources are available for the other bit positions, then the constraint schema $H = (*1**0)$ would describe the set of evaluable solutions $E(\sigma_t)$; the $*$ is a wildcard symbol which means that a bit position can have any possible value (thus in the binary case either value 0 or 1). A general property of a schema is its *order* $o(H)$, representing the number of defined bit positions (Reeves and Rowe, 2003); for the above example we have $o(H) = 2$. In the presence of multiple constraints ERC_i, solutions have to fall into the union of the schemata $\mathbf{x} \in \bigcup_i H_i$ associated with the constraints. In this study we consider discrete search spaces, mainly of pseudo-Boolean nature or $X \in \{0, 1\}^l$. In non-discrete spaces, we might require $E(\sigma_t)$ to restrict solution parameters to lie within or out of certain parameter value ranges rather than to

[5] In this work we consider these two ERC types only but we are aware of other types of which some are defined in the technical report (Allmendinger and Knowles, 2010a).

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

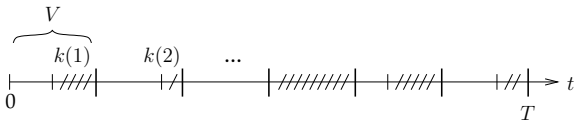

Figure 4: An illustration of how a commitment relaxation ERC may partition the optimization time into epochs of length V, and how it may be potentially activated. The activation period $k(j)$ during the jth epoch is represented by the dashed part.

take specific parameter values. In this case, ERCs could be defined in terms of functions over the input vector space, and corresponding inequality/equality conditions, i.e., using standard constrained optimization notation, except that the trigger(s) of the constraint(s) also need to be specified.

4.2 Commitment relaxation ERCs

A *commitment relaxation ERC* commits (forces) 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.

We refer to the period of time during which some variable(s) setting (or schema) H is forbidden from changing as an *epoch*, and denote its duration by V. We 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 illustrates the partition of the optimization time into epochs, and a possible distribution of activation periods. From the figure it is 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.

The corresponding implementation of a commitment relaxation ERC is defined by Algorithm 1. The method $commRelaxERC(t_{\text{ctf}}^{\text{start}}, t_{\text{ctf}}^{\text{end}}, V, H, \mathbf{x}, t)$ takes as input the parameters $t_{\text{ctf}}^{\text{start}}, t_{\text{ctf}}^{\text{end}}, V$, and H, a candidate solution $\mathbf{x}$ that is to be checked for evaluability, and the current (global) time step t. The output is a boolean value indicating whether $\mathbf{x}$ is evaluable or not (in our EA, shown in Algorithm 3, we call the method at Line 21).[7] The method maintains two local variables, $last\_activation$ and k, required to

[7] We will describe the EA in more detail in Section 6.1.

Algorithm 1 Implementation of a commitment relaxation ERC

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

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 $commRelaxERC(t^{\text{start}}_{\text{ctf}}, t^{\text{end}}_{\text{ctf}}, V, H)$. 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: 1) a solution is non-evaluable if it violates at least one ERC, 2) a repaired solution has to satisfy all activated ERCs and not only the ones that were violated, and 3) it needs to be checked whether a repaired solution activates an ERC that was not activated before. This extension will be considered later in the case study, which we present in Section 8.

4.3 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 5). 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 2. The method $perERC(t^{\text{start}}_{\text{ctf}}, t^{\text{end}}_{\text{ctf}}, k, P, H, \mathbf{x}, t)$ takes as input the parameters $t^{\text{start}}_{\text{ctf}}, t^{\text{end}}_{\text{ctf}}$, k, P, and H, a candidate solution $\mathbf{x}$ that is to be checked for evaluability, and the current (global) time step t. The output is a boolean value indicating whether $\mathbf{x}$ is evaluable or not (in our EA, shown in Algorithm 3, we call the method at Line 21).

In future, we will denote periodic ERCs by $perERC(t^{\text{start}}_{\text{ctf}}, t^{\text{end}}_{\text{ctf}}, k, P, H)$. A potential extension of a periodic ERC is that the period length and the activation period refer

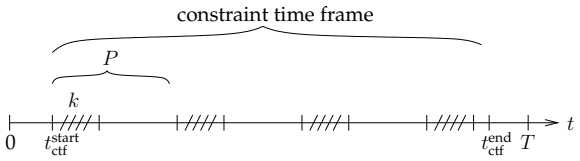

Figure 5: An illustration of a periodic ERC $perERC(t_{\text{ctf}}^{\text{start}}, t_{\text{ctf}}^{\text{start}}, k, P, H)$. The ERC is activated every P time steps for an activation period of always k time steps.

Algorithm 2 Implementation of a periodic ERC

1: $perERC(t_{\text{ctf}}^{\text{start}}, t_{\text{ctf}}^{\text{end}}, k, P, H, \mathbf{x}, t)\{$
2: **if** $t \in$ ctf $\wedge$ $(t - t_{\text{ctf}}^{\text{start}})$ mod $P < k$ $\wedge$ $\mathbf{x} \notin H$ **then**
3: return *false* // $\mathbf{x}$ is not evaluable
4: **else**
5: return *true* $\}$ // $\mathbf{x}$ is evaluable

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).

5 Constraint-Handling Policies for ERCOPs

This section introduces five constraint-handling policies for dealing with non-evaluable solutions arising due to ERCs. The policies are applicable not only to the above ERC types but (in similar form) also to other ERCs. Three of the policies (forcing, regenerating, and the subpopulation strategy) apply repairing (i.e. modify the genotype of a solution) and two (waiting and penalizing) avoid it in order to prevent drift-like effects in the search direction. Note, although some of the policies have been used to cope with standard constraints (we will point this out where applicable), the effect of them when handling ERCs is unknown. In the description of the policies we assume that multiple ERCs of a particular ERC type 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 $\exists \mathbf{x} \in \bigcup_i H_i$. 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 policy *forces* a non-evaluable solution $\mathbf{x}$ 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. Strategies of this kind have been used previously e.g. in constrained combinatorial optimization (Liepins and Potter, 1991).

A drawback of this policy is that enforcing changes in decision variable values may destroy potentially good genotypes. Later, we will investigate this aspect more closely.

2. Regenerating: The aim of this policy, which is similar to the *death penalty* approach (Schwefel, 1975) originating from the evolution strategies community, is to overcome the potential drawback of forcing. In fact, as the name of the policy 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;[8] 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 policy is that for large L it can be computationally expensive, while for small L, it could be that it reduces often to the forcing policy.

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 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 policies 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.

A drawback of the subpopulation strategy is that if we have more than one ERC, i.e. $r > 1$, then the number of subpopulations needed 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 activated ERCs.

4. Waiting: This policy 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.[9] This policy is identical to the way Schwefel (1968) handled unavailable conical rings in his flashing nozzle design problem (see Section 1).

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

5. Penalizing: Like waiting, this policy 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 policy within an elitist EA, and 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 policy is popular in the genetic algorithm (GA) community,

[8] Notice that the Hamming distance between a solution $\mathbf{x}$ and a schema H is calculated based on the order-defining bits of H only.

[9] NB to implement the freezing period, the global time counter t is set directly to the end of the longest activation period of all currently violated ERCs (i.e. we do not actually submit null solutions to bridge the period); this step is realized in Line 27 of Algorithm 3.

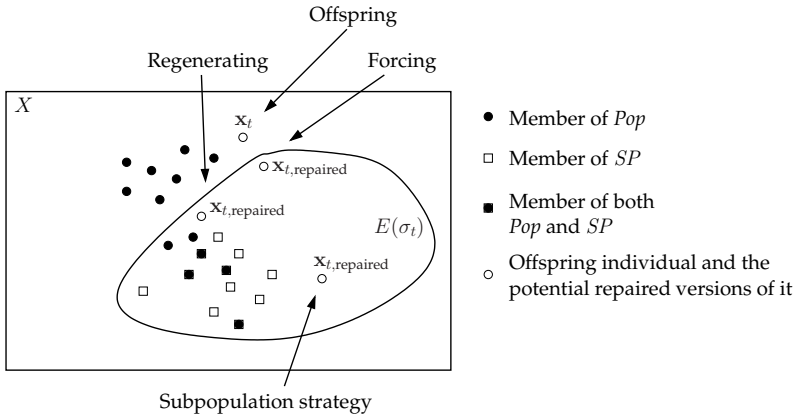

Figure 6: A depiction of the current population *Pop* (filled circles and squares) and an offspring individual $\mathbf{x}_t$, which is feasible but not evaluable (because it is in X but not in $E(\sigma_t)$). Solutions indicated by the filled squares coexist in both the actual EA population *Pop* and the population SP maintained by the subpopulation strategy. The three solutions $\mathbf{x}_{t,\text{repaired}}$ indicate repaired solutions that might have resulted after applying one of the three 'repairing' policies to $\mathbf{x}_t$: while forcing simply flips incorrectly set bits of $\mathbf{x}_t$ and thus creates a repaired solution that is as close as possible to $\mathbf{x}_t$ but not necessarily fit, regenerating creates a new solution in $E(\sigma_t)$ 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 $E(\sigma_t)$.

where it can be regarded as a *static penalty function method* (Coello, 2002).

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 evaluated solutions 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 6 visualizes how the policies forcing, regenerating, and the subpopulation strategy may repair a non-evaluable solution.

6 Experimental Setup

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

6.1 Evolutionary algorithm

We augment the constraint-handling policies on an EA with a $(\mu + \lambda)$-ES reproduction scheme, an elitist approach, which we believe would be generally applicable in this domain. The algorithm also uses binary tournament selection (with replacement) for parental selection, uniform crossover (Syswerda, 1989), bit flip mutation, and does not check whether a currently non-evaluable solution has been evaluated previously,

Algorithm 3 Generational EA with constraint-handling policies

Require: ERC_1,...,ERC_r (set of ERCs), f (objective function), T (time limit), μ (parent population size), λ (offspring population size), #Policy (number of selected constraint-handling policy; see Section 5)

$t = 0$ (global variable representing the current time step), $Pop = \emptyset$ (current population), $OffPop = \emptyset$ (offspring population)
while $|Pop| < \mu \ \wedge \ t < T$ **do**
  generate solution $\mathbf{x}$ at random
  $\mathbf{x}$ = functionWrapper($\mathbf{x}, t$)
  $Pop = Pop \cup \{\mathbf{x}\}$; t++
while $t < T$ **do**
  $OffPop = \emptyset$
  repeat
    generate two offspring $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ by selecting two parents from Pop, and then recombining and mutating them
    $OffPop = OffPop \cup \{\mathbf{x}^{(1)}\}$
    if $|OffPop| < \lambda$ **then**
      $OffPop = OffPop \cup \{\mathbf{x}^{(2)}\}$
  until $|OffPop| = \lambda$
  for $i = 0$ to λ **do**
    if $t < T$ **then**
      $\mathbf{x}_i$ = functionWrapper($\mathbf{x}_i, t$) // $\mathbf{x}_i$ represents the ith solution of $OffPop$
      t++
  form new Pop by selecting the best μ solutions from the union population $Pop \cup OffPop$

functionWrapper($\mathbf{x}, t$){
$y_t = $ null
if $\mathbf{x}$ satisfies the ERCs ERC_1,...,ERC_r **then**
  $\mathbf{x}_t = \mathbf{x}$; $y_t = f(\mathbf{x}_t)$
else
  if #Policy $= 1 \vee$ #Policy $= 2 \vee$ #Policy $= 3$ **then**
    $\mathbf{x}_t = $ repair($\mathbf{x}, t$); $y_t = f(\mathbf{x}_t)$ // apply the repairing strategy selected
  if #Policy $= 4$ **then**
    $t = t + \delta$; $\mathbf{x}_t = \mathbf{x}$ // δ is the number of time steps we have to wait until the activation periods of all ERCs that are currently violated by $\mathbf{x}$ have passed
    if $t < T$ **then**
      $y_t = f(\mathbf{x}_t)$
  if #Policy $= 5$ **then**
    $\mathbf{x}_t = \mathbf{x}$; $y_t = c$ // c is a constant, representing poor fitness
return $\mathbf{x}_t$ and y_t}

i.e. identical solutions may be evaluated multiple times. Algorithm 3 shows the pseudocode of the EA. We use the *function wrapper* as the interface between the EA and the ERCOP, allowing us to conveniently manage the ERCs, constraint-handling policies, and evaluation of solutions; calling this wrapper is similar to calling the objective function f in a standard optimization problem. Notice from the pseudocode that we are using a Lamarckian population update in this paper; i.e. a repaired solution is

used for evaluation and also replaces the original solution. Additional performance-enhancing mechanisms commonly used in EAs, such as diversity preservation techniques (Goldberg and Richardson, 1987; Mahfoud, 1995) or adaptive parameter control (Davis, 1989), may affect the results, but are not considered here.

6.2 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 policy 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 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 8, we do use data and constraints from a real closed-loop problem).

Our set of selected test functions comprises: 1) OneMax, 2) a competing optima problem, TwoMax, and 3) several MAX-SAT problem instances with many local optima. In the subsequent case study we will also see a variant of NK landscapes ($NK\alpha$ landscapes) being used; we will introduce this test function here too. Of course, we cannot guarantee that the test functions mimic real (closed-loop) problems, but a diverse set of functions as used here should be sufficient to draw some tentative conclusions about the effects of ERCOPs generally, depending on results observed.

OneMax: For a binary solution vector $\mathbf{x} \in \{0,1\}^l$, the unimodal *OneMax* function (Mühlenbein and Schlierkamp-Voosen, 1993) takes the sum over the bit values of all bit positions

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

and has its optimum $f = l$ for the bit string consisting only of 1-bits.

TwoMax: Our bimodal *TwoMax* function contains one local and one global optimal solution, which are represented by solutions consisting only of 0-bits and 1-bits, respectively (see Figure 7). Similar to Pelikan and Goldberg (2000), we achieve this bimodal structure by having a steeper slope leading to the solution consisting only of 1-bits. Hence, if $\#1\text{s}$ denotes the number of 1-bits in a solution vector $\mathbf{x}$ and $b > 1$ the factor by which the global optimal solution shall be fitter than the local optimal solution, then the TwoMax function is defined by

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

MAX-SAT: Given a collection of clauses involving l binary variables $x_i, i = 1, ..., l$, the *satisfiability* (SAT) problem asks whether or not there is a variable assignment such that all clauses are simultaneously satisfied (Hansen and Jaumard, 1990). A generalization of the SAT decision problem is the *maximum satisfiability* (MAX-SAT) problem, which asks for a variable assignment that satisfies the maximum number of clauses (Qasem and Prügel-Bennett, 2010). MAX-SAT and SAT problems are of high practical relevance as many challenging real-world problems can be efficiently formulated in SAT form (De

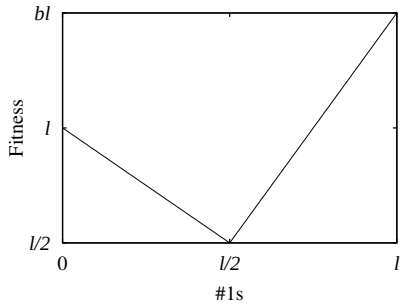

Figure 7: A TwoMax function with a local optimal solution at #1s = 0 and a global optimal solution at #1s = l. The parameter $b > 1$ specifies the factor by which the global optimal solution shall be fitter than the local optimal solution.

Jong and Spears, 1989), e.g. hardware and software verification problems, and routing in FPGAs. Another reason for choosing this problem is the presence of a *backbone* (the backbone of a MAX-SAT instance is the schema into which all the optimal solutions fall), which is a convenient property when analyzing the impact of ERCs on search.

We consider several (ten) benchmark instances of a uniform random 3-SAT problem, which can be downloaded online.[10] The instances have $l = 50$ variables and 218 clauses and are satisfiable. Similar to (Hansen and Jaumard, 1990; Qasem and Prügel-Bennett, 2010), we treat the 3-SAT instances as MAX-3-SAT optimization problems with fitness calculated as the proportion of satisfied clauses. We also conducted experiments on other challenging multimodal test problems, $NK\alpha$ landscapes (Kauffman, 1989), which we introduce next.

$NK\alpha$ **landscapes:** The general idea of the $NK\alpha$ model (Hebbron et al., 2008) is to extend Kauffmans 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. As α increases, an increasing influence is given to a minority of variables, while, for $\alpha = 0$, the $NK\alpha$ model reduces to Kauffman's original NK model with neighbors being selected at random. This model has already been used previously to analyze certain aspects of real-world closed-loop problems; as an example see (Thompson, 1996).

[10] `http://people.cs.ubc.ca/~hoos/SATLIB/benchm.html`; the names of the (ten) instances are "uf50-218/uf50-0*.cnf", where * is 1,2,4,6,8,11,19,22,24 and 25. These instances have a backbone with an order of 40 or greater, allowing us to analyze different ERC setups in the experimental study; in this study, we will run our EA on each instance for 50 runs to obtain the average performance on this problem type.

Table 1: EA parameter settings.

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

Table 2: Parameter settings of constraint-handling policies.

Policy	*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

6.3 Parameter settings

The parameter settings of the EA and the policies are given in Table 1 and 2, respectively. The settings of the test functions are outlined in Table 3. We choose these search space sizes l (15, 30 and 50) as they correspond to typical search space sizes we have seen — e.g. a drug combinations problem with a library of about 30 drugs as used by Small et al. (2011). The reason for setting the scaling factor b so small is that it makes the problem more challenging due to the low selection pressure to climb up the optimal slope. The optimization times T (see Table 3) are set such that we can assess both positive and negative effects of an ERC on the convergence speed and the solution quality obtained at the end of an algorithm run. To analyze the impact of the preparation and recovery time we will consider also different settings for T, but this will be pointed out where applicable.

Any results shown are average results across 500 independent algorithm runs. To allow for a fair comparison of the policies, we use a different seed for the random number generator for each EA run but the same seeds for all policies. This allows us to apply a repeated-measures statistical test, the Friedman test (Friedman, 1937), to investigate significant performance differences between policies.

7 Experimental Study

The performance of a policy 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 we consider the following aspects: 1) what genetic material represented by a constraint schema H needs to be introduced into a population to cause a performance impact; 2) 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; 3) at what stage during a run does it need to be introduced to yield a performance impact; and 4) the effects of the preparation and recovery durations. We give detailed observations

Table 3: Parameter settings of test functions f.

Test function f	*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

on these effects here, and summarise the key findings in Section 9.

7.1 Commitment relaxation ERC

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 instances, we represent a poor bit by flipping a randomly selected bit from the backbone of an instance;[11] 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 a backbone, and 0-bits to refer to their complements.[12]

Figure 8 shows how the final average best solution fitness is affected for the different policies on OneMax. The results obtained on TwoMax and the MAX-SAT instances are very similar and are shown in Appendix A. 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 policy 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 $4 < o(H) < 7$ in the top left plot).
- With respect to the *order of the constraint schema* $o(H)$ (top left plot), we observe that

[11] We identified the backbones of our MAX-SAT instances from optimal solutions obtained from running a generational GA for 1000 generations, 500 times independently.

[12] 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 policies will be optimizing subject to the same constraint schemata.

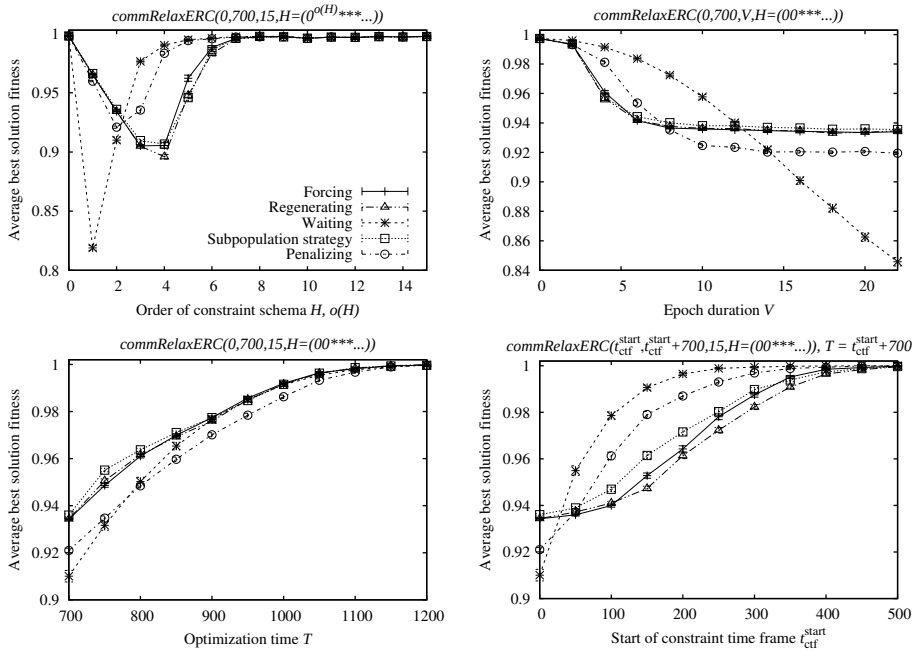

Figure 8: Plots showing the average best solution fitness found and its standard error on OneMax as a function of the order of the constraint schema $o(H)$ (top left), the epoch duration V (top right), the optimization time T (bottom left), and the start of the constraint time frame $t_{\text{ctf}}^{\text{start}}$ (bottom right). Note, while the optimization time in the top plots is fixed to $T = 700$, as specified in Table 3, the parameter T varies in the bottom plots. For each setting shown on the abscissa, a Friedman test (significance level of 5%) has been carried out. In the top left plot, waiting performs best in the range $2 < o(H) < 6$, while, in the top right plot, it performs best in the range $2 < V < 12$ with the subpopulation strategy being best in the range $V > 12$. In the bottom left plot, the subpopulation strategy performs best for $T = 750$, while, in the bottom right plot, waiting performs best in the range $0 < t_{\text{ctf}}^{\text{start}} < 300$. There is no clear winner for the other settings.

there is a value that has the largest negative effect on the optimization; it is around 4 for the 'repairing' policies (forcing, regenerating or the subpopulation strategy), and lower for penalizing and waiting.

The non-monotonic performance impact on the repairing policies, and partially on penalizing, is due to two competing forces: 1) the probability of activating a constraint, which decreases exponentially with $o(H)$, and 2) 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 policies 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 $0 < V < 14$) 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 policies 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 policies (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.

Figure 9 analyzes how constraint schemata that represent genetic material of different qualities affect the performance obtained with the constraint-handling policies. Although ERCs can have large effects on performance, one can see from the plots that the majority of the constraint schemata do not have an 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 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 policies 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 policies 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 10 — which compares the performance of penalizing against the one obtained with waiting (left plot) and the subpopulation strategy (right plot) — it is apparent that the performance differences between the policies observed previously is also maintained largely (as we go up the rows of squares).

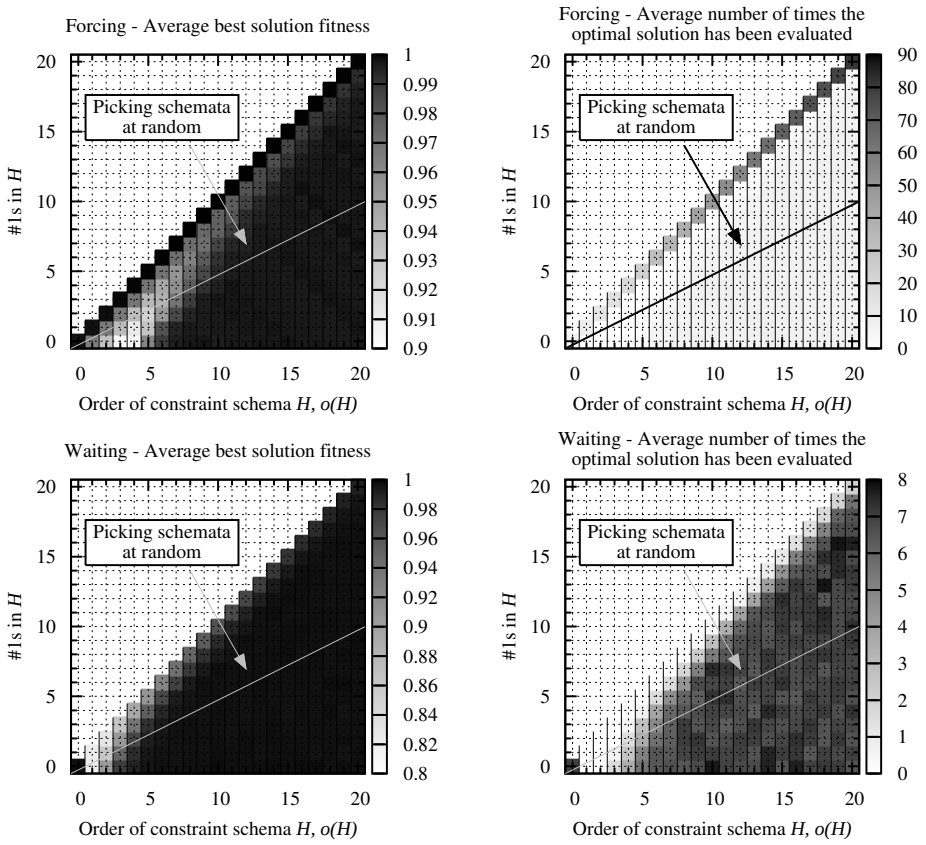

Figure 9: Plots showing the average best solution fitness obtained (left) and the average number of times the optimal solutions has been evaluated during the optimization (which is an indication of the convergence speed) (right) by forcing (top) and waiting (bottom) on OneMax as a function of the order 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 $o(H) = \#1\text{s} = 0$.

On the MAX-SAT instances, the range of constraint schemata causing a performance impact is smaller; this is apparent from Figure 11. From the plot we observe that while again low-order schemata affect the performance significantly, higher-order 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 $T = 800$ time steps.

7.2 Periodic ERC

With the insights we gained about the policies 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.

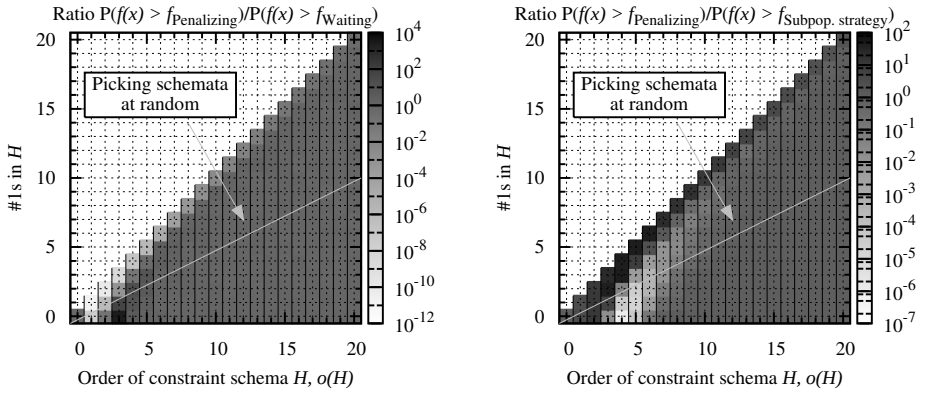

Figure 10: Plots showing the ratio $\mathrm{P}(f(x) > f_{\text{Penalizing}})/\mathrm{P}(f(x) > f_{\text{Waiting}})$ (left) and $\mathrm{P}(f(x) > f_{\text{Penalizing}})/\mathrm{P}(f(x) > f_{\text{Subpop. strategy}})$ (right) on OneMax as a function of the order of the constraint schema $o(H)$, and the 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 policy $*$. If $\mathrm{P}(f(x) > f_*)/\mathrm{P}(f(x) > f_{**}) > 1$, then policy $**$ is able to achieve a higher average best solution fitness than policy $*$ 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 $*$.

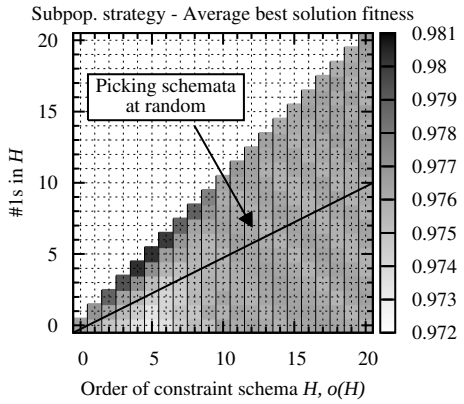

Figure 11: A plot showing the average best solution fitness obtained by the subpopulation strategy on several MAX-SAT instances as a function of the order of the constraint schema $o(H)$, and the number of order-defining bits in H set correctly for the ERC $commRelaxERC(0, 800, 15, H)$. The straight line represents the expected performance when picking a schema (i.e. the order-defining bits and their values) with a particular order at random.

Figure 12 shows how the performance of the different policies 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 instances are similar and are shown in Appendix B. In comparison to commitment relaxation ERCs,

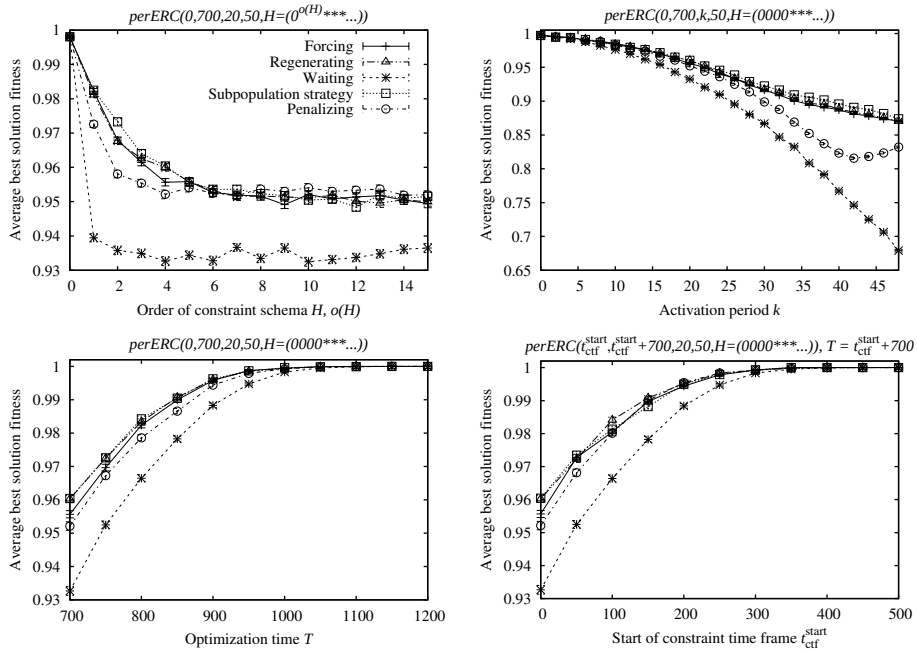

Figure 12: Plots showing the average best solution fitness found and its standard error on OneMax as a function of the order of the constraint schema $o(H)$ (top left), the activation period k (top right), the optimization time T (bottom left), and the start of the constraint time frame $t_{\text{ctf}}^{\text{start}}$ (bottom right). For each setting shown on the abscissa, a Friedman test (significance level of 5%) has been carried out. In the top left plot, the subpopulation strategy performs best for $o(H) = 2$; there is no clear winner for the other settings.

the main difference we observe from Figure 12 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 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 13 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 14), 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

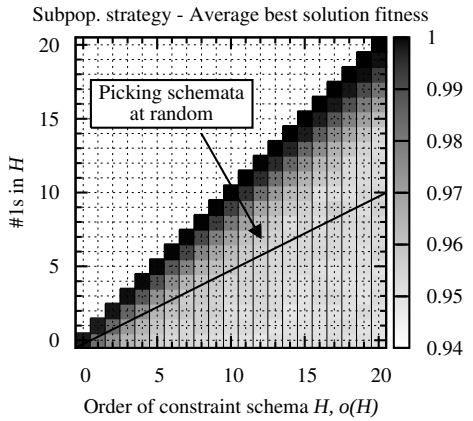

Figure 13: A plot showing the average best solution fitness obtained by the subpopulation strategy on OneMax as a function of the order of the constraint schema $o(H)$, and the number of order-defining bits in H with value 1 for the ERC $perERC(0, 700, 20, 50, H)$. The straight line represents the expected performance when picking a schema (i.e. the order-defining bits and their values) with a particular order at random.

plot of Figure 14).

On the MAX-SAT instances, 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 policies, 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 B.

8 Case Study

In this section we demonstrate one way for selecting a suitable constraint-handling policy for an real-world application involving ERCs. For this we use the same experimental setup as used in the instrument configuration application described by O'Hagan et al. (2005, 2007). We now give a more detailed description of this application.

Application description: The application is concerned with optimizing the configuration parameters of a gas-chromatography mass spectrometer instrument so as to maximize its ability to separate and detect a complex (biological) sample. Resource constraints arise here because certain parameters of the instrument configuration cannot be changed widely from one experiment to the next, without incurring a severe "change-over cost" associated with having to clean parts of the instrument (Dunn, 2011).[13] The problem is defined over $l = 15$ integer variables and a total search space of 7.32×10^9 configurations. 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 (to be maximized). Two ERCs are used to model the parameters (here related to oven temperatures of the instrument) that must be prohibited from varying

[13]This particular ERC was avoided by O'Hagan et al. (2005, 2007) by artificially reducing the ranges of these parameters, which may have compromised the optimization to a degree.

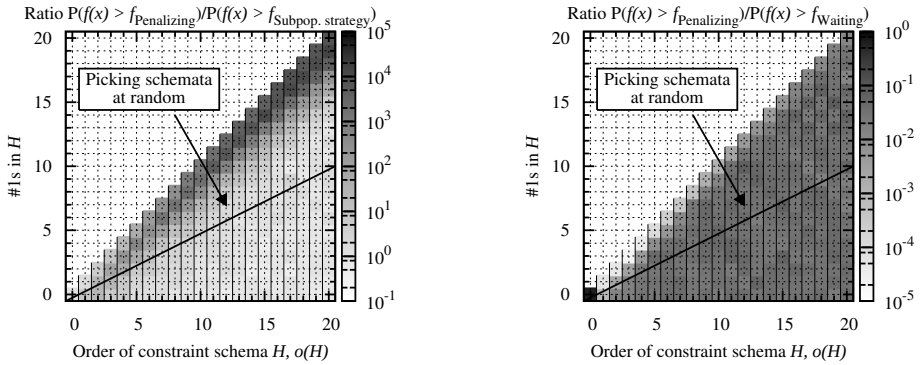

Figure 14: Plots showing the ratio $\mathrm{P}(f(x) > f_{\text{Penalizing}})/\mathrm{P}(f(x) > f_{\text{Subpop. strategy}})$ (left) and $\mathrm{P}(f(x) > f_{\text{Penalizing}})/\mathrm{P}(f(x) > f_{\text{Waiting}})$ (right) on OneMax as a function of the order of the constraint schema $o(H)$, and the 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 policy $*$. If $\mathrm{P}(f(x) > f_*)/\mathrm{P}(f(x) > f_{**}) > 1$, then policy $**$ is able to achieve a higher average best solution fitness than policy $*$ 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 $*$.

over too wide a range.

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 (which cause the ERCs to arise). 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 optimizer or constraint-handling policy for the above described ERCOP. A 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 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 6.2, and also provided the settings of the problem parameters N, K, and α in Table 3.[14] The (four) selected settings allow us to cover landscapes featuring different degrees of epistasis and topologies. To cope with the integer representation, we need to modify the mutation operator, which shall now select a random setting from

[14]Note that $NK\alpha$ landscapes are binary problems by default. Transforming them to account for an integer representation is straightforward and involves the generation of a^K different fitness values for each neighbourhood, where a is the alphabet size.

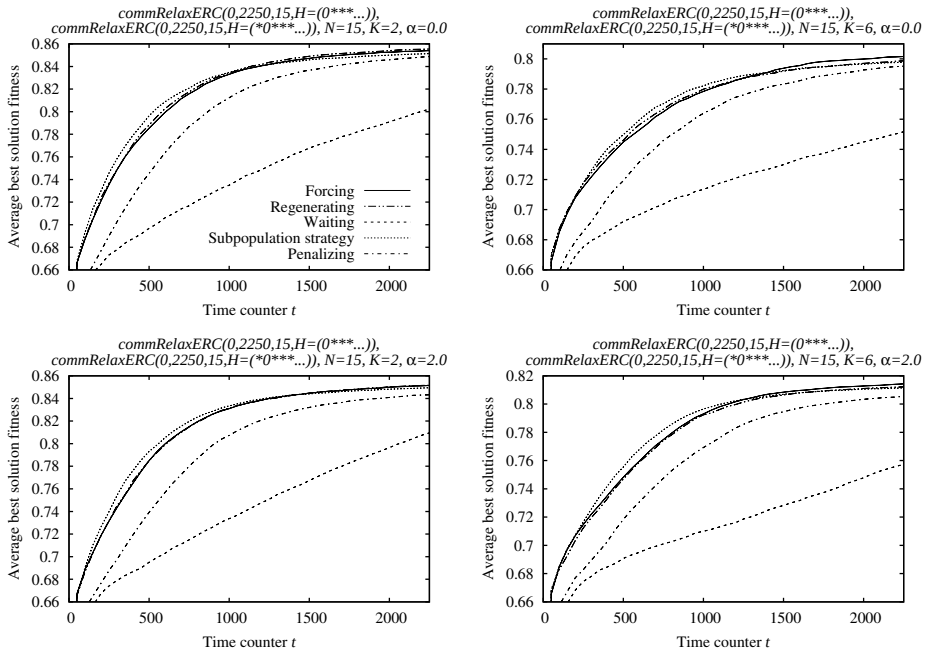

Figure 15: Plots showing the average best solution fitness obtained on $NK\alpha$ landscapes with $N = 15$ and $K = 2, \alpha = 0.0$ (top left), $K = 6, \alpha = 0.0$ (top right), $K = 2, \alpha = 2.0$ (bottom left), and $K = 6, \alpha = 2.0$ (bottom right) as a function of the time counter t; results 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 $commRelaxERC(0, 2250, 15, H = (*0 * * * ...))$. We have carried out a Friedman test (significance level of 5%) for each of the four $NK\alpha$ landscapes at time step $t = 2250$. In the top and bottom right plot, forcing achieves the best performane at the end of the search among the policies; there is no clear winner on the other two $NK\alpha$ landscapes.

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 3).

Figure 15 shows the average best solution fitness obtained by the constraint-handling policies on the four $NK\alpha$ 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 policy (forcing, regenerating, or the subpopulation strategy) should be clearly favored over a waiting or penalizing policy. 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 policies tends to increase with K and/or α. The waiting policy 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 perfor-

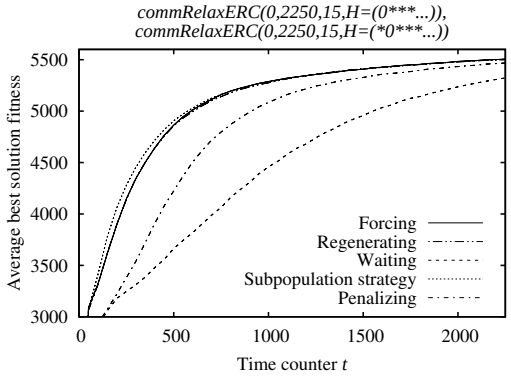

Figure 16: A plot showing the average best solution fitness obtained on a fitness landscape interpolated from real-world data as a function of the time counter t. The optimization was subject to the two commitment relaxation ERCs $commRelaxERC(0, 2250, 15, H = (0 * * * ...))$ and $commRelaxERC(0, 2250, 15, H = (*0 * * * ...))$.

mance of the repairing policies.

Based on these results, if a single policy is to be chosen, then we would select the policy, forcing, for the real-world problem as it performs best after $T = 2250$ 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).[15] In comparison to the $NK\alpha$ landscapes considered for offline testing, the interpolated landscape is smoother and contains significantly fewer local optima; both aspects are attributed to the low number of data points.

Nevertheless, as it is apparent from Figure 16, the performance of the policies on the interpolated landscapes is largely in alignment with the findings made on the $NK\alpha$ landscapes: The repairing policies tend to perform better than waiting and penalizing, and, while the subpopulation strategy performs best at the beginning of the optimization, all repairing policies tend to perform identically at the end of the optimization. Clearly, in reality one is usually able to perform a single optimization run only 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.

[15] 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.

9 Summary and Conclusion

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 this study, we have proposed and analyzed various policies for dealing with non-evaluable solutions, and assessed them for two types of ephemeral resource constraints (ERCs) — commitment relaxation ERCs and periodic ERCs — using four test problems: OneMax, TwoMax, several MAX-SAT problem instances, 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 that used landscape data from a real closed-loop optimization problem.

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 policy adopted. Which choice of policy 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 policy. 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 policy.

With respect to the impact of the individual ERC types, our analysis concluded that with commitment relaxation ERCs, we would tentatively say that repairing policies should not be used (i.e. the genotype of a solution should not be modified) for the majority of constraint settings, while they are appropriate policies with periodic ERCs. 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 converge to a suboptimal population state. In situations where it should be repaired, we can tentatively suggest that a policy that aims at creating fit repaired solutions should be preferred over a naïve forcing policy.

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. Our immediate attention is turning to the design and tuning of intelligent search policies. In (Allmendinger and Knowles, 2011) we have already shown that an EA that learns offline (using a reinforcement learning agent) and online (using a

multi-armed bandit algorithm) when to switch between the different static constraint-hanlding policies introduced here can yield better performance than the static strategies themselves. We are also looking at the treatment of problems where some solutions are more costly in time or resources to evaluate than others. The challenge there is that the optimizer has not only to account for the fitness of solutions, but also for their differential costs of evaluation.

Acknowledgment

The authors would like to thank Julia Handl for critical comments of a draft of this manuscript. We would also like to thank Hans-Paul Schwefel for answering many questions about his experience with resource constraints.

References

Allmendinger, R. and Knowles, J. (2010a). Ephemeral resource constraints in optimization and their effects on evolutionary search. Technical Report MLO-20042010, University of Manchester.

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

Allmendinger, R. and Knowles, J. (2011). Policy learning in resource-constrained optimization. In *GECCO '11 — Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pages 1971–1978.

Bäck, T., Knowles, J., and Shir, O. M. (2010). Experimental optimization by evolutionary algorithms. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (Companion)*, pages 2897–2916.

Bedau, M. A. (2010). Coping with complexity: Machine learning optimization of highly synergistic biological and biochemical systems. Keynote Talk at the 12th Annual Conference on Genetic and Evolutionary Computation.

Booker, A. J., Dennis, J. E., Frank, P. D., Serafini, D. B., Torczon, V., and Trosset, M. W. (1999). A rigorous framework for optimization of expensive functions by surrogates. *Structural and Multidisciplinary Optimization*, 17(1):1–13.

Borodin, A. and El-Yaniv, R. (1998). *Online Computation and Competitive Analysis*. Cambridge University Press.

Bosman, P. A. N. and Poutré, H. L. (2007). Learning and anticipation in online dynamic optimization with evolutionary algorithms: The stochastic case. In *GECCO '07 — Proceedings of 9th Annual Conference on Genetic and Evolutionary Computation*, pages 1165–1172.

Box, G. E. P. (1957). Evolutionary operation: A method for increasing industrial productivity. *Applied Statistics*, 6(2):81–101.

Box, G. E. P., Hunter, J. S., and Hunter, W. G. (2005). *Statistics for Experimenters: Design, Innovation, and Discovery*. Wiley, 2nd edition.

Branke, J. (2001). *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers.

Büche, D., Stoll, P., Dornberger, R., and Koumoutsakos, P. (2002). Multiobjective evolutionary algorithm for the optimization of noisy combustion processes. *IEEE Transactions on Systems, Man, and Cybernetics C*, 32(4):460–473.

Caschera, F., Gazzola, G., Bedau, M. A., Moreno, C. B., Buchanan, A., Cawse, J., Packard, N., and Hanczyc, M. M. (2010). Automated discovery of novel drug formulations using predictive iterated high throughput experimentation. *PloS ONE*, 5(1):e8546.

Coello, C. A. C. (2002). 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.

Cressie, N. (1993). *Statistics for Spatial Data*. Wiley.

Davis, L. (1989). Adapting operator probabilities in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 61–69.

De Jong, K. A. and Spears, W. M. (1989). Using genetic algorithms to solve NP-complete problems. In *Proceedings of the International Conference on Genetic Algorithms*, pages 124–132.

Dunn, W. B. (March 2011). Email discussion with W. B. Dunn.

Finkel, D. E. and Kelley, C. T. (2009). Convergence analysis of sampling methods for perturbed lipschitz functions. *Pacific Journal of Optimization*, 5:339–350.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701.

Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the 2nd International Conference on Genetic Algorithms*, pages 41–49.

Hansen, P. and Jaumard, B. (1990). Algorithms for the maximum satisfiability problem. *Computing*, 44(4):279–303.

Hebbron, T., Bullock, S., and Cliff, D. (2008). $NK\alpha$: Non-uniform epistatic interations in an extended NK model. In *Artificial Life XI*, pages 234–241.

Heckler, R. and Schwefel, H.-P. (1978). Superimposing direct search methods for parameter optimization onto dynamic simulation models. In *Proceedings of the Conference on Winter Simulation*, pages 173–181.

Herdy, M. (1997). Evolutionary optimization based on subjective selection — evolving blends of coffee. In *European Congress on Intelligent Techniques and Soft Computing*, pages 640–644.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. MIT Press.

Jarvis, R., Rowe, W., Yaffe, N., O'Connor, R., Knowles, J., Blanch, E., and Goodacre, R. (2010). Multiobjective evolutionary optimisation for surface-enhanced Raman scattering. *Analytical and Bioanalytical Chemistry*, 397(5):1893–1901.

Judson, R. S. and Rabitz, H. (1992). Teaching lasers to control molecules. *Physical Review Letters*, 68(10):1500–1503.

Kauffman, S. (1989). Adaptation on rugged fitness landscapes. In *Lecture Notes in the Sciences of Complexity*, pages 527–618.

Klockgether, J. and Schwefel, H.-P. (1970). Two-phase nozzle and hollow core jet experiments. In *Engineering Aspects of Magnetohydrodynamics*, pages 141–148.

Knowles, J. (2009). Closed-loop evolutionary multiobjective optimization. *IEEE Computational Intelligence Magazine*, 4(3):77–91.

Liepins, G. E. and Potter, W. D. (1991). A genetic algorithm approach to multiple-fault diagnosis. In *Handbook of Genetic Algorithms*, pages 237–250.

Mahfoud, S. W. (1995). *Niching methods for genetic algorithms*. PhD thesis, University of Illinois at Urbana-Champaign.

Michalewicz, Z. (2010). Some thoughts on wine production. Keynote Talk at the International Conference on Parallel Problem Solving from Nature — PPSN XI.

Michalewicz, Z. and Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32.

Montgomery, D. C. (1976). *Design and Analysis of Experiments*. Wiley.

Mühlenbein, H. and Schlierkamp-Voosen, D. (1993). Optimal interaction of mutation and crossover in the breeder genetic algorithm. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, page 648.

Nguyen, T. T. (2011). *Continuous dynamic optimisation using evolutionary algorithms*. PhD thesis, University of Birmingham.

Nocedal, J. and Wright, S. J. (1999). *Numerical Optimization*. Springer.

O'Hagan, S., Dunn, W. B., Brown, M., Knowles, J., and Kell, D. B. (2005). 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.

O'Hagan, S., Dunn, W. B., Knowles, J., Broadhurst, D., Williams, R., Ashworth, J. J., Cameron, M., and Kell, D. B. (2007). Closed-loop, multiobjective optimization of two-dimensional gas chromatography / mass spectrometry for serum metabolomics. *Analytical Chemistry*, 79(2):464–476.

Pelikan, M. and Goldberg, D. E. (2000). Genetic algorithms, clustering, and the breaking of symmetry. In *Proceedings of Parallel Problem Solving from Nature — PPSN VI*, pages 385–394.

Qasem, M. and Prügel-Bennett, A. (2010). Learning the large-scale structure of the MAX-SAT landscape using populations. *IEEE Transactions on Evolutionary Computation*, 14(4):518–529.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog.

Rechenberg, I. (2000). Case studies in evolutionary experimentation and computation. *Computer Methods in Applied Mechanics and Engineering*, 2-4(186):125–140.

Reeves, C. R. and Rowe, J. E. (2003). *Genetic algorithms — Principles and Perspectives: A guide to GA theory*. Kluwer Academic Publishers.

Schwefel, H.-P. (1968). Experimentelle optimierung einer Zweiphasendüse. Bericht 35 des AEG-Forschungsinstituts Berlin zum Projekt MHD-Staustrahlrohr.

Schwefel, H.-P. (1975). *Evolutionsstrategie und numerische Optimierung*. PhD thesis, Technical University of Berlin.

Shir, O. and Bäck, T. (2009). Experimental optimization by evolutionary algorithms. Tutorial at the 11th Annual Conference on Genetic and Evolutionary Computation.

Shir, O. M. (2008). *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.

Shir, O. M., Roslund, J., and Rabitz, H. (2009). Evolutionary multi-objective quantum control experiments with the covariance matrix adaptation. In *GECCO '09 — Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pages 659–666.

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

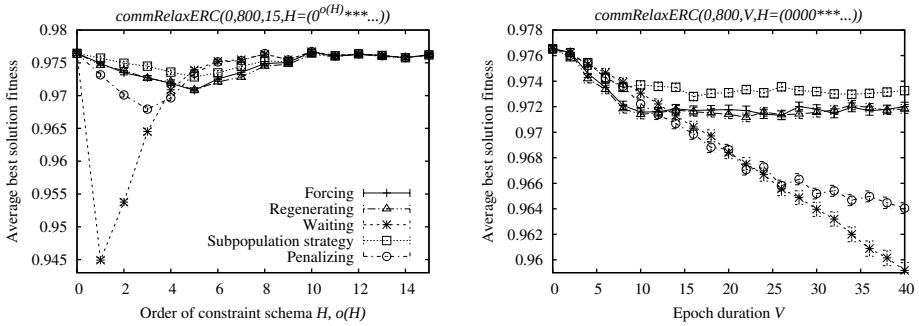

Figure 17: Plots showing the average best solution fitness found and its standard error on several MAX-SAT instances as a function of the order of the constraint schema $o(H)$ (left) and the epoch duration V (right). For each setting shown on the abscissa, a Friedman test (significance level of 5%) has been carried out. In the left and right plot, the subpopulation strategy performs best in the ranges $1 < o(H) < 5$ and $V > 10$, respectively; there is no clear winner for the other settings.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9.

Thompson, A. (1996). *Hardware Evolution: Automatic design of electronic circuits in reconfigurable hardware by artificial evolution*. PhD thesis, University of Sussex.

Vaidyanathan, S., Broadhurst, D. I., Kell, D. B., and Goodacre, R. (2003). Explanatory optimization of protein mass spectrometry via genetic search. *Analytical Chemistry*, 75(23):6679–6686.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1:67–82.

Appendix

A Commitment composite ERCs

Figure 17 and 18 show the results obtained with the different constraint-handling strategies on several MAX-SAT instances and the TwoMax function, respectively. In general, the subpopulation strategy tends to obtain a higher average best solution fitness than forcing and regenerating because it converges quicker to a suboptimal population state. Further evidence of this behaviour is given by the plots in the right column of Figure 18, which show the probability that the majority of a population climbs up the optimal slope.

B Periodic ERCs

Figure 19 and 20 show the results obtained with the different constraint-handling strategies on the MAX-SAT instances and the TwoMax function, respectively. Unlike on the MAX-SAT instances (and OneMax), waiting (and penalizing) tend to perform significantly better than the repairing policies for the majority of constraint parameter settings on the TwoMax problem. 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 $o(H)$. From Figure 21, which compares the performance of waiting against the one of the subpopulation strategy for different constraint schemata, we observe that waiting is able to maintain a significant performance advantage over a repairing policy (all three repairing policies performed similarly) for schemata below the straight line.

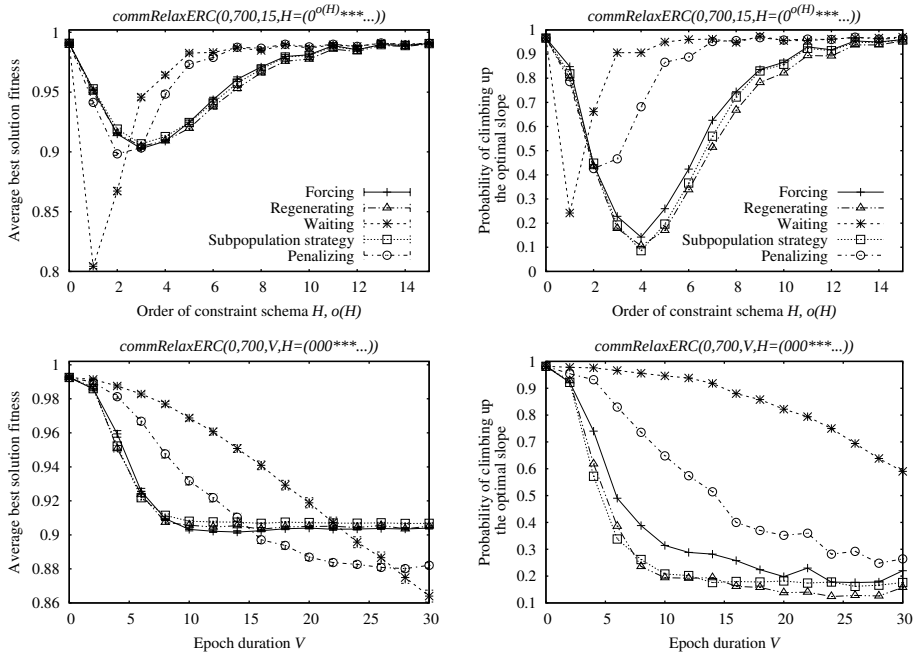

Figure 18: Plots showing the average best solution fitness found and its standard error (left column) and the probability (measured by the relative number of algorithmic runs) that the majority of the individuals in a population ends up on the optimal slope at the end of the optimization (right column) on TwoMax as a function of the order of the constraint schema $o(H)$ (top row), the epoch duration V (bottom row). For each setting shown on the abscissa (of the plots on the left-hand side), a Friedman test (significance level of 5%) has been carried out. In the top left plot, waiting performs best in the range $2 < o(H) < 6$, while, in the the bottom left plot, it performs best in the range $2 < V < 22$ with the subpopulation strategy being best in the range $V > 22$; there is no clear winner for the other settings.

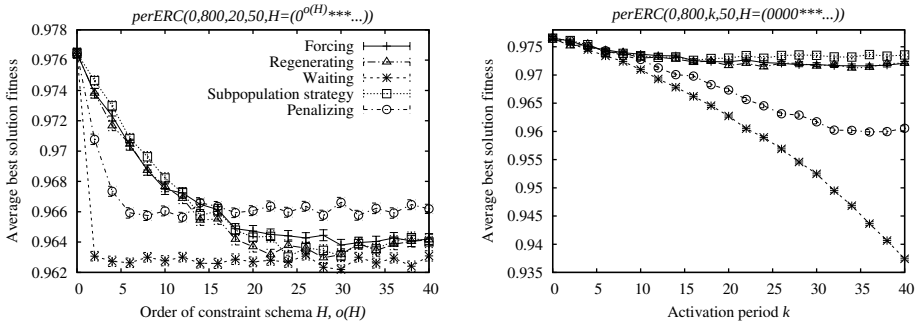

Figure 19: Plots showing the average best solution fitness found and its standard error on several MAX-SAT instances as a function of the order of the constraint schema $o(H)$ (left) and the activation period k (right). For each setting shown on the abscissa, a Friedman test (significance level of 5%) has been carried out. In the left plot, the subpopulation strategy performs best for $o(H) = \{2, 4, 8, 10\}$, while penalizing performs best in the range $o(H) > 16$. In the right plot, the subpopulation strategy performs best in the range $k > 24$; there is no clear winner for the other settings.

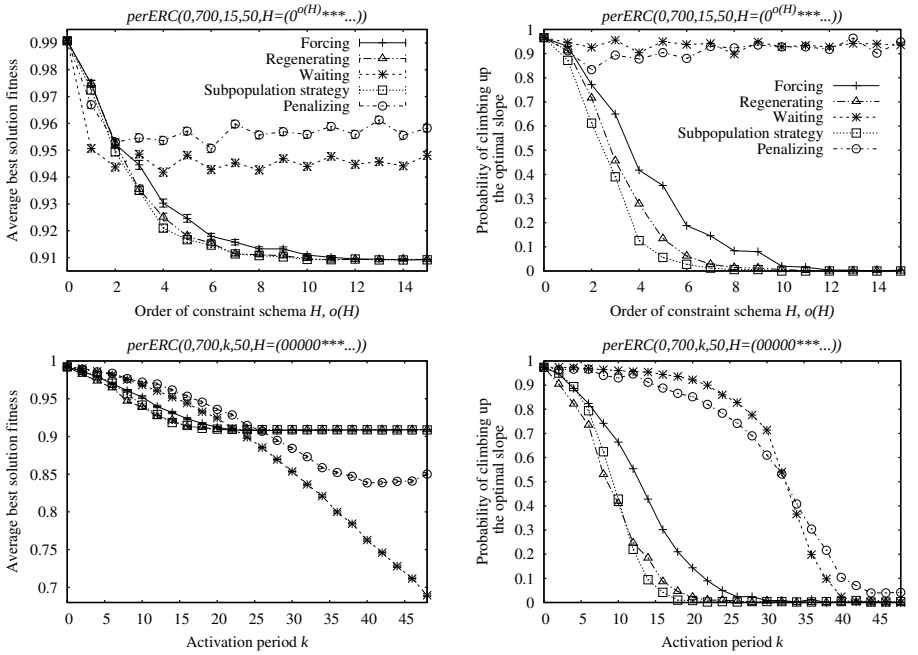

Figure 20: Plots showing the average best solution fitness found and its standard error (left column) and the probability (measured by the relative number of algorithmic runs) that the majority of the individuals in a population ends up on the optimal slope at the end of the optimization (right column) on TwoMax as a function of the order of the constraint schema $o(H)$ (top row) and the activation period k (bottom row). For each setting shown on the abscissa (of the plots on the left-hand side), a Friedman test (significance level of 5%) has been carried out. In the top and bottom left plot, penalizing performs best in the ranges $o(H) > 2$ and $16 < k < 24$, respectively; there is no clear winner for the other settings.

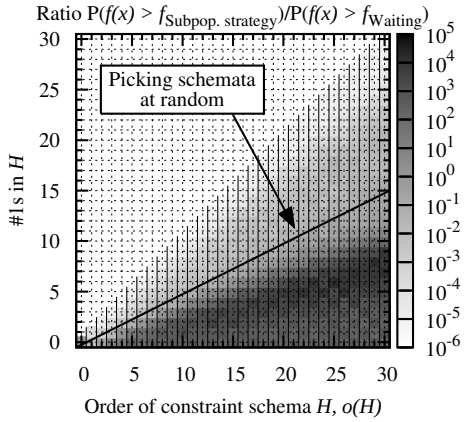

Figure 21: A plot showing the ratio $\mathrm{P}(f(x) > f_{\text{Subpop. strategy}})/\mathrm{P}(f(x) > f_{\text{Waiting}})$ on TwoMax as a function of the order of the constraint schema $o(H)$, and the number of order-defining bits in H with value 1 for the ERC $perERC(0, 700, 15, 50, H)$; here, x is 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 policy $*$. If $\mathrm{P}(f(x) > f_*)/\mathrm{P}(f(x) > f_{**}) > 1$, then policy $**$ is able to achieve a higher average best solution fitness than policy $*$ 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 $*$.