Ephemeral Resource Constraints in Optimization and Their Effects on Evolutionary Search

Richard Allmendinger and Joshua Knowles

Abstract

This work deals with a new class of optimization problems: problems in which solutions, despite being feasible, may be *unavailable* for evaluation from time to time (temporarily). Such problems may arise wherever resources are required in the evaluation of solutions, such as in experimental optimization and expensive computer simulations. Under certain circumstances, such as time and budget restrictions, machine breakdowns, absence of required skilled employees, and so on, the resources needed to conduct an experiment may not be available, so a solution can be temporarily non-evaluable. We model the temporary unavailabilities of resources by what we are calling *ephemeral resource constraints* (ERCs). In this paper, we define ERC problems and provide an initial set of ERC types which represent real-world limitations. To analyze the impact of the proposed ERC types on evolutionary search we engage in a theoretical and empirical investigation. The theoretical investigation uses Markov chains to analyze the impact of one of the proposed ERC types on common selection and reproduction models used within EAs. The experimental investigation analyzes the impact of the other ERC types on more complete EAs. Further research into the effects of ERCs is needed, and we facilitate this by providing an initial suite of test problems.

Index Terms

optimization; evolutionary experimentation; evolutionary computation; expensive objective function; constrained optimization;

I. Introduction

An optimization problem of generic form can be defined as

$$\begin{aligned} \text{Maximize} \quad & f(\vec{x}) \\ \text{subject to} \ \ & \vec{x} \in X, \end{aligned} \tag{1}$$

Richard Allmendinger and Joshua Knowles are with the University of Manchester, UK (email: allmendr@cs.man.ac.uk, j.knowles@manchester.ac.uk)

where $\vec{x} = (x_1, ..., x_l)$ is a *solution vector* for which values must be found and X a *feasible search space*. The *objective function* (also known as *fitness function*) $f : X \mapsto Y$ represents a mapping from X into the *objective space* $Y \in \mathbb{R}$ and is in general *black-box* [1], meaning we have no knowledge or access to it.

Black box optimization algorithms a can be represented as mappings from previously visited solutions to a single new solution in X [1]. Formally, $a : \phi(t) \mapsto \vec{x}_t$, where the *search history* $\phi(t) = \{(\vec{x}_1, y_1), ..., (\vec{x}_{t-1}, y_{t-1})\}$ denotes the time-ordered set of solutions visited until time step $t-1$, and $\vec{x}_i$ and y_i indicate the X value, respectively, the corresponding Y value of the ith successive element in $\phi(t)$. While in standard optimization problems, the objective value $f(\vec{x}_t)$ of a feasible solution $\vec{x}_t \in X$ is $y_t = f(\vec{x}_t)$ (sometimes even for infeasible solutions), in *ephemeral resource-constrained optimization problems* (ERCOPs), which are the focus of this study, we have

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

where $E_t(\sigma)$ represents the *set of evaluable solutions* (or *evaluable search region*) at time step t. In our case, $E_t(\sigma)$ is defined by a set of schemata into which solutions have to fall in order to be evaluable. The set $E_t(\sigma)$, or the *ephemeral resource constraints* (ERCs), may *change over time* depending on a set of problem-specific parameters σ. These parameters may relate to things like *random events*, *uncontrolled factors* and the *search history*.

Our motivation for ERCOPs comes from a number of optimization scenarios that recently arose in the experimental sciences, particularly in the field of combinatorial biochemistry applications and instrument set-up optimization in analytical chemistry [2]–[10], as well as quantum control in physics [11], [12]. These problems have been tackled using evolutionary algorithms (EAs) [13]–[15], which are population-based search techniques that employ a form of simulated evolution for finding good solutions for a wide range of problems including noisy [16], [17], constrained [18], [19], dynamic [20], and multiobjective problems [21]–[23]. In the motivating problems, one can obtain an objective value only by conducting a physical experiment and thus by using resources such as time, money, skilled engineers, people (e.g. in drug trials) and instruments. The set E_t then represents all resources available at a particular time step t and thus the set of all experiments that can be conducted at this time step. At a particular time, or under a particular set of circumstances, the resources required to conduct an experiment may not be available, so solutions can become temporarily non-evaluable. These temporary changes in resource availabilities, or, equivalently, in E_t, are modeled by the ERCs.

So far, ERCs have been mentioned only recently in published work by Knowles [2], suggesting that there is little or no research on how ERCs affect an optimization process and what are good strategies to overcome associated problems. From personal communication with one of the pioneers of experimental optimization, Hans-Paul Schwefel, however, we learnt that ERCs were already encountered during his experiments back in the 60s. These pioneering experimental optimization problems were mainly concerned with shape design problems for fluid dynamic models [24], [25].

In one of Schwefel's experimental optimization problems the aim was to find an optimal shape of a flashing nozzle that offers maximum energy efficiency. The nozzle was constructed from a series of conic brass rings, each having a ’left’ and ’right’ diameter such that the right diameter of one had to match the left of the subsequent one. Several copies of the set of rings would be required to allow unfettered evolution of the nozzle shape. Thus, the evaluation of some solutions required brass rings that were not available at the outset, and had first to be ordered, produced, and delivered, before the actual experiment could be conducted. Since the optimizers were not faced with (strict) budget nor with time constraints all solutions could be evaluated, albeit sometimes delayed. In modern experimental optimization problem, which are often of higher complexity and may involve severe time and/or budget restrictions, there are several factors that may further complicate the process of ensuring resources availability: (i) restricted storage space; (ii) time lags between ordering resources and receiving them; (iii) shelf-lives of resources; (iv) availability of resources in certain fixed batch sizes only.

In another problem encountered by Schwefel, the evaluation of a single solution was done by running a time-consuming simulation on a computer. During some simulations the simulation broke down (i.e. an execution error or exception occurred). Although the failure occurred for always the same solutions, thus making the resource constraint rather permanent or static, this problem shows that ERCs can be *unknown* and occur *unexpectedly* even during an evaluation. However, ERCs may not only occur suddenly but also on a *periodical* basis as is the case, for example, when machines or people are available only on particular weekdays.

ERCs may also occur due to *uncontrolled factors*, for example, with experiments that depend on the weather. Perhaps more interestingly ERCs may arise due to *previously made decisions*. For example, if instrument settings must (or can) be changed only in certain intervals, then choosing a setting activates a temporary *commitment constraint* allowing only solutions that make use of this particular setting to be evaluable.

A factor that was not encountered in Schwefel’s experiments but which is quite common in modern experimental problems is the presence of non-homogeneous *experimental costs* in the sense that some experiments might be more expensive to conduct than others. Thus, under a limited budget, this might cause an optimizer not only to follow fitness gradients but also to account for variable experimental costs. This very realistic problem formulation has been considered in the Robot Scientist study of King et al. [5], albeit for an inference problem rather than an optimization problem.

Another common factor in modern experimental problems is that experimental equipment is often capable of doing *experiments in batches* (truly in parallel) rather than on a sequential basis. Here, the property of EAs to employ simultaneously a population of solutions is particularly convenient. However note that the offspring population size might be dictated by the batch size [2]. Deciding which experiments to conduct in a batch was also an important aspect in the Robot Scientist study and has been investigated in detail by Byrne [26].

Optimizing ERC problems is similar to several research areas: traditionally, experimental problems are dealt with using statistical methods referred to as *experimental design* or *design of experiments* (DoE) [27]–[29]. 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 usually high-dimensional search spaces and one is ultimately interested in finding a single optimal solution. Despite sharing the common feature of time-linkage, the aim of finding a single optimal (and ultimate) solution makes ERCOPs different from *online (dynamic) optimization problems* [30], [31], where the aim is to optimize rather a cumulative score over some period of time. Unlike in traditional *dynamic optimization* [20], the objective space in ERCOPs does not change over time and thus the optimal solution does not need to be tracked. A further related area is *constrained optimization* [18], [19], [32]. However, the fact that ERCs prevent the evaluation of solutions that are otherwise feasible makes them materially different from standard constraints including dynamic constraints [33].

At this point, it should be apparent that the variety of ERC types is potentially large and that their impact on an optimization process is as yet unknown. Hence, before one can start designing efficient search policies, such as suitable scheduling or learning strategies, one needs to first define formally the problem and understand how ERCs might generally affect a search process. It is these issues that provide the focus for this study.

Organization

The rest of this paper is organized as follows. In Section II, a general problem definition of ERCOPs is given and the concept of a communication channel that mediates between an optimizer and the problem itself to model the ERC is introduced. An initial suite of different ERCs including constraints that we encountered in our own collaborative work and that seem to be common in real-world is defined in Section III. To analyze the impact of the ERCs on evolutionary search we engage in a theoretical and empirical investigation. In the theoretical investigation (Section IV) we use the concept of Markov chains to analyze the impact of one of the proposed ERCs on common models for performing selection and reproduction within EAs. Before we proceed with the experimental analysis in Section VI, which will be on the impact of the other ERCs on a set of selected EAs, we describe in Section V the choice of test functions, the specific ERCs, and give all parameter settings in Section V. In the concluding section we draw together the findings from the experimental and theoretical analyses and discuss directions for further research.

II. General problem formulation of ERC optimization problems

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

Consider Figure 1, which illustrates the basic setup of a real experimental problem. In general, this setup establishes a closed-loop interaction between an optimizer (e.g. an evolutionary algorithm (EA)) and the experimental platform including a (human) experimentalist [2], [5], [34]. The EA submits a population plan (i.e. a generation) to be evaluated by the experimental platform; note, the size of the generation might be dictated by the experimental platform. When the experiment is completed, a generation of fitness values is returned. But the process is complicated by the fact that resources are needed to conduct the experiments, and these might run out e.g. as a function of time, or previous actions taken (or both). If the EA requests the evaluation of a solution that cannot be evaluated

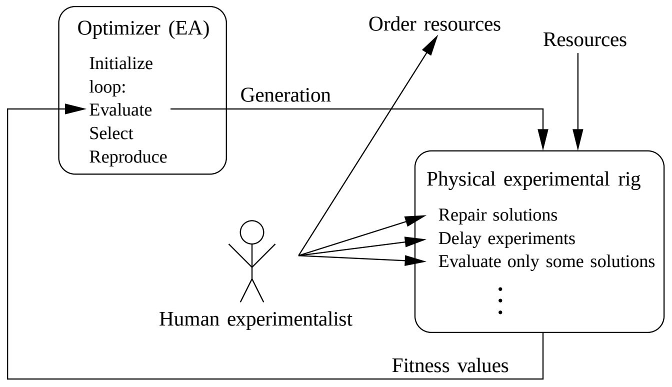


Fig. 1. A closed-loop experimental problem. When a resource needed to conduct an experiment is unavailable, the experimentalist may intervene: more resources may be ordered; only the evaluable solutions of a generation may be evaluated; the experiments may be delayed; or non-evaluable solutions may be *repaired* by hand.

then something must happen, and this something must be defined. In a real experiment, a human might intervene to order some new resources, or allow the EA to generate some alternative solution(s), or do something else.

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

A. General problem formulation of ERCOPs

An ERCOP is a standard optimization problem as given in (1), augmented by a set of dynamic constraints that restrict the solutions evaluable by a black box optimization algorithm as defined in (2). The *set of evaluable solutions* (or *evaluable search space*) at time step t, $E_t(\sigma)$, is a function of a *set of problem-specific parameters* σ. The set σ may include parameters such as various types of counters (e.g. cost counter, time counter and evaluation counter), the search history, random variables (which may encode random events like machine breakdowns), and so forth. To specify a particular ERCOP, E_t is defined in terms of one or more *ephemeral resource constraint* (ERC) functions, which map a particular set of circumstances (depending on σ) to schemata representing allowed or forbidden solutions. E_t is the union of schemata, when schemata positively define the feasible region; it is the complement of the union of schemata when the ERCs define the feasible region negatively, i.e. by defining disallowed regions.[1] Compared to standard constraints, the meaning of ERCs is different: assuming that the ERC functions can be computed, a solution $\vec{x}$ that violates an ERC at time t, or $\vec{x} \notin E_t$, is not *infeasible* but it is rather *non-evaluable*. That is, the experiment that is associated with $\vec{x}$ cannot be conducted and thus the objective function is undefined for solution $\vec{x}$ at time t. Figure 2 illustrates a common situation in ERCOPs with respect to the distribution of solutions across E_t and the feasible search space X.

[1]This becomes clearer in Section III where specific ERC functions are defined. In that section we will refer to specific ERC functions by h.

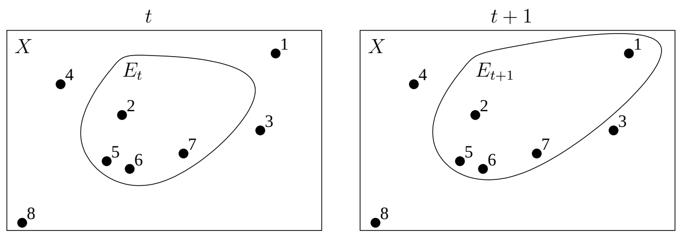


Fig. 2. A schematic diagram showing how a population consisting of the solutions 1,2,3,4,5,6,7,and 8 might be distributed across the feasible search space X and the evaluable search space E_t. At time step t, only the solutions 2,5,6, and 7 can be evaluated while the solutions 1,3,4, and 8 must be repaired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

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 not only refer to function evaluations of single solutions, as is the case in standard optimization problems, but also e.g. to evaluations of ERC functions, real time units (e.g. seconds), or cost units (e.g. pounds). This notion of time allows ERCs to be dependent amongst others on the number of evaluated solutions, expenses, or a certain date, such as days of the week (independent of the number of evaluated solutions).

The normal assumption is that all evaluations take equal time or resources, but this need not be the case. Generally, experiments may be of different durations and have non-homogeneous costs in terms of the financial or temporal resources they require. However, as the aim of this paper is to investigate solely the impact of ERCs on the optimization and not the effects of non-homogeneous costs or experimental durations (which form an important part of ongoing investigations), we conform here to the normal assumption.

One important aim in defining an ERCOP or the simulation environment is to achieve (as far as possible) a logical separation between (i) the objective function (e.g. a OneMax function or an NK landscape), which is preserved intact, (ii) the ERCs and resource handling, and (iii) the optimizer or EA. If we can achieve a separation then we can test different EAs – standard off-the-shelf ones that are blind to resource constraints, or more advanced ones that are augmented with the ability to order resources preemptively, repair solutions between generations, and so on – all on the same platform. This would allow us to make meaningful statements about whether it is worthwhile using advanced strategies for dealing with the ERCs or not. (The additional separation between objective function and ERCs allows us to interchange objective functions with little difficulty, enabling us to understand how different objective functions are affected by different ERCs).

A schematic of the simulation environment we use is shown in Figure 3. The EA is shown on the left and the objective function on the right. Between them sits what we are calling the *communication channel*, which has several roles that enable the complete ERCOP to be well-defined. It collects the EA's population plans, simulates time and other variables, and based on these it implements the ERCs, thus determining if solutions in the population plan are evaluable or not. Upon discovering a non-evaluable solution, a number of actions are defined. These include ordering new resources, communicating with the EA, submitting null solutions (these will be explained below), repairing solutions, and so on.

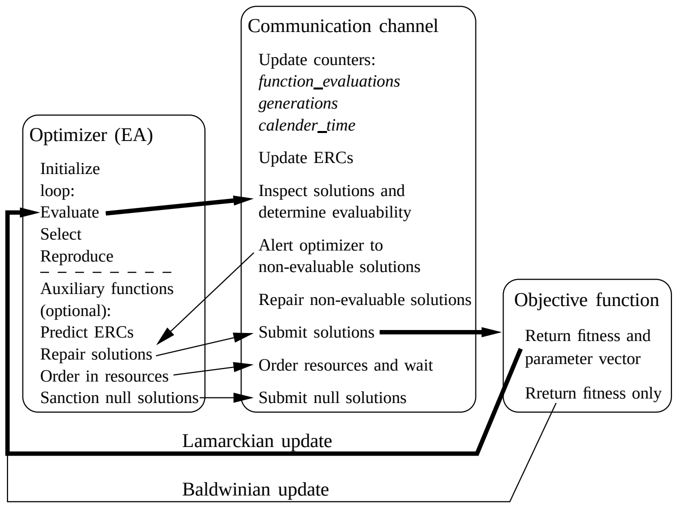


Fig. 3. Schematic of a simulation environment for a closed-loop problem with ERCs. The environment is divided logically into three units: the optimizer, the communication channel, and the objective function. The main role of the communication channel is to determine if ERCs are active, and to do this it maintains several (time) counters as well as the history of the search. When an ERC affects a solution to be evaluated, the communication channel may interact with the optimizer, to determine what must happen. But a simple optimizer (a standard EA) will not have any facility to do this. In this case, the communication channel uses some default actions such as performing a simple repair of non-evaluable solutions. The set of all arrows indicates how the three units of the simulator might interact. The dark arrows indicate how the simulators used in this paper interact, i.e., the EA is simple, repairs are done by default by the communication channel, and the genotype (parameter vector) of repaired solutions is returned to the optimizer, which thus uses Lamarckian population update.

In this paper, we wish to enable simple (off-the-shelf) EAs to run normally on our ERCOPs. To do this, we need to ensure that the EA may submit any population plan[2] at all, and it will receive in response a set of (corresponding) fitness values, none of which is null. There are two possible ways to achieve this. One is to have the communication channel order in resources (and, if necessary, *wait* in simulated time until they are available) so that all solutions do become evaluable. The other possibility is to have the communication channel repair non-evaluable solutions before submitting them to the objective function. The latter admits a further pair of choices: either the fitness values only of the repaired solutions are returned to the EA (Baldwinian population update) or the fitness and also the repaired solutions are returned to the EA (Lamarckian population update). These choices are illustrated in Figure 3.

When using the communication channel for interacting with more advanced EAs, we would allow the EA itself to make resource orders (perhaps preemptively), to poll the communication channel for information on current resource levels and the state of other variables like time, and for it to be responsible for repairing solutions. Provided the rules governing when the ERCs are active are correctly implemented by the communication channel, we will have the same basic ERCOP. It is just that we can extend the range of actions open to the optimizer during the optimization process.

We mentioned above the concept of null solutions, which we now briefly explain. The submission of a null

[2]Note that the EA may be steady state in which case the population may be of size 1

Algorithm 1 Communication channel used in this paper

```
Require: $ERCsSet$ (set of ERCs), $f$ (objective function), $T$ (time limit), $batchSize$ (number of experiments that can be done in a batch)
1: $t = 0$    // time counter
2: $EvalSol = \emptyset$    // set of currently evaluable solutions
3:
4: functionWrapper($P$){    // depending on $batchSize$, $P$ is either a single solution or a set of solutions that is submitted by an optimizer for evaluation
5: if $t < T$ then
6:   for $i$ = 1 to min$\{|P|, batchSize\}$ do
7:     if $\vec{x}_i$ satisfies all ERCs in $ERCsSet$ then
8:       $EvalSol = EvalSol \cup \vec{x}_i$
9:     else
10:      repair $\vec{x}_i$ and $EvalSol = EvalSol \cup \vec{x}_{i,\text{repaired}}$
11:    end if
12:  end for
13:  evaluate all solutions in $EvalSol$    // calls to $f$
14:  $t$++
15: end if
16: return $EvalSol$ (i.e. both fitness values and solutions)}
```

solution can be seen as the counterpart of repairing a non-evaluable solution. Null solutions increment only the time without making use of any other resources. An optimizer might submit them, for example, if it wishes to wait until a missing resource is again available.[3] Although null solutions (or equivalently incomplete batches) might play a useful role in handling ERCs, we do not consider it further in this paper, instead preferring to focus initially on a default repairing strategy for non-evaluable solutions.

The communication channel used in this paper is given in Algorithm 1 in form of pseudocode. Here, time steps refer to function evaluations of up to $batchSize$ solutions, where $batchSize$ is the maximal number of experiments that can be done in a batch. At each time step an optimizer is allowed to submit a population plan or generation P, which is iterated through by checking each solution for evaluability. While evaluable solutions are immediately remembered, non-evaluable ones are first repaired and then remembered. Once P has been iterated through, all remembered solutions are evaluated using some objective function. Finally, the time counter is incremented and the objective value as well as the repaired individuals are returned (i.e. Lamarckian population update is used). This

[3]Null solutions come into play particularly when experiments can be done in batches (truly in parallel). Assuming that $batchSize$ experiments can be done in parallel, the optimizer might choose to do only $m < batchSize$ due to ERCs.

process is repeated until a time limit T is reached.

III. Initial suite of ERCs

In this section we propose a set of ERC types including types designed to model constraints that we encountered in our own collaborative work and that would seem to be common in real-world applications. Before we define each of these ERC types, we want to introduce three elements that are common to all of them. These elements are the constraint time frame, the activation period and the constraint schema.

A. 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, of a batch of up to $batchSize$ solutions, real time units (e.g. seconds), a calendar period (e.g. Tuesday 2-4pm), or something else. Here, they refer to function evaluations of a batch of up to $batchSize$ solutions.

The *constraint time frame* (ctf) is $\{t | t_{\text{ctf}}^{\text{start}} \leq t < t_{\text{ctf}}^{\text{end}}\}$ where t represents some counter unit, as above. During the time frame, one or more ERCs may be active. Outside the time frame, no ERCs can be active. That is, $E_t(\sigma) \subseteq X, \forall t \in$ ctf, and $E_t(\sigma) = X, \forall t \notin$ ctf. The period of time $t < t_{\text{ctf}}^{\text{start}}$ and $t \geq t_{\text{ctf}}^{\text{end}}$ is the *preparation period*, respectively, the *recovery period*.

The restriction imposed by an ERC during the activation period can be of different forms. We consider the case where solutions have to fall into a particular *constraint schema* H, respectively, cannot be a member of H, in order to be evaluable. The schema H represents a particular subset of solutions that share some common properties. For instance, consider solution vectors to be binary strings of length $l = 5$. Then, the constraint schema $H = (*1**0)$ describes the set of all solutions with value 1 at bit position 2 and value 0 at bit position 5; 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). The two general properties of a schema are its *order* $o(H)$ and its *length* $l(H)$, representing the number of defined bit positions, respectively, and the distance between the first and last defined bit position [35]; for the above example we have $o(H) = 2$ and $l(H) = 3$. In this study we consider binary search spaces, but in non-binary ones, H might restrict solution parameters to lie within or out of certain parameter value ranges rather than to take specific parameter values.

B. Suite of ERCs

This section introduces four ERC types: (i) commitmnet ERCs, (ii) commitment composite ERCs, (iii) random ERCs, and (iv) periodic ERCs.

Commitment ERCs

A *commitment ERC* is an ERC that is activated for an activation period of always k time steps if: (i) a solution from a particular (always the same) constraint schema H is evaluated and (ii) there is no ongoing activation (i.e. a

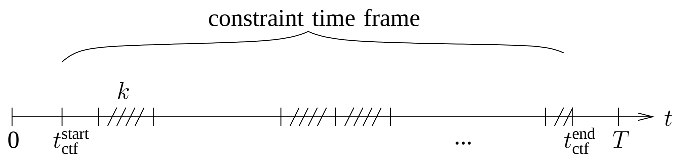

constraint time frame
k
0
$t_{\text{ctf}}^{\text{start}}$
...
$t_{\text{ctf}}^{\text{end}}$
T
t

Fig. 4. An illustration of how a commitment ERC may be activated during the constraint time frame. Activation periods are represented by the dashed parts.

new activation period can only be started if the previous one is over). A real-world example of a commitment ERC is:

"*In an optimization problem involving the selection of instrument settings, setting* xyz *once set cannot be changed over the next two hours*."

In this example, the activation period is $k = 2$ (assuming a time step represents one hour) and the constraint schema H represents the parameter combination that corresponds to instrument setting xyz. The restriction imposed during the activation period is that solutions have to fall into H (i.e. use instrument setting xyz) in order to be evaluable.

Figure 4 illustrates a possible distribution of activation periods during the constraint time frame. From this figure it is apparent that the total number of activation periods during the constraint time frame can vary between 0 and $\lfloor (t_{\text{ctf}}^{\text{start}} - t_{\text{ctf}}^{\text{end}})/k \rfloor$, where $\lfloor ... \rfloor$ is the floor function. 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 ERC function can be defined by

$$\begin{aligned} h : \text{if } & \{t - last\_activation \geq k\} \\ & \quad \text{if } \{t \in \text{ctf} \ \wedge \ \vec{x} \in H\} \Rightarrow \{last\_activation = t\} \\ & \text{else} \\ & \quad \text{if } \{t \in \text{ctf} \ \wedge \ \vec{x} \notin H\} \Rightarrow \{\vec{x} \notin E_t\}, \end{aligned} \tag{3}$$

where $last\_activation$ is initially set to $t_{\text{ctf}}^{\text{start}} - k$; the first two lines of Equation 3 are responsible for the activation of the ERC while the else clause ensures that solutions have to be in H during an activation. In future, we will denote a commitment ERC of this form by $commERC(t_{\text{ctf}}^{\text{start}}, t_{\text{ctf}}^{\text{end}}, k, H)$. In the communication channel of Algorithm 1, Equation 3 is checked at Line 7. A potential extension to this simple commitment ERC is to maintain not one but many different constraint schemata H_i so that multiple commitments may co-exist.

Commitment composite ERCs

A *commitment composite ERC* is an ERC that asks an optimizer at certain time steps to define several (always the

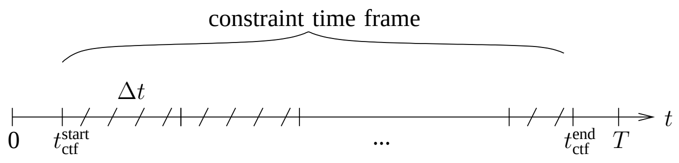

constraint time frame
Δt
t
0
$t_{\text{ctf}}^{\text{start}}$
...
$t_{\text{ctf}}^{\text{end}}$
T

Fig. 5. An illustration of a commitment composite ERC. The ERC is activated during the entire constraint time frame and an optimizer is given the opportunity to order new composites after every Δt time steps.

same) solution parameters to order (or fix) a composite; i.e. a composite can be considered as a partial solution (or a specific constraint schema H). There are no restrictions on the remaining solution parameters. Once a composite has been ordered, it has to be used until there is a new chance to order a composite (i.e. evaluable solutions need to be a member of H). In this paper, we consider constraints that ask an optimizer after each Δt time steps to order $\#SC$ composites.[4]

Put in other words, at each time step during the constraint time frame, an optimizer maintains $\#SC$ constraint schemata H_i and a solution needs to be a member of one of them. All H_i share the same order-defining bits, which we call in this context the *composite-defining bits*, and can be updated every Δt time steps. A graphical example of this ERC is given in Figure 5. To denote the composite-defining bits, we use again the notation of schemata. We represent a composite-defining bit position within a schema by $\#$ and denote the associated schema, which we call the *high level constraint schema*, by $H_\#$. For example, a high level constraint schema of order $o(H_\#) = 2$ with the composite-defining bit positions 2 and 5 would correspond to $H_\# = (*\#**\#)$ (assuming solution vectors being of length $l = 5$). Note, in a binary search space, a schema $H_\#$ is associated in total with $2^{o(H_\#)}$ different composites or constraint schemata.

A simplified real-world example of a commitment composite ERC is:

"*In an optimization problem charged with finding the best configuration of a mass spectrometer, some of the independent variables refer to the type of well plate wafers to use in the instrument. A wafer is characterized by two variables, which are related to its preparation: etch time (3 different levels) and etch concentration (2 different levels). Each Monday we are given the opportunity to order 2 types of well plates, and the subsequent experiments must use those (until the following Monday); previously ordered wafers must not be used any more as they are passed their recommended lifetime.*"

In this example, the two composite-defining bits are the solution bits that are associated with the etch time and the etch concentration, the number of time steps between two orders is $\Delta t = 7$ (assuming a day is one time step), and the number of storage cells is $\#SC = 2$; here, the overall number of different composites is $3 \times 2 = 6$.

[4]The initials SC refer to storage cells. In the real-world problem where we encountered a more complex commitment composite ERC, composites had a 'reuse number' and needed to be stored in separate storage cells.

This example represents a simplified version of a commitment composite ERC that we encountered in a real-world problem. However, although we faced storage limitations too, we were able to order wafers whenever we wanted but orders were associated with time-lags. Also, wafers had to be ordered in batches, were associated with shelf lives, and could be used only a certain number of times.

This ERC and a commitment ERC have a different effect on the optimization. Although both ERCs depend on decisions made in the past, this ERC is activated all the time while a commitment ERC may not be activated at all. On the other hand, a commitment ERC can cause an optimizer to get stuck in the sense that the ERC is activated over and over, causing all solutions to be from the same schema. By contrast, if the setting of Δt and $\#SC$ allows it, this ERC provides an optimizer at least the chance to order new composites and thus to evaluate solutions from different schemata.

The corresponding ERC function can be defined by

$$h : \text{if } \{t \in \text{ctf} \;\wedge\; (\bigwedge_{i=1}^{\#SC} \vec{x} \notin H_i(t))\} \Rightarrow \{\vec{x} \notin E_t\}, \tag{4}$$

where $H_i(t), i$=1, ..., $\#SC$, is the constraint schema that corresponds at time step t to the composite that is stored in 'storage cell' i. In future, we will denote a commitment composite ERC of this form by $commCompERC(t_{\text{ctf}}^{\text{start}}, t_{\text{ctf}}^{\text{end}}, \Delta t, H_{\#}, \#SC)$. In the communication channel of Algorithm 1, Equation 4 is checked at Line 7. The opportunity for an optimizer to order new composites, given the conditions are met, is given between Line 5 and 6. Note, for repairing one needs to provide an optimizer not only with a single constraint schema but with $\#SC$ constraint schemata, each corresponding to one of the available composites.

Random ERCs

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

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

We want to avoid situations in which there is no evaluable solution. This is simply achieved by removing any ERCs from the set $ActERCs_t$ that have a constraint schema that is contradictory to the constraint schema of a newly activated ERC. Clearly, this can be realized differently, for example, by not allowing a new ERC to enter $ActERCs_t$ rather than removing contradictory ERCs from $ActERCs_t$. Allowing contradictions and thus being perhaps unable

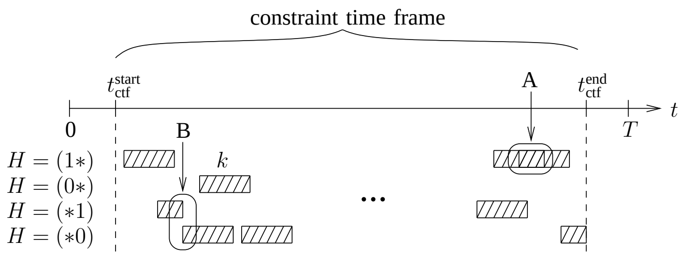


Fig. 6. A random ERC with an order of $o(H) = 1$, an activation period of k time steps and an activation probability of $p \approx 2/k$ ($k \geq 2$) is applied to binary strings of length $l = 2$. The 4(=$2^{o(H)}\binom{l}{o(H)}$) possible constraint schemata are represented on the left side. Each dashed block represents the activation period of an ERC in the set $ActERCs_t$. One can see that multiple ERCs can coexist, including ones with identical constraint schemata (A), and that an ERC is removed from $ActERCs_t$ if its activation period expires or if its constraint schema contradicts the one of a newly activated ERC (B).

to evaluate any solution from time to time is of course also an option. Figure 6 shows how the set $ActERCs_t$ may vary over time and the effects of enforced ERC removals using our approach.

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

"*In a multistage production process, a manufactured product runs through a series of stages, where, at each stage, the product is processed by a single machine which progresses it towards its final manufactured state. We wish to optimize the production facility/process in the following sense. For each stage there is a choice from among several machines (or machine types) capable of advancing the product to the next stage; we wish to select the best machine to use at each stage. Each machine may breakdown at any time step with a probability of 2% in which case it needs to be repaired; standard repairing takes 5 hours including test runs. Repairing time may increase if machine failures are more severe than expected (*A in Figure 6) *but, if necessary, a machine can be used before the test runs are finished (*B in Figure 6)."

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

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

The corresponding ERC function can be defined by

$$h : \text{if}\{t \in \text{ctf} \ \wedge \ (\bigvee_{i=1}^{|ActERCs_t|} \vec{x} \in H_i(t))\} \Rightarrow \{\vec{x} \notin E_t\}, \tag{5}$$

where $H_i(t), i$=1, ..., $|ActERCs_t|$ is the constraint schema of the ith ERC in the set $ActERCs_t$ at time step t. In future, we will denote a random ERC of this type by $randERC(t_{\text{ctf}}^{\text{start}}, t_{\text{ctf}}^{\text{end}}, k, o(H), p)$. In the communication

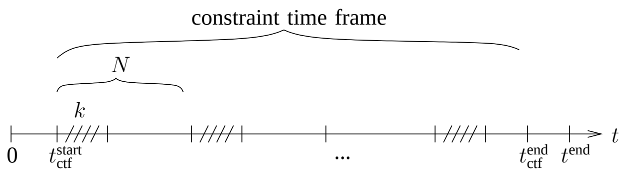

constraint time frame
N
k
0
$t_{\text{ctf}}^{\text{start}}$
...
$t_{\text{ctf}}^{\text{end}}$
t^{end}
t

Fig. 7. An illustration of a periodic ERC $perERC(t_{\text{ctf}}^{\text{start}}, t_{\text{ctf}}^{\text{start}}, k, N, H)$. The ERC is activated every N time steps for an activation period of always k time steps.

channel of Algorithm 1, Equation 5 is checked at Line 7. The drawing of random numbers for potential activations of new ERCs, and the removal of expired or contradictory ERCs from $ActERCs_t$ is performed between Line 5 and 6.

Periodic ERCs

A *periodic ERC* models the unavailability of a specific resource, represented by a constraint schema H, at regular time intervals. That is, the ERC is activated every N time steps (*period length*) for an activation period of always k time steps (see Figure 7). As the ERC models unavailabilities of resources, an individual is not allowed to be a member of the constraint schema. An example of a periodic ERC is:

"*In an optimization problem requiring the knowledge of skilled engineers to operate instruments, engineer* xyz *is not available on Mondays.*"

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

The corresponding ERC function can be defined by

$$\begin{aligned} h : \text{if } \{t \in \text{ctf} \ \wedge \ t_{\text{ctf}}^{\text{start}} + Ni \le t < \{t_{\text{ctf}}^{\text{start}} + Ni + k\}, \\ i = 0, 1, 2, \ldots \wedge \ \vec{x} \in H\} \Rightarrow \{\vec{x} \notin E_t\}. \end{aligned} \tag{6}$$

In future, we will denote periodic ERCs by $per(t_{\text{ctf}}^{\text{start}}, t_{\text{ctf}}^{\text{end}}, k, N, H)$. In the communication channel of Algorithm 1, Equation 6 is checked at Line 7. Potentially, the period length and the activation period may refer to different counter units. For example, consider the maintenance of machines. Maintenance might take hours (k is measured in real time units) and machines might need to be maintained after they have been used a certain number of time (N is measured in function evaluations).

IV. Theoretical study

This section conducts a theoretical investigation on the impact of periodic ERCs on two selection and reproduction schemes commonly-used within EAs. For this, we use the concept of Markov chains, which is introduced in the next section. We then derive the Markov model (transition probabilities) and analyze the simulation results. Finally, a summary is provided.

A. Markov chains

A *Markov process* is a random process that has *no memory* of where it has been in the past such that only the current *state* of the process can influence the next state. If the process can assume only a finite or countable set of states, then it is usual to refer to it as a *Markov chain* [36].

One can think of a Markov chain as a sequence $X_0, X_1, X_2, \ldots$ of random events occurring in time [35]. Suppose $S_0, ..., S_n$ are the $n+1$ possible values that each of the random variables X_t can take. Then, a chain moves from a state S_m at time t, to a state S_r at time $t+1$ with a probability of $p_{mr} = P(X_{t+1} \in S_r | X_t \in S_m)$. The probabilities p_{mr} $(m, r = 0, ..., n)$ are called *transition probabilities* and form the $n+1 \times n+1$ matrix **P**, the *transition matrix*. Thus, the probability that the chain is in state S_r at time t is the rth entry in the vector

$$\vec{u}_t = \vec{u}_0 \mathbf{P}^t, \tag{7}$$

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

When an evolutionary algorithm (EA) is modeled by a Markov chain it is easy to see that the population is the natural choice for describing a state. The transition probabilities then express the likelihoods that an EA changes from a current population to any other possible population after applying the stochastic effects of selection, crossover and/or mutation. It is also possible to consider other effects such as noisy fitness functions [37], niching [38] and *élitism* [39]. Once the transition matrix is calculated it can be used to calculate a variety of measurements, such as the first hitting time of a particular state or the probability of hitting a state at all. An overview of tools of Markov chain analysis can be found in any general textbook on stochastic processes, see e.g. [36], [40].

The drawback of modeling EAs with Markov chains is that the size of the required transition matrix grows exponentially in both the population size and string length. To keep Markov chain models manageable it is therefore common to use small population sizes and string lengths, see e.g. Goldberg and Segrest [41] and Horn [38]. Other options, which allow the modeling of more realistic EAs, are to make simplifying assumptions about the state space [42] or to use matrix notation only, see e.g. Vose and co-authors [43], [44] and Davis and Principe [45].

B. Modeling ERCs with Markov models

In this section we derive the transition probabilities for EAs optimizing in the presence of periodic ERCs. Our Markov chain model is based on the model of Goldberg and Segrest [41], which considers a simple environment that is composed of two individual types: Type A has always a fixed objective value (or fitness) of $f(A)$, while type B has a fitness of $f(B)$. This limitation allows for an intuitive definition of states. For a fixed population size of n, there are n+1 possible states, where state S_m represents a population with m type A individuals and $n-m$ type B individuals. Furthermore, in this simple EA model we do not apply mutation and crossover such that an offspring shall be simply a copy of the selected parent.

Goldberg and Segrest [41] used this model to investigate the effect of drift for a simple EA that used a generational reproduction scheme combined with fitness proportionate selection. They also extended the model to include mutation. Horn [38] extended it further to include niching. We extend it to include periodic ERCs and use the resulting model to analyze the impact of the ERC on two selection strategies, fitness proportionate and binary tournament selection, and two reproduction schemes, generational and steady state reproduction without *élitism*.

Selection probabilities

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

$$P_m(A) \;=\; \frac{mf(A)}{mf(A)+(n-m)f(B)}\;. \tag{8}$$

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

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

$$\begin{aligned} f(A)=f(B):\qquad & P_m(A) = \left(\frac{m}{n}\right)^2 + \frac{m(n-m)}{n^2} \\ f(A)>f(B):\qquad & P_m(A) = \left(\frac{m}{n}\right)^2 + 2\frac{m(n-m)}{n^2}. \end{aligned} \tag{9}$$

Transition probabilities

In our environment, the transition probabilities depend on the selected reproduction scheme, which in turn depends on the selected selection strategy. We first consider a *generational reproduction scheme* as already used in Holland's original genetic algorithm [13]; with regard to the experimental analysis where this genetic algorithm itself is considered, we denote this scheme by GGA. With GGA, the entire current population is replaced by the offspring population. That is, n selection steps are carried out per time step (with replacement). Using the selection probability $P_m(A)$ either for FPS or BTS, the transition probabilities $p_{mr} = P(X_{t+1} \in S_r | X_t \in S_m)$ for GGA of moving at time t from a state S_m with m type A individuals and $n-m$ type B individuals, to a state S_r with r type A individuals at time $t+1$, are defined as follows:

For $m = 0$

$$\begin{aligned} p_{mm} &= 1 \\ p_{mr} &= 0, \quad r = 1, ..., n. \end{aligned} \tag{10}$$

For $0 < m < n$ and $0 \leq r \leq n$

$$p_{mr} = \binom{n}{r} P_m(A)^r (1 - P_m(A))^{n-r}.$$

For $m = n$

$$\begin{aligned} p_{mr} &= 0, \quad r = 0, ..., n-1 \\ p_{mm} &= 1. \end{aligned}$$

With *steady state reproduction*, the population is updated after each selection step. Usually, an offspring individual replaces the worst individual in the population. This replacement strategy, however, ensures that the number of the less fit individual type in the population does not increase. Thus, to allow for a fair comparison with GGA, an offspring does not replace the worst individual in the population but a randomly chosen one regardless of its fitness. With regard to the experimental analysis, we denote this reproduction scheme by SSGA (rri). Note, replacing a random individual is usually not the method of choice but it has been shown elsewhere [46] that GGA and SSGA (rri) yield similar performance. Bearing in mind that one time step corresponds to one selection step, we obtain following transition probabilities for SSGA (rri):

For $m = 0$

$$\begin{aligned} p_{mm} &= 1 \\ p_{mr} &= 0, \quad r = 1, ..., n. \end{aligned} \tag{11}$$

For $0 < m < n$

$$\begin{aligned} p_{mr} &= 0, \quad r = 0, ..., m-2 \\ p_{mm-1} &= (1 - P_m(A))\frac{m}{n} \\ p_{mm} &= P_m(A)\frac{m}{n} + (1 - P_k(A))\frac{n-m}{n} \\ p_{mm+1} &= P_m(A)\frac{(n-m)}{n} \\ p_{mr} &= 0, \quad r = m+2, ..., n. \end{aligned}$$

For $m = n$

$$\begin{aligned} p_{mr} &= 0, \quad r = 0, ..., n-1 \\ p_{mm} &= 1. \end{aligned}$$

These transition probabilities will be the entries of the transition matrix **P**. Notice that the effect on the optimization imposed by the generational reproduction schema is identical to optimizing an ERCOP with $batchSize = n$. By contrast, steady state reproduction can be seen as optimizing the same ERCOP but using $batchSize = 1$.

Constrained transition probabilities for a periodic ERC

Note that for the same number of reproductive events to occur, SSGA (rri) needs to perform n time steps for each time step of GGA. Hence, in order for an ERC to impose the same effect on the optimization, we express ERCs in this section in terms of selection steps rather than time steps.

Consider a periodic ERC $perERC(in, (i+1)n, k, n, H = (B))$ ($i \in \mathbb{N}, k \leq n$), which is activated at selection step in for a period of n selection steps, i.e. one time step for GGA and n time steps for SSGA (rri). During the activation period of $k \leq n$ selection steps, we are not allowed to select type B individuals, or, equivalently, we have to select type A individuals. In the case where a type B individual is selected during this period, we repair it by simply forcing it into the right schema; i.e. it is converted into a type A individual. Before we derive the constrained transition probabilities for GGA we want to point out a few aspects:

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

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

For $m = 0$

$$\begin{aligned} p_{mr} &= 0, \quad r = 0, ..., k-1, k+1, ..., n \\ p_{mk} &= 1. \end{aligned} \tag{12}$$

For $0 < m < n$ and $0 \leq r < k$

$$p_{mr} = 0$$

For $0 < m < n$ and $k \leq r \leq n$

$$p_{mr} = \binom{n^{\text{new}}}{r^{\text{new}}} P_m(A)^{r^{\text{new}}} (1 - P_m(A))^{n^{\text{new}} - r^{\text{new}}}.$$

For $m = n$

$$\begin{aligned} p_{mr} &= 0, \quad r = 0, ..., n-1 \\ p_{mm} &= 1. \end{aligned}$$

Note, in the case where one and the same activation period constrains selection steps within two successive time steps, one needs to constrain both time steps and update the number of the constrained selection steps for each of the two time steps accordingly. A similar update of the number of constrained time steps needs to be made if the period length and activation period is greater than the population size, i.e. $N \geq k > n$; in this case, all n selection steps within a time step may be constrained.

With SSGA (rri), the ERC prevents us, during the activation period, from moving from a current state S_m to a state S_{m-1}, which can only be reached if a type B individual replaces a type A individual. As above, during the constraint time frame, the state S_0 is not an absorbing state anymore, so we move directly to state S_1. We obtain the following new transition probabilities for each of the k constrained time steps:

For any $m = 0$

$$\begin{aligned} p_{mr} &= 0, \quad r = 0, 2, 3, ..., n \\ p_{m0} &= 1. \end{aligned} \tag{13}$$

For any $0 < m < n$

$$\begin{aligned} p_{mr} \quad &= 0, \quad r = 0, ..., m-1 \\ p_{mm} \quad &= \frac{m}{n} \\ p_{mm+1} &= \frac{n-m}{n} \\ p_{mr} \quad &= 0, \quad r = m+2, ..., n. \end{aligned}$$

For any $m = n$

$$\begin{aligned} p_{mr} &= 0, \quad r = 0, ..., n-1 \\ p_{mm} &= 1. \end{aligned}$$

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

Calculating proportions of individual types in a population

One way to analyze the impact of an ERC on different selection and reproduction schemes is to monitor the proportion of the two individual types in a population. For this, one needs to first calculate the probability of ending up in any of the possible states $S_i, i = 0, ..., n$ after t time steps. In an unconstrained environment, this is done according to Equation 7 using the transition matrix $\mathbf{P}$. In a constrained environment we cannot use the transition matrix $\mathbf{P}$ across all t time steps but have to swap it with the constrained transition matrix $\mathbf{P}_c$ for the time steps that consist of constrained selection steps. For example, for a periodic ERC $perERC(in, (i+d)n, k, n, H = (B))$, where $d \in \mathbb{N}$ is the number of periods for which the ERC is activated, we calculate the probability vector at time step t ($t \geq d + i$) for GGA as follows:

$$\vec{u}_t = \vec{u}_0 \mathbf{P}^i \mathbf{P}_c^d \mathbf{P}^{t-d-i},$$

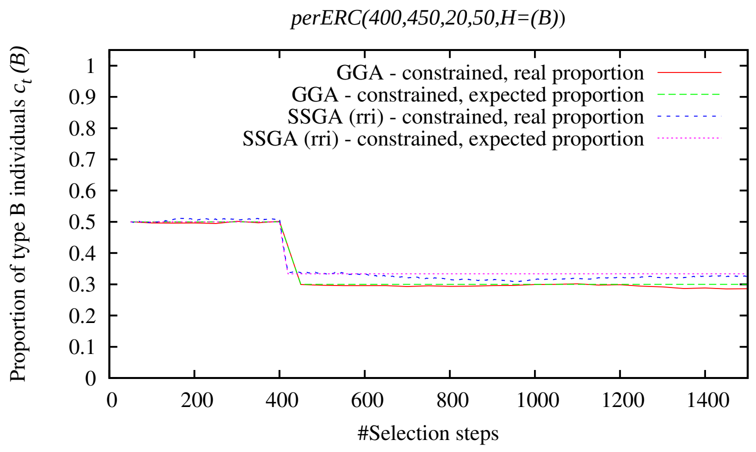


Fig. 8. Shown is the proportion of type B individuals $c_t(B)$ for GGA and SSGA (rri) as a function of the number of selection steps. Both individual types have equal fitness and the used constraint settings are given above the plot. The terms *real* and *expected* refer to proportions obtained by actually running the EA, respectively, by running the Markov chain. The EA results are averaged across 500 independent runs.

where the transition matrices **P** and $\mathbf{P}_c$ are calculated using Equations 10 and 12, respectively. The initial distribution vector $\vec{u}_0$ has a 1 at the ith entry and a 0 in the others, if we want to start with a population of exactly i type A individuals.

By contrast, for SSGA (rri), the probability vector after the same number of selection steps or at time step nt for the same ERC is calculated by

$$\vec{u}_{nt} = \vec{u}_0 \mathbf{P}^{in} (\mathbf{P}_c^k \mathbf{P}^{n-k})^d \mathbf{P}^{(t-d-i)n},$$

where the transition matrices **P** and $\mathbf{P}_c$ are calculated according to the Equations 11 and 13.

Remember, for periodic ERCs where the number of constrained selection steps within a time step is different from the length of activation period, one needs to update the constrained transition matrix of GGA accordingly.

Having obtained the probabilities of ending up in all the different states, we can calculate the expected proportions $c_t(A)$ and $c_t(B)$ of type A and B individuals in a population at time step t as follows:

$$c_t(A) = \frac{1}{n} \sum_{i=0}^{n} i u_t^i \ , \ c_t(B) = 1 - c_t(A),$$

where u_t^i is the ith entry of $\vec{u}_t$.

C. Simulation results

This section uses the measure of the expected individual type proportion to analyze the impact of period ERCs on two selection strategies, FPS and BTS, and two reproduction schemes, GGA and SSGA (rri). We consider first the case where both individual types have equal fitness values and then the case where they are different. For the analysis, we keep the population size fixed at $n = 50$.

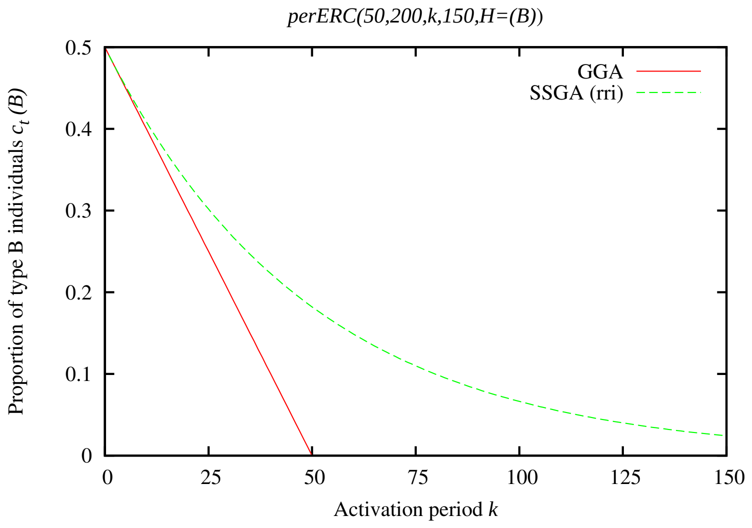


Fig. 9. Shown is the proportion of type B individuals $c_t(B)$ for GGA and SSGA (rri) at selection step 1000 as a function of the activation period k. Both individual types have equal fitness.

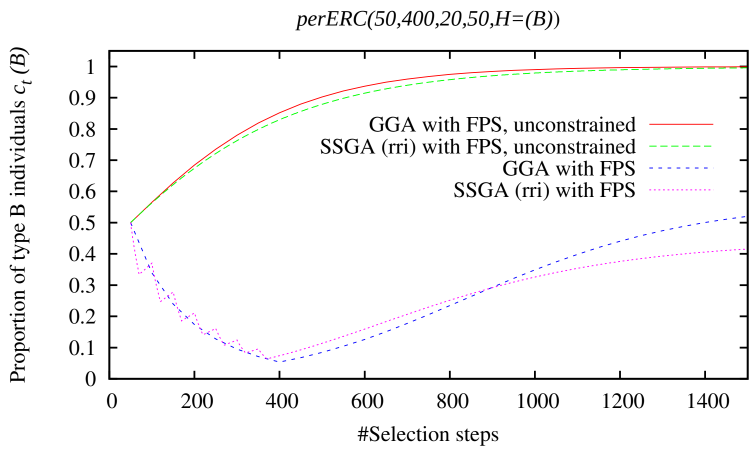
Identical fitness values: $f(A) = f(B)$

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

From Figure 8 we can see that an expected proportion of 0.5 is achieved until selection step 400 at which we activate the periodic ERC $perERC(400, 450, 20, 50, H = (B))$, which has a unique activation period of $k = 20$ selection steps.[5] Clearly we see here that after the constraint is lifted (selection step 420) there is no recovery of either EA.Although this effect can be put down to the specifics of the model (no selection pressure towards either individual type), we will see in the following theoretical and experimental studies, several results which display a similar pattern. That is, a constraint can have a permanent or long-lived effect on search performance even if it was active for a short time.

From the figure it is also clear that after the ERC is lifted, SSGA (rri) maintains a higher proportion of type B individuals than GGA. SSGA (rri) outperforms GGA because with this reproduction scheme there is a chance that an offspring of type A replaces another type A individual that is currently in the population. Of course, if such a situation occurs, then the proportion does not change by performing a selection step. By contrast, with GGA, all offspring are carried over to the next population meaning that the proportion of type B individuals in the population is a linear function of the activation period. This difference between the two reproduction schemes is also shown

[5]Note, in an EA performing optimization of a function, the number of performed selection steps displayed on the x-axes of Figure 8 would be equivalent to the number of performed function evaluations.



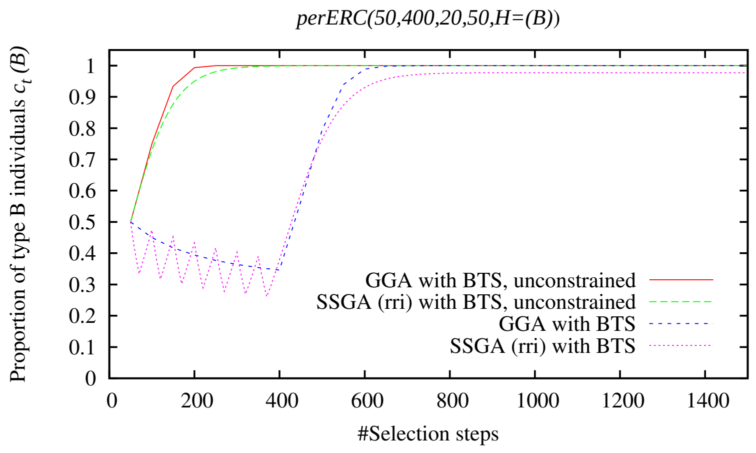


Fig. 10. Shown is the proportion of type B individuals $c_t(B)$ for FPS (top) and BTS (bottom) as a function of the number of selection steps. The term *unconstrained* refers to the proportions obtained in an ERC-free environment.

in Figure 9. From the figure one can see that SSGA (rri) is able to maintain a proportion of 0.2 after an activation period of $k = 50$, which is equal to the population size, while GGA cannot maintain a single type B individual in the population. Note, in the case where $k > 50$, the constraint is activated for more than one time step when using GGA. For example, for $k = 70$ the constraint restricts all 50 selection steps within one time step and 20 selection steps within the subsequent one.

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

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

When both individual types have different fitness values, the aim of an EA is to converge as quickly as possible to an optimal population consisting only of the fitter individual type. We focus our investigations mainly on the more interesting case where an ERC has a negative effect on the convergence behavior. Thus, the fitness of the

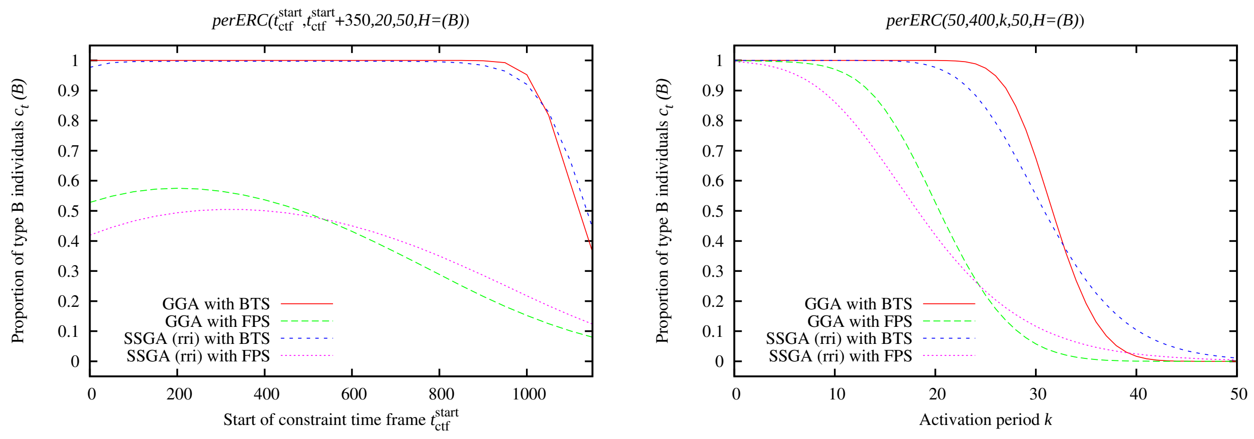


Fig. 11. Shown is the proportion of type B individuals $c_t(B)$ at selection step 1500 as a function of the start of the constraint time frame $t_{\text{ctf}}^{\text{start}}$ (left) and the activation period k (right).

individual type that cannot be selected, in our case type B, needs to be higher. If not otherwise stated, the fitness values are $f(A) = 1.0$ and $f(B) = 1.3$.

As the basis for our investigations we use the periodic ERC $perERC(50, 400, 20, 50, H = (B))$. This ERC is activated after the initialization for seven periods, each consisting of 50 selection steps whereby 20 of them are constrained. Figure 10 presents two plots which give an insight into the effect of this ERC on the convergence behavior of the four possible EA variants – GGA with BTS, GGA with FPS, SSGA (rri) with BTS, SSGA (rri) with FPS.[6]

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

Figures 11, 12 and 13 indicate how the proportion of type B individuals is affected when altering the constraint parameters. We can observe that:

- Longer activation periods degrade the performance of all EAs (Figure 11, right).
- Fixing the constraint time frame duration, but translating it (Figure 11, left), yields a non-monotonic effect on performance (all EAs, but most apparently with FPS): more preparation time gives more time to fill the population with fit individuals, whereas little recovery time detriments final fitness. These two effects trade off against each other.
- Increasing the duration of the constraint time frame (Figure 12) degrades performance.

[6]Note, we get the zigzag-shaped line for SSGA (rri) during the constraint time frame because $c_t(B)$ is plotted after each population update. That is, while $c_t(B)$ is plotted for SSGA (rri) after each selection step, for GGA it is plotted every n selection steps.

- Changing the fitness ratio (Figure 13) has only a switching effect on BTS (when the fitter individual changes), but for FPS the ratio smoothly affects final proportion up to a saturation point.
- Overall, comparing GGA with SSGA we see that SSGA achieves the higher proportion of fit individuals during the constraint time frame, and it recovers more rapidly after the constraint is lifted, but its rate of recovery does not reach the rate achieved by GGA, and ultimately GGA reaches a higher proportion (see Figs. 9 and 10). This can be explained by the replacement strategy of SSGA (rri): offspring may replace individuals in the population that are from the same type. During the activation period, this is beneficial as the number of poor type A individuals in the population does not increase linearly with the activation period. However, during the unconstrained selection steps, this may be disruptive in the sense that fit type B offspring may replace other

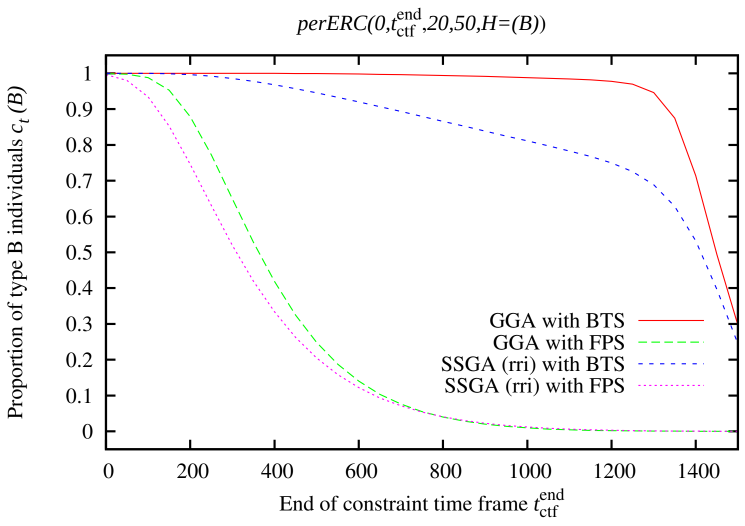


Fig. 12. Shown is the proportion of type B individuals $c_t(B)$ at selection step 1500 as a function of the end of the constraint time frame $t_{\text{ctf}}^{\text{end}}$.

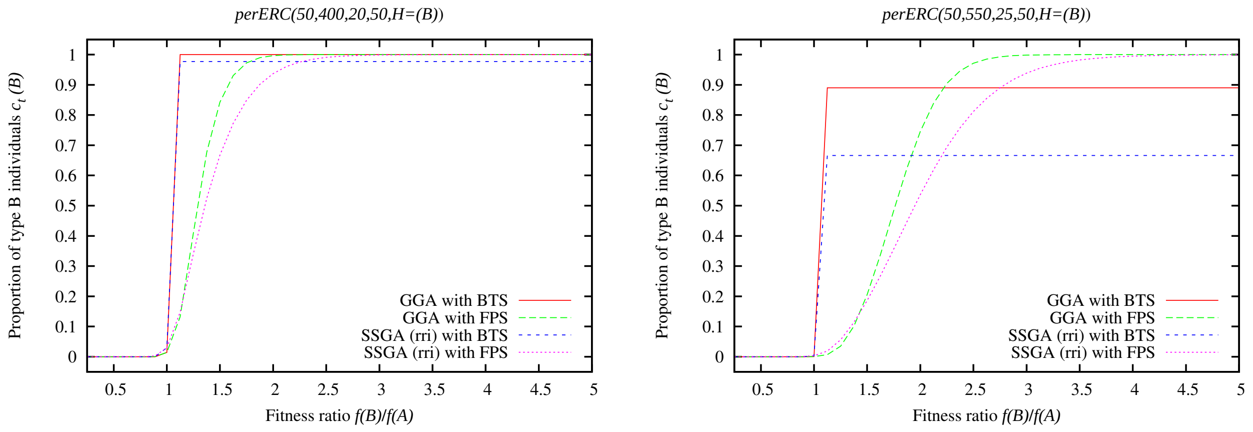


Fig. 13. Shown is the proportion of type B individuals $c_t(B)$ at selection step 1500 as a function of the fitness ratio $f(A)/f(B)$ for two periodic ERCs.

type B individuals of the current population, which slows down the convergence.

D. *Summary of theoretical study*

This section made use Markov chains to analyze the impact of periodic ERCs for a simple environment and EA model. The environment was composed of only two individual types and the EA model applied only a selection operator. In the EA model we considered two selection strategies, FPS and BTS, and two reproduction schemes, GGA and SSGA (rri). We observed that for one and the same reproduction scheme, BTS is more robust than FPS due to its independence to the fitness value of the individual types. However, FPS was able to match and even outperform the performance of BTS if the ratio of the individual type fitnesses was high, i.e. if a larger selection pressure than for BTS was obtained. The crucial difference between the two reproduction schemes is that GGA carries out many selection steps before the population is updated, while SSGA (rri), or steady state reproduction in general, carries out only a single one. This enables SSGA (rri) during the activation periods to replace less fit individuals with other less fit individuals of the current population, but also prevents SSGA (rri) on a long run from a quicker convergence in the remaining periods. By contrast, the performance of GGA depends linearly on the activation period but there are now drawbacks if the ERC is not activated. This crucial difference between the reproduction schemes means that SSGA (rri) is able to outperform GGA during the activation period and in situations where the advantage over GGA gained in the activation period(s) can be maintained until the next activation period or until the end of the optimization. In terms of the constraint parameter, this occurs when there is a long activation period, a short recovery period, and the constraint time frame is set late.

The findings gained in this theoretical investigation are valid for more complex environments and EA models too. Moreover, they are also largely valid for the remaining ERC types as we shall see in the experimental study. For this reason, we do not consider periodic ERCs in the experimental analysis, preferring to show the impact of other ERCs.

V. Experimental setup

Unlike the theoretical analysis, the experimental analysis is conducted in a more complex environment where the fitness value is obtained using a function f and more than two individual types exist, i.e. strings consist of more than 1 bit. The EAs are extended by additional operators too. This section describes the test functions, EAs and the parameter settings as used in the experimental analysis, which investigates the impact of the remaining three ERCs.

A. *Evolutionary algorithms*

To obtain better results on a variety of problem types, EAs have been often extended by performance enhancing mechanisms, such as diversity and niching mechanisms. As this paper aims to analyze the impact of ERCs on an EA's performance without having to account for disruptive effects coming from any additional mechanisms, we consider four standard EAs in the experimental analysis: (i) generational genetic algorithm (GGA), (ii) steady state genetic algorithm where a randomly selected individual is replaced (SSGA (rri)), (iii) steady state genetic algorithm where the worst individual is replaced (SSGA), and (iv) an EA using the $(\mu+\lambda)$ES selection and reproduction

TABLE I
CLASSIFICATION OF THE FOUR CONSIDERED EAS ACCORDING TO THEIR DISTINCTIVE FEATURES

	generational reproduction	steady state reproduction
non-*élitism* based optimization	GGA	SSGA (rri)
élitism based optimization	(μ+λ)ES	SSGA

scheme, but standard operators. All four EAs apply selection, mutation and crossover, and distinguish themselves in the reproduction scheme and/or in that they employ *élitism* or not; due to convenience reasons we equip all EAs with the same selection operator. A classification of the EAs according to these two features is given in Table I. We give a short description of each of the four EAs.

GGA [13] and (μ+λ)ES [14], [15] employ a generational reproduction scheme in the sense that they generate an entire offspring population, respectively, λ offspring from the individuals of the current population; μ refers to the population size of (μ+λ)ES. However, while GGA replaces the entire current population with the offspring population, (μ+λ)ES forms a new population by selecting the μ best individuals from the combined pool of offspring and current population. That is, (μ+λ)ES employs a mechanism that ensures that fit individuals found during the search are not lost. This mechanism is known as *élitism* [47].

As the name implies, the two steady state genetic algorithms employ a steady state reproduction scheme in the sense that they generate one offspring at a time. However, while SSGA [48] inserts this offspring into the population by replacing the worst individual, given the offspring is fitter, SSGA (rri) replaces a randomly selected individual from the population regardless of its fitness. Similar to (μ+λ)ES, SSGA employs a mechanism to maintain fit individuals found during the search.

The selection, crossover and mutation operators employed by all EAs and their parameter settings are outlined in Section V-C.

B. Test functions f

To date, a number of test functions posing different types of challenges to an optimization algorithm, such as non-convexity, mixed non-linearity, multiple objectives, deception, many local optima, and so on, have been proposed in the literature. However, despite the variety of test functions, to test a new algorithm or a new concept it is common to consider first rather simple and to an algorithm designer familiar functions. This allows for a more convenient and confident performance analysis and assessment.

We follow the common approach and select two rather simple and one more challenging unconstrained binary test function: (i) OneMax, (ii) TwoMax and (iii) a 3-SAT instance. We describe briefly all three functions.

OneMax

The *OneMax* function is a bit counting function that has become popular in the theoretical analysis of genetic algorithms (see e.g. [39], [49]). For a binary solution vector $\vec{x} \in \{0,1\}^l$, it takes the sum over the bit values of all bit positions

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

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

TwoMax

The *TwoMax* function is a bimodal function that can be seen as the multimodal counterpart of OneMax. It has two local optimal solutions: solutions consisting only of 1-bits and 0-bits. In its original version, the slopes leading to these two solutions are symmetric. As the OneMax function, this function has become popular in theoretical works on the analysis of genetic algorithms (see e.g. [50], [51]). To make the TwoMax function more realistic and interesting, its symmetric property has often been broken (see e.g. [39], [51], [52]), turning it into a function with a local and a global optimal solution. The two common modifications are to change either the steepness of either slope or to simply increase the fitness value of either optimal solution. We opt for the former and make the solution consisting only of 1-bits to a unique global optimum. Hence, if $\#1$s denotes the number of 1-bits in a solution vector $\vec{x}$ and b the factor by which the global optimal solution shall be fitter than the local optimal solution, then the TwoMax function is defined by

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

Figure 14 illustrates this TwoMax function.

MAX-SAT

Given a boolean expression consisting of a set of clauses, which in turn are formed by binary variables $x_i, i = 1, ..., l$, the *maximum satisfiability* (MAX-SAT) problem asks for the maximum number of clauses that can be satisfied by any assignment $\vec{x}$. A MAX-SAT problem is the optimization form of a *satisfiability* (SAT) decision problem, which only asks whether there is a satisfying assignment at all or not. SAT is the archetypal NP-complete optimization problem, while MAX-SAT is an NP-hard problem widely studied in optimization.

The single instance of MAX-SAT we consider is a uniform random 3-SAT problem and can be downloaded online.[7] The instance has 50 variables and 218 clauses and is satisfiable. We treat this 3-SAT instance as a MAX-SAT optimization problem, with fitness calculated as the proportion of satisfied clauses.

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

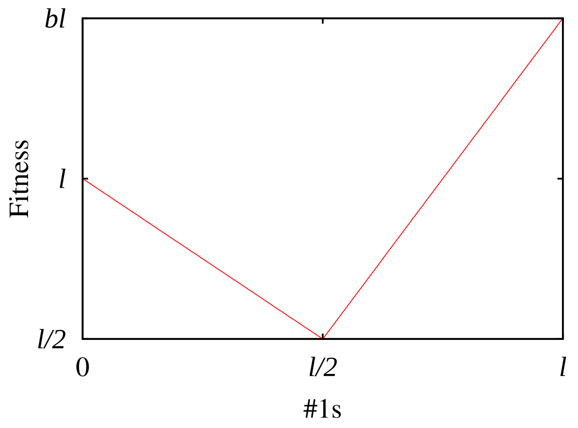


Fig. 14. TwoMax with a local optimal solution at #1s = 0 and a global optimal solution at #1s = l. The parameter b specifies the factor by which the global optimal solution shall be fitter than local optimal solution.

C. Parameter settings

We analyze the impact of three ERC types (commitment ERC, commitment composite ERC and random ERC) on the performance of four EAs ((μ+λ)ES, SSGA, GGA, and SSGA (rri)) for the OneMax and TwoMax function, and a 3-SAT problem instance. For the OneMax and TwoMax function all experiments will be performed using $l = 30$ solution parameters, while the 3-SAT instance requires $l = 50$. The scaling factor b for the TwoMax function will be set to $b = 1.1$, i.e. the global optimal solution has a fitness of 33 while the local one of 30.

We equip all four EAs with the same selection, mutation and crossover operators and keep their parameter settings fixed throughout the study. For selection we use binary tournament selection with replacement in the same way as in the theoretical analysis: two members of the current population are chosen at random with replacement and the fitter of the two becomes a parent; if both are equally fit then we chose a parent at random. This selection step is repeated to obtain two parents. With a probability of 0.7, the two parents are then recombined using uniform crossover [53]. Each of the two resulting offspring (which are simply copies of the two parents if crossover is not applied) is then mutated using bit flip mutation with a per-bit mutation rate of $1/l$. We also fix the population size of all EAs to $n = 50$ and set $\mu = \lambda = 50$, which makes the evolution strategy a (50+50)ES.

For the experimental study we use again the measure of time steps rather than selection steps. On OneMax and TwoMax all EAs are run for always $T = 1500$ time steps on the 3-SAT instance for $T = 15000$ time steps. Any results shown are average results across 500 independent algorithm runs. Recall that we use Lamarckian population update in this paper. Furthermore, we do not submit null solutions and use the communication channel in sequential experimentation mode ($batchSize = 1$), meaning one time step corresponds to a function evaluation of a single solution. But remember that the performance difference between batch and sequential experimentation can also be assessed by comparing the performance between the two reproduction schemes. The employed repairing strategies are specified in the experimental study for each of the ERC types separately.

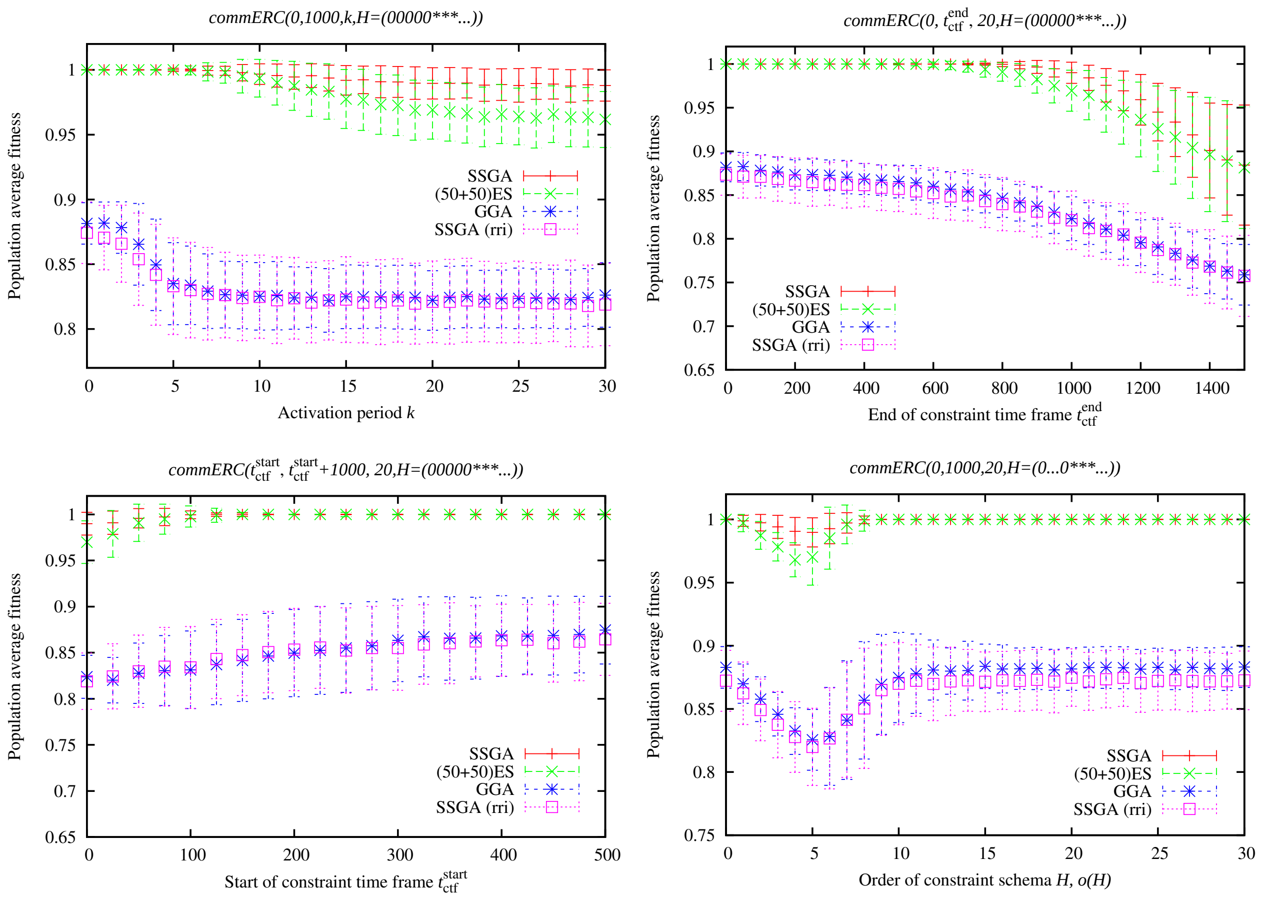


Fig. 15. Shown is the final population average fitness and its standard error on OneMax as a function of the activation period k (top left), the end and start of the constraint time frame, $t_{\text{ctf}}^{\text{end}}$ and $t_{\text{ctf}}^{\text{start}}$ (top right and bottom left), and the order of the constraint schema H, $o(H)$ (bottom right).

VI. Experimental study

Whether the effect of an ERC decreases or perhaps even improves an optimizer's performance depends predominantly on its impact on the population diversity and the optimization direction. To assess the impact on these two factors one has to answer questions like: (i) what genetic material represented by a constraint schema needs to be introduced into a population to cause a performance impact; (ii) how much of it, or, rather, how many individuals of a constraint schema need to be introduced into a population to cause a performance impact; (iii) when does it have to be introduced to yield a performance impact; (iv) how much preparation time and recovery time is available? This section analyzes these and related questions for commitment ERCs, commitment composite ERCs and random ERCs, using three test functions: OneMax, TwoMax and a 3-SAT problem instance.

A. Commitment ERC

Here, we repair a non-evaluable solution by simply forcing it into the constraint schema H, i.e. all bits that do not match the values of the order-defining bits of H are flipped.

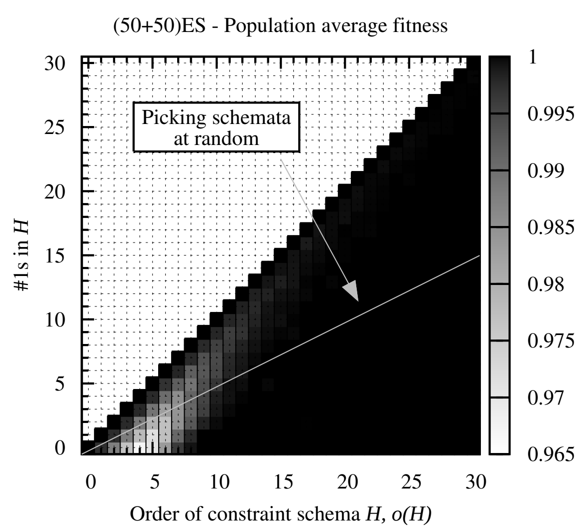
OneMax

Figure 15 shows how various constraint parameters of a commitment ERC may affect the performance of the EAs on OneMax. In this problem, the constraint schemata are of order 5 (the first five bits) and all order-defining bits of H are 0-bits (recall that this is represented by $H = (00000 * * * *...)$). That is, H represents genetic material that contradicts entirely an algorithm's optimization direction. The case where H represents different genetic material is investigated later.

From Figure 15, the following observations can be made.

- Longer activation periods k (top left plot) degrade performance, although a saturation point is reached beyond which further increases in activation period have no effect. The onset of the performance drop-off and the saturation point occur at higher k for the $\acute{e}$litist EAs, and performance is also maintained at a much higher level. $\acute{E}$litist EAs do not accept individuals from a poor schema into the population, hence it is unlikely for repaired individuals to enter the population; this has the effect of 'freezing' the optimization during the constrained time frame. For non-$\acute{e}$litist EAs, the population can fill up with repaired individuals from poor schema, a state from which it then takes longer to recover.
- Longer constraint time frames (leaving less time for recovery) affect all EAs negatively, but the $\acute{e}$litist EAs maintain their unconstrained performance level for much longer.
- With later start times of the constraint time frame there is a positive monotonic effect on the performance of all EAs. This effect is different to the one observed for the periodic ERC in the Markov chain study (where there was a trade off between more recovery time and more preparation time). This is because with a commitment constraint 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. Thus later constraints are less disruptive.
- There is a clear 'sweet-spot' in respect of the order of the schema which has maximum (negative) effect on the optimization. As the order $o(H)$ increases it becomes less likely that an optimizer activates the ERC at all; this is due to the generally lower probability of falling into a higher order schemata and the fact that the genetic material is even further away from an optimizer's search direction. However, on the other hand, although a low order increases the probability of generating individuals from H, it does not affect the search as much because fewer bits need to be optimized during the recovery period. With respect to these two aspects, an order of $o(H) \approx 5$ tends to have the worst trade off for an optimizer, as can be seen in the bottom right plot of Figure 15. Note, the order for which the worst trade off is obtained is a function of the string length l and the population size n. In general, the worst trade off shifts to only a slightly higher order than 5 as l and/or n increase. The reason for the shift is that a repaired individual is allowed to be poorer (i.e. $o(H)$ is allowed to be higher) as l and/or n increase but still have a high likelihood of being inserted into the population. However, the reason for the shift being only little is that the probability of activating the ERC decreases exponentially with the order.

Just as poor genetic material might prevent an optimizer from searching and evaluating of fitter individuals, good



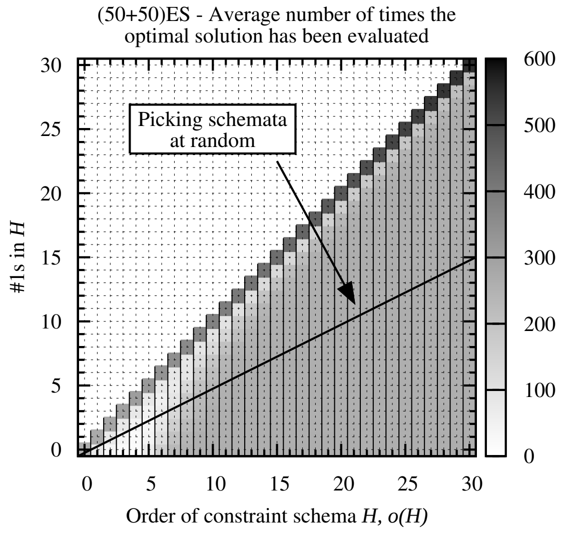


Fig. 16. Shown is the final population average fitness (left) at and the average number of times the optimal solutions has been evaluated during the optimization (right) for (50+50)ES on OneMax as a function of the order of the constraint schema H, $o(H)$, and the number of order-defining bits in H with value 1 for the ERC $commERC(0, 1000, 20, H)$. The straight line represents the expected performance when picking a schema for a particular order at random.

genetic material may be beneficial in terms of finding fitter individuals and finding them more quickly. The effect on the performance of constraint schemata that represent genetic material of different quality is shown in Figure 16.[8] The average number of times the optimal solution has been evaluated can be seen as a measure for the convergence speed, where a higher number indicates a quicker convergence. The performance of the other EAs, here not shown, follows a similar pattern.

At first glance, one can see from Figure 16 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. If the represented genetic material is optimal, i.e. $o(H) = \#1\text{s}$, then an activated ERC prevents the generation of individuals that are less fit with respect to the order-defining bits. That is, an optimizer benefits from such an ERC in terms of both the population average fitness and the convergence speed (this is seen by the increasingly dark shade in the elements when moving up the diagonal of the right plot of Figure 16).

Interesting is the effect of constraint schemata that represent near-optimal genetic material (seen best as a light

[8]If not otherwise stated, then both the order-defining bits and the positions of the 1-bits among the order-defining bits are chosen at random in the heat maps

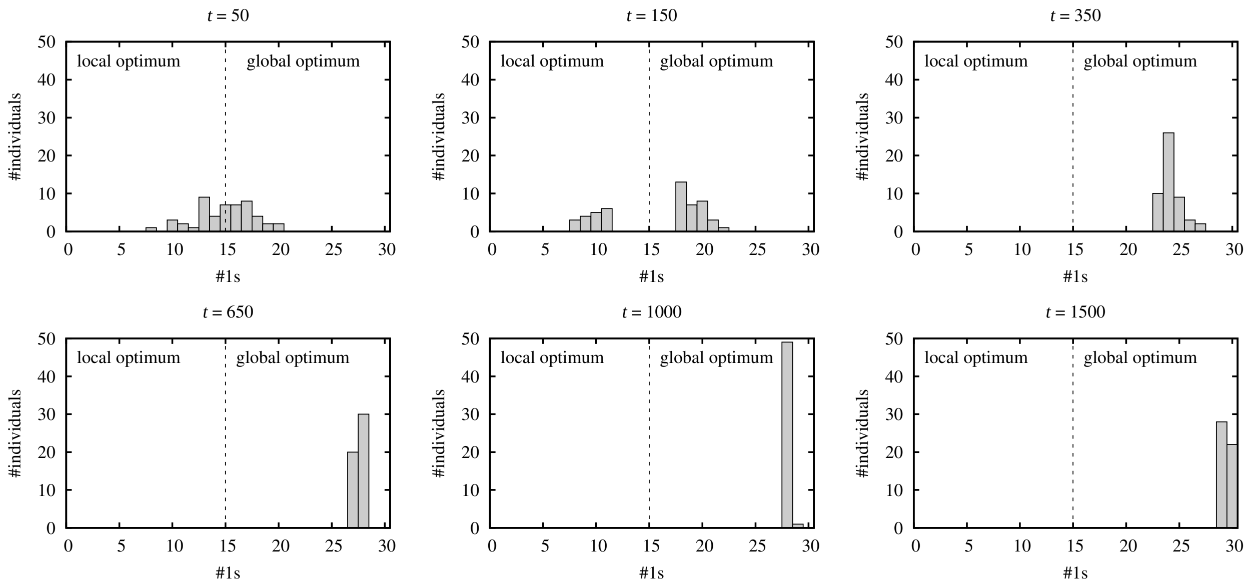


Fig. 17. Snapshots of a single run of (50+50)ES at different time steps t whilst optimizing TwoMax subject to the ERC $commERC(0, 1000, 20, H = (00 * * * ...))$. The horizontal axis represents the number of 1-bits in an individual's solution vector and the vertical axis the number of corresponding individuals in the population.

patch just below the diagonal in the right plot of Figure 16). Individuals falling into these schemata are likely to be fit and thus be inserted into the population. This causes the ERC to be activated over and over, which is initially beneficial, but at the same time this prevents an optimizer from generating even fitter individuals. Once the constraint is lifted, diversity needs to be again introduced among a few poor order-defining bits, which may not be possible during the recovery period and thus may result in a non-optimal performance. For GGA and SSGA (rri), for which the results are not shown here, near-optimal constraint schemata of medium and higher order may even improve the performance compared to the unconstrained performance. Responsible for this effect is their inability to find very fit individuals in an unconstrained search reliably. More precisely, the near-optimal genetic material stored in the population during the constraint time frame is used as a stepping stone to find fit individuals during the recovery period that would otherwise not be found.

TwoMax

An illustration of the effect of a commitment ERC on TwoMax is given in the form of histogram plots in Figure 17. The ERC considered has an activation period of $k = 20$ and a constraint schema of order $o(H) = 2$ that represents poor genetic material. The local and global optimum is located at $\#$1s=0 and $\#$1s=30, respectively. After the random initialization, the individuals consist on average of 15 1-bits. As the optimization proceeds, the selection pressure and the effect of drift push the entire population to either side, whereas the optimal slope is more likely to be climbed up. Since the constraint schema is of low order, it is very likely that the corresponding genetic material (regardless of its quality) is propagated throughout the population. Indeed, at time step $t = 1000$,

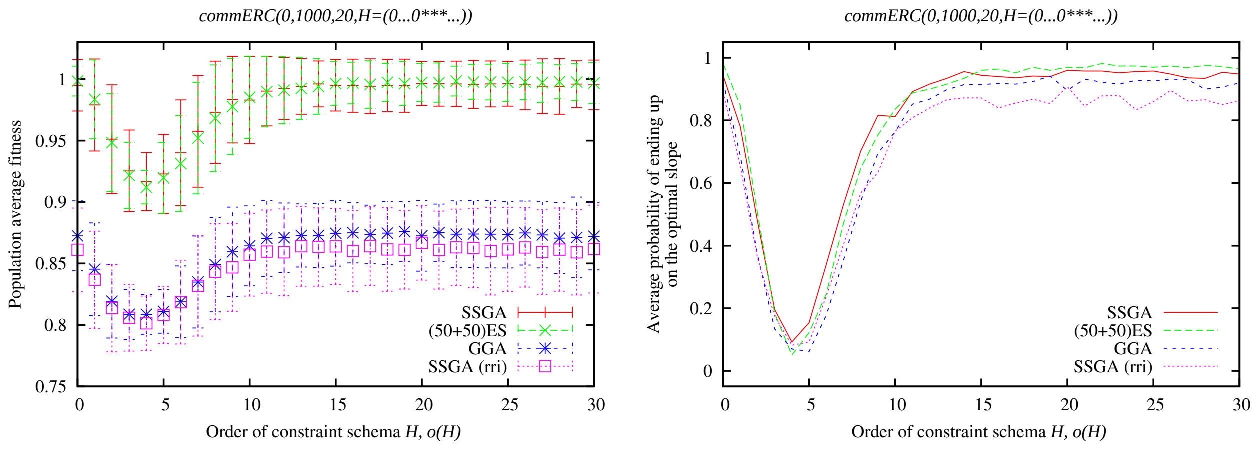


Fig. 18. Shown is the final population average fitness and its standard error (left) and the probability that the majority of the individuals in a population ends up on the optimal slope at the end of the optimization (right) on TwoMax as a function of the order of the constraint schema H, $o(H)$.

just before lifting the constraint, the population is nearly uniform with almost all individuals containing the poor genetic material. However, the provided recovery time is sufficient to introduce again diversity into the population and to eventually find the global optimal solution, though insufficient to converge to the optimal population.

We also measured for which constraint settings the probability of climbing up the optimal slope is affected. Once the majority or all individuals of a population are on a slope, then it is unlikely that individuals on the other slope are generated and thus that slopes are switched, in particular not if the local and global optimal solution have a similar fitness. Moreover, once a population has decided to climb up either slope, it behaves like on OneMax and the effect of an ERC on the performance is similar to that on OneMax too. Figure 18 demonstrates this for the impact of the order $o(H)$.

The left and right plot of Figure 18 show the effect of the order $o(H)$ on the population average fitness, respectively, the probability that the entire population is on the optimal slope at the end of the optimization. Once again, the largest performance impact is present for an order of $o(H) \approx 5$ and high-order constraint schemata have practically no impact at all. However, the population average fitness decreases more than on the OneMax problem, especially for SSGA and (50+50)ES. From the right plot, we can see that this is caused by the low probability of converging to the optimal slope. In fact, for $o(H) \approx 5$ this probability reduces for all EAs by a factor of almost ten compared to an unconstrained environment. The fact that (50+50)ES is able to outperform SSGA in the unconstrained case and for low-order constraint schemata is due to the higher probability of being on the optimal slope, which in turn is the result of its higher population diversity. On this simple bimodal problem, however, it seems that this property cannot yield a sufficient advantage over the quicker convergence speed of SSGA as the impact of the ERC becomes more severe, i.e. as the order $o(H)$ increases.

What is slightly different from the results obtained on OneMax is the effect of constraint schemata that represent non-poor genetic material. Figure 19 shows this effect for GGA in terms of the probability of climbing up the

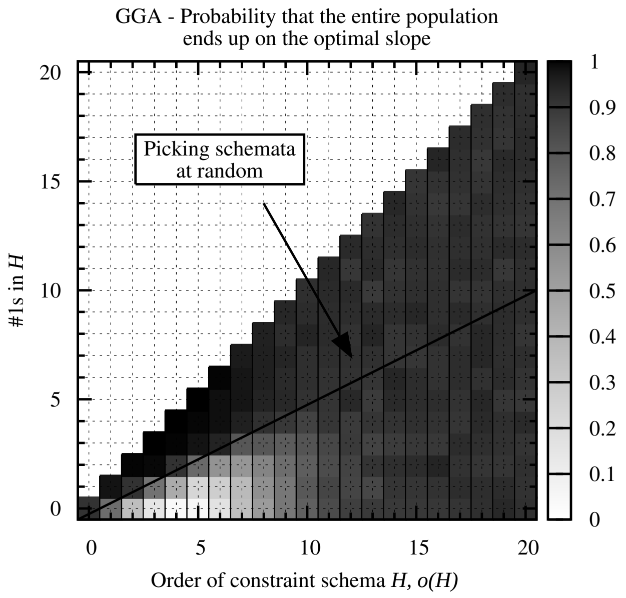


Fig. 19. Shown is the probability that the entire population ends up on the optimal slope at the end of the optimization for GGA on TwoMax as a function of the order of the constrained H, $o(H)$, and the number of order-defining bits in H with value 1 for the ERC $commERC(0, 1000, 20, H)$. The straight line represents the expected performance when picking a schema for a particular order at random.

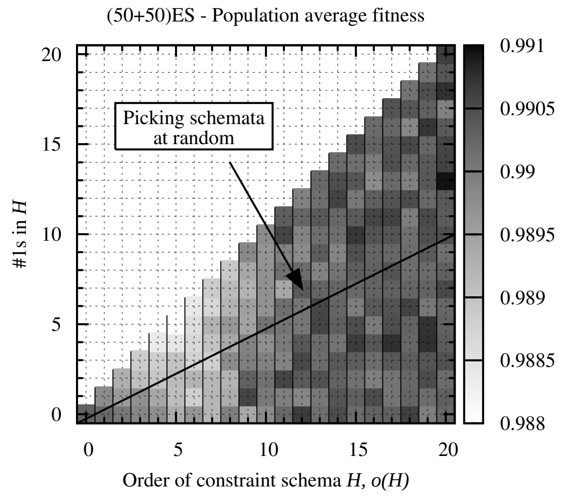
optimal slope;[9] the other EAs reacted similarly. Recall that the 'decision' to climb up either slope is made by an algorithm usually within the first few hundred time steps, as we have seen in the histogram plots. During this period, the population is most diverse and undecided in the search direction, which gives ERCs with low-order constraint schemata the highest chance to propagate their genetic material and thus to affect the search direction. As can be seen from the figure, this is exactly the range of orders that show an impact. As one would expect, constraint schemata with few 1-bits decrease the probability of converging to the optimal slope while schemata with many 1-bits increase it. The population average fitness, which is not shown here, changes accordingly.

3-SAT

Unlike OneMax and TwoMax, this problem consists of many local and global optima. Thus, it is interesting to analyze not only the population average fitness but also the proportion of runs that found an optimal solution; in the following, we will call such runs *successful runs*. Another difference to the other two problems is that the individual bits cannot be optimized independently. That is, the impact on the performance does not depend only on the order of a constraint schema and the number of order-defining bits being set to 1 but also on the positions of the order-defining bits within a bit string.

Figure 20 shows the population average fitness and the proportion of successful runs obtained with (50+50)ES as a function of the order $o(H)$ and the number of 1-bits among the order-defining bits. To keep the analysis as

[9]Results in Figure 19 are displayed only for an order of up to 20 as the performance was not affected for higher orders



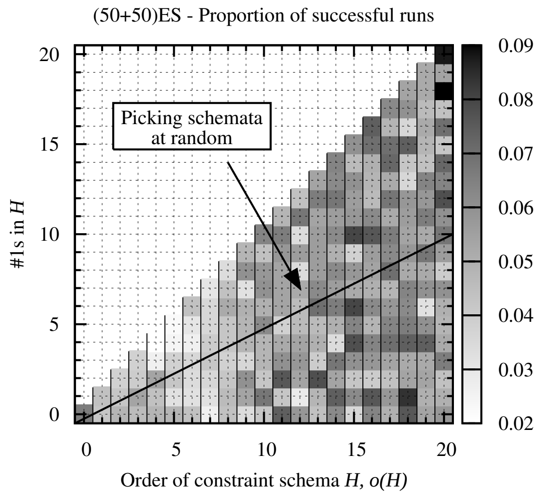


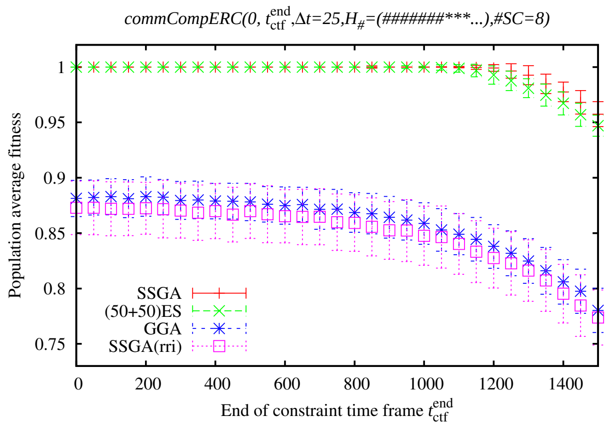
Fig. 20. Shown is the population average fitness (left) and the proportion of runs that found an optimal solution (right) for (50+50)ES on a 3-SAT problem instance as a function of the order of the constraint schema H, $o(H)$, and the number of order-defining bits in H with value 1 for the ERC $commERC(0, 7500, 20, H)$.

general as possible, we chose the order-defining bits in this experiment for each algorithm run at random. Similar to the results obtained on OneMax, the figure shows that constraint schemata of low order ($o(H) \leq 10$) affect the performance. Again, the reason is that constraint schemata of lower order are most likely to yield an activation regardless of an EA's optimization stage and the genetic material they are representing. Constraint schemata of higher order do not affect the performance because of the absence of unique optimal genetic material and the low probability of finding optimal genetic material at all.

As the positions of the order-defining bits are selected at random, the results allow us to make only general assumptions about the structure of the problem. For example, the low population average fitness obtained for constraint schemata of order $o(H) \approx 5$ consisting of many 1-bits may indicate that fit individuals are more likely to consist of more 0-bits than 1-bits, or that 1-bits are tricky to be set correctly. The uniform impact of low-order constraint schemata on the proportion of successful runs may indicate that the global optimal solutions consist of the same number of 1-bits and 0-bits.

The results for the other EAs, not shown here, feature a similar pattern as for (50+50)ES. However, the two non-*élitism*-based EAs perform better than the two *élitism*-based EAs with respect to the proportion of successful runs but not to the population average fitness.

The effect on the performance of the remaining constraints parameter is in alignment with the results obtained for OneMax: commitment ERCs with low-order constraint schemata harm the performance more severely as the activation period and the constraint time frame is extended or the recovery time shortened.



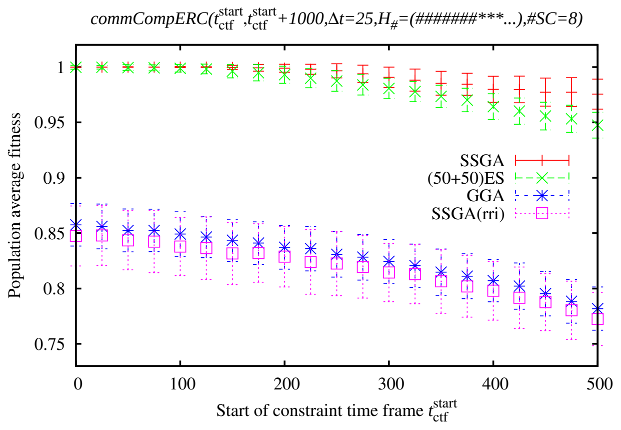


Fig. 21. Shown is the final population average fitness and its standard error on OneMax as a function of the end and start of the constraint time frame, $t_{\text{ctf}}^{\text{end}}$ and $t_{\text{ctf}}^{\text{start}}$.

B. Commitment composite ERC

For this ERC type, we use an unbiased ordering and repairing strategy. In fact, we order random composites (i.e. composite-defining bits are 0 or 1 with equal probability) whenever possible (i.e. after every Δt time steps). For repairing, we select a random composite from the set of currently stored composites, and use this as a template to specify the composite-defining bits. Note, with these two strategies, a composite will consist in average of the same number of 1-bits and 0-bits, so will a repaired individual with respect to the composite-defining bits.

OneMax

The key observations for this ERC type on OneMax are as follows.

- The longer the constraint time frame the worse the performance of all EAs, but the effect is much less with *é*litism (Figure 21, left). This result is very similar to what we observe in Figure 15 for the commitment constraint.
- With respect to the start time of the constraint time frame we see the *opposite* effect to that observed on the commitment ERC. Here, it is worse to delay the constraint time frame because this gives less time for recovery; the preparation time has little positive influence and so does not offset having little time for recovery.

These results, particularly the positive effect of *é*litism, can be understood quite easily. Since repaired individuals will generally be poor (considering that the composites ordered in our setup are made of random bits), a non-*élitist* EA will finish the constraint time frame with many poor individuals in its population, and have much to do during the recovery period. An *élitist* EA, in contrast, will only allow fit repaired individuals to enter the population and so will either (i) maintain a memory of (unconstrained) good solutions generated during the preparation time, or (ii) find and maintain repaired individuals of high fitness, which occasionally occur due to ‘lucky’ orders having large numbers of 1-bits in the composite-defining bits.

Figure 22 shows the effect of Δt on the population average fitness of (50+50)ES for different orders $o(H_{\#})$.

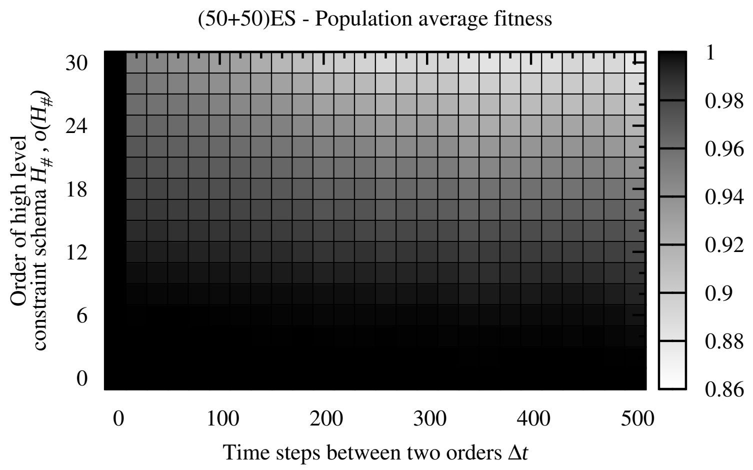


Fig. 22. Shown is the final population average fitness of (50+50)ES on OneMax as a function of the number of time step between two orders Δt and the order of the high level constraint schema $H_\#$, $o(H_\#)$, for the ERC $commCompERC(0, 1000, \Delta t, H_\# = (\#...\#{*}{*}{*}...), \#SC = 8)$. The column $\Delta t = 0$ and the row $o(H_\#) = 0$ represent the population average fitness obtained in an ERC free environment.

One can see that ordering composites more frequently can compensate slightly the negative effect coming from an increasing order. The small number of storage cells prevents a better performance for medium and high orders. Note, a higher number of composites tested during the constraint time frame allows also for a higher population diversity with respect to the composite-defining bits and not only a higher probability of selecting a good composite. Being more diverse with respect to the composite-defining bits means that a shorter recovery period is needed to introduce diversity among these bits before fitter individuals can be generated. Thus, a low diversity contributes to the poor performance of (50+50)ES for large Δt too.

The results for the remaining EAs are not shown here. But SSGA tends to perform slightly better than (50+50)ES, which is due to its faster convergence, an important property on this problem. The lack of memory of the two non-*élitism*-based EAs causes their performance to be predominantly a function of the order. The number of time steps between two orders plays only a role if it can affect the population diversity with respect to the composite-defining bits in the final population before lifting the constraint.

TwoMax

On TwoMax, a commitment composite ERC enables an optimizer to evaluate repaired individuals from either slope for a longer period of time. In fact, for $o(H_\#) > l/2$, repaired individuals from either slope might be evaluated throughout the constraint time frame. With *élitism*, this gives an optimizer the opportunity to build up a bias towards the optimal slope through both the composite-defining bits and the unconstrained bits. In particular, an optimizer is given the chance to re-optimize the values of the unconstrained bits if they represent suboptimal genetic material. The resulting bias increases then also the probability that a population climbs up the optimal slope during the recovery period (see Figure 23). For this to take place, however, apart from a high order $o(H_\#)$,

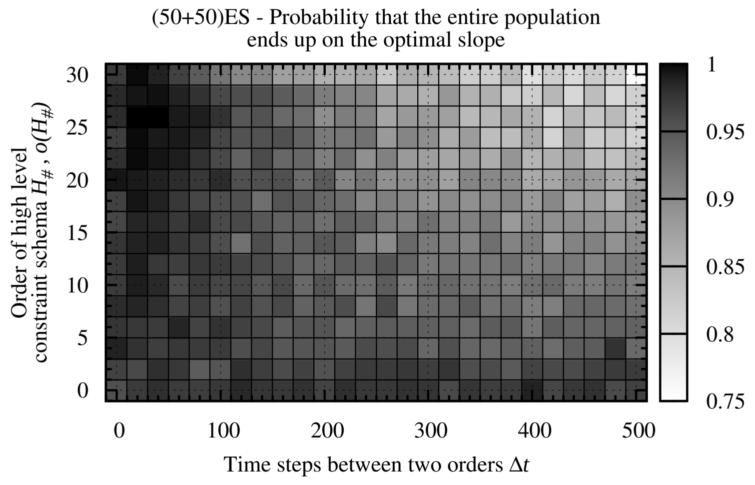


Fig. 23. Shown is the probability that the entire population ends up on the optimal slope at the end of the optimization for GGA on TwoMax as a function of the time step between two orders Δt and the order of the high level constraint schema $H_{\#}$, $o(H_{\#})$, for the ERC $commCompERC(0, 1000, \Delta t, H_{\#} = (\#...\# * * * ...), \#SC = 8)$.

two factors need to be present: (i) a high probability of encountering a non-evaluable individual in order to affect the optimization at all and (ii) a preferably large number of different composites tested during the constraint time frame in order to increase the probability of selecting many composites with more 1-bits and potentially being able to reverse a suboptimal search direction; the constraint time frame should also be set as early as possible because less bias is required to reverse the search direction of a less optimized population and the probability of a repaired individual being inserted into the population is higher too. In Figure 23, these factors are fulfilled making a performance advantage apparent for $o(H_{\#}) \geq 15$ and $\Delta t \leq 100$ (the column $\Delta t = 0$ and row $o(H_{\#}) = 0$ indicate the unconstrained performance). Note, the drop in the performance for large Δt and high orders $o(H_{\#})$ is due to two reasons: the low number of different composites available during the constraint time frame and the low population diversity with respect to the composite-defining bits, which might prevent an optimizer from converging on either slope during the recovery period.

Without *élitism*, results not shown here, it is difficult to optimize the composite-defining bits during the constraint time frame, especially if many individuals need to be repaired. Hence, a bias in the final population before lifting the constraint towards either slope needs to come from the unconstrained bits. Thanks to selection pressure, these are slightly more likely to represent optimal genetic material (if a sufficient number of different composites is available during the constraint time frame) and thus contribute to a slightly higher probability of climbing up the optimal slope during the recovery period; obviously, the advantage is not as significant as with *élitism*.

3-SAT

The presence of multiple local and global optima makes it rather unlikely that a bias towards a single solution is built up during the constraint time frame. The fact that the unconstrained bits cannot be optimized independently

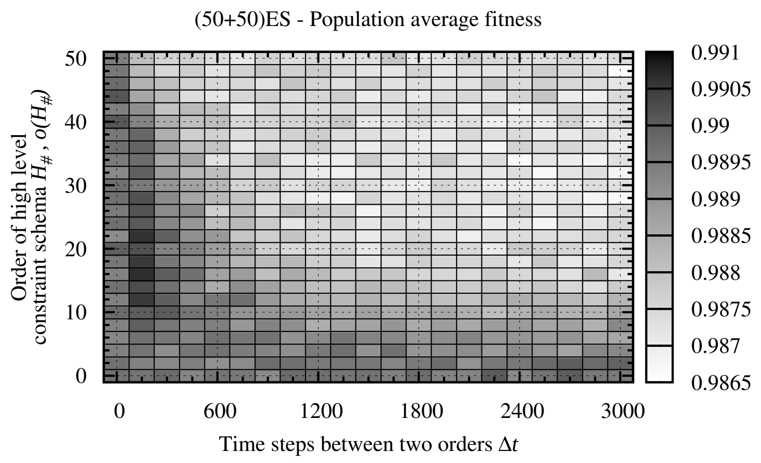


Fig. 24. Shown is the final population average fitness for (50+50)ES on a 3-SAT problem instance as a function of number of time steps between two orders Δt and the order of the high level constraint schema $H_\#$, $o(H_\#)$, for the ERC $commCompERC(0, 7500, \Delta t, H_\# = (\#...\# * * * ...), \#SC = 8)$.

from the composite-defining bits might complicate this process further.

Figure 24 shows how the population average fitness of (50+50)ES is affected as a function of Δt and $o(H_\#)$; the number of storage cells is kept small, increasing the probability of an individual being non-evaluable. Once again, with *élitism* an optimizer benefits from ordering new composites more frequently. The reason is similar as on the other problems: suitable composites are more likely to be selected throughout the constraint time frame, allowing an optimizer to fill the population with fit individuals that are at the same time diverse at least with respect to the composite-defining bits. These individuals help then an optimizer during the recovery period to find even fitter individuals without getting trapped at local optima too quickly. For this, however, the recovery period needs to be of sufficient length.

C. Random ERC

Here, we restrict ourselves to ERCs with constraint schemata of order one (for an example see Figure 6 in Section III-B). In a binary search space, this makes repairing straightforward.

OneMax

Increasing activation probability p and activation period k both lead to an increased numbers of bits constrained per time step. This slows convergence during the constraint time frame, in a similar way to increasing the order $o(H_\#)$ of a commitment composite ERC (cf. Figure 24), leading to a drop off in final fitness achieved (Figure 25).

However, although a large value of p has a generally detrimental effect on performance, it can lead to an increased diversity in the population of an EA. This is seen in Figure 26, where the population diversity is measured using pair-wise Hamming distance (see e.g. [54]). The reason for the effect, which occurs for approx $k \leq 20$, is that a larger number of different bits is constrained to different values, but only for a short period. This is somewhat akin

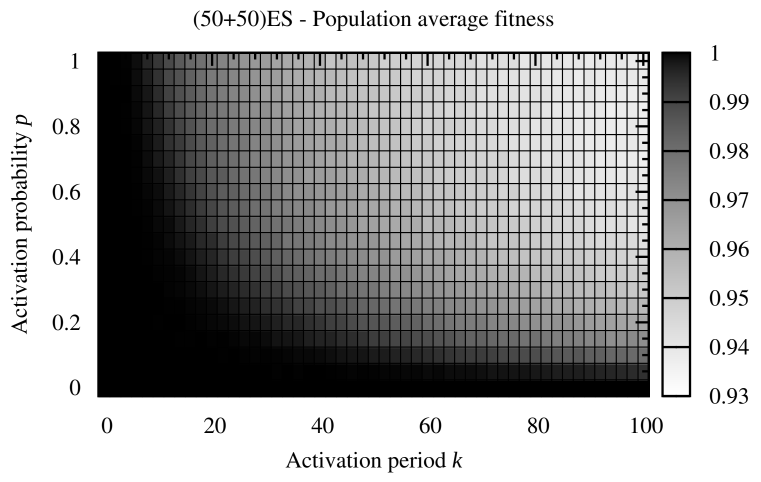


Fig. 25. Shown is the final population average fitness for (50+50)ES on OneMax as a function of the activation period k and the activation probability p for the ERC $randomERC(0, 1000, k, o(H) = 1, p)$.

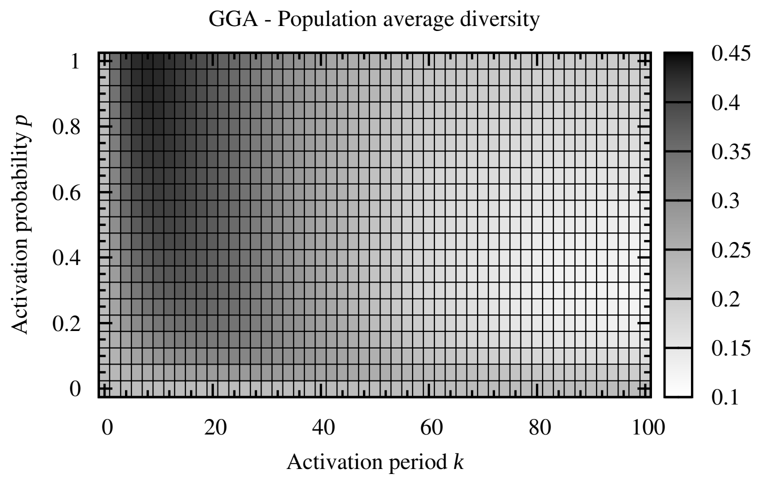


Fig. 26. Shown is the population average diversity for GGA during the constraint time frame on OneMax as a function of the activation period k and the activation probability p.

to a high mutation rate.

We can see in Figure 26 that for $k > 40$, an increase in p has first a negative and then at the higher levels a positive effect on diversity. This pattern can be understood if one considers that for large values of k most (or all) individuals may be constrained: then increasing p slightly from a low level will tend to add more constraints (more constrained bits) and so reduce diversity of the population; when increasing p from an already high level, an already constrained bit may be re-constrained (flipped), which increases diversity. These effects on diversity, whilst not important on OneMax, do seem to affect performance on the multimodal problems, TwoMax and Max-SAT considered next.

The effect of shifting the constraint time frame is not shown but, similarly to a commitment composite ERC, the performance reduces the later the constraint time frame is set. The reasons are similar too. With *élitism*, the

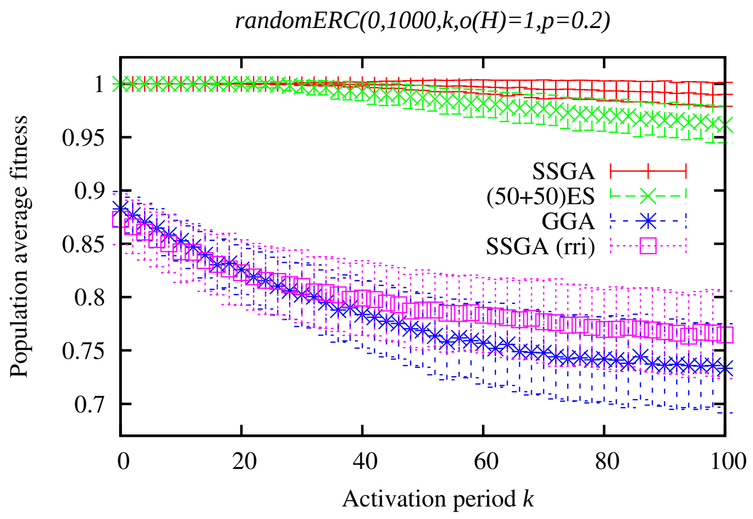


Fig. 27. Shown is the final population average fitness and its standard error on OneMax as a function of the activation period k.

advantage gained during a longer preparation period is insufficient to compensate the drawback coming from a shorter recovery period; also here, the population tends to freeze during a late constraint time frame because the quality of the constrained bits remains the same. Without *élitism*, the required recovery period is always the same. Consequently, the shorter this period the poorer the performance.

TwoMax

Figure 28 shows the effect of the activation period k and the activation probability p on the probability of (50+50)ES to climb up the optimal slope. A correlation with the available population diversity (which was shown in Figure 26) is apparent. In fact, a lower population diversity tends to reduce the probability of climbing up the optimal slope. The reason is that, in the process of building up a bias towards either slope, the values of the constrained bits gain in importance as less diversity is available. Of course, on average these values are unbiased. However, in the case where more bits are constrained to the same values, it is more likely that a bias is built up towards the corresponding slope. Compared to an unconstrained optimization, this increases the probability that the suboptimal slope is climbed up.

Without *élitism* (results not shown here), the higher population diversity available in the range $40 \leq k \leq 100$, $0.4 < p \leq 1$ does not significantly improve the probability of climbing up the optimal slope compared to performance in the range $40 \leq k \leq 100$, $0 < p < 0.4$. The reason is simply that the resulting larger number of constrained bits in combination with the lack of memory leaves less space for optimization during the constraint time frame. The search direction followed by an optimizer during the recovery period depends mainly on the quality of the composites selected to generate the final population before lifting the constraint.

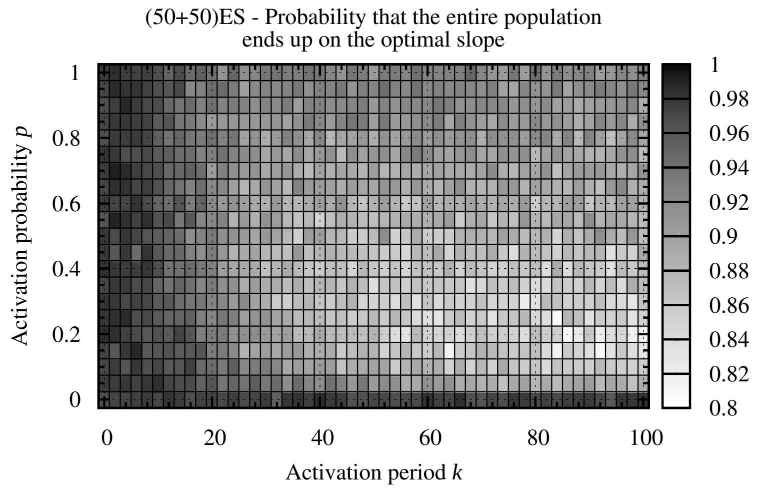


Fig. 28. Shown is the probability that the entire population ends up on the optimal slope at the end of the optimization for (50+50)ES on TwoMax as a function of the activation period k and activation probability p for the ERC $randomERC(0, 1000, k, o(H) = 1, p)$.

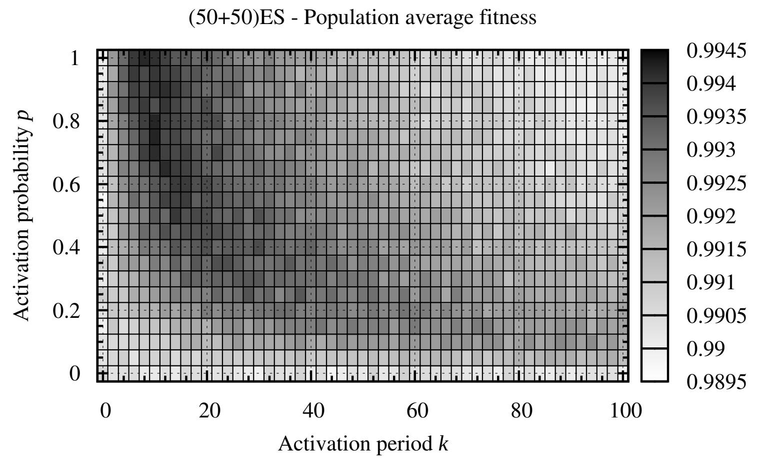


Fig. 29. Shown is the final population average fitness for (50+50)ES on a 3-SAT problem instance as a function of the activation period k and activation probability p for the ERC $randomERC(0, 7500, k, o(H) = 1, p)$.

3-SAT

Getting trapped at local optima and subsequently losing population diversity is a common situation on this multimodal problem, especially when *élitism* is used. Additional diversity coming from a random ERC may help an optimizer to jump out from a local optimum and to find even better search regions. In Figure 29, where the population average fitness of (50+50)ES is shown as a function of the activation period k and the activation probability p, one can see that the population average fitness improves significantly in the range $0 < k \leq 40$ as p increases (compare with Figure 26). Here, the constraint enables an optimizer to find good search regions that would otherwise not be found.

VII. Summary and conclusion

Any optimization problem that involves resources, such as time, money, instruments, skilled engineers, patients, and so forth, in the evaluation of solutions may also be subject to temporary non-availabilities of these resources. In such situations, solutions that are otherwise *feasible* may not be *evaluable* temporarily. So far, in published work, problems with these constraints have not been considered. In this study, we have provided an appropriate general problem formulation and termed such problems *ephemeral resource-constrained optimization problems* (ERCOPs). We have indicated that population-based optimization algorithms like *evolutionary algorithms* (EAs) are suitable optimizers for ERCOPs. We then have proposed an initial set of four *ephemeral resource constraint* (ERC) types which model various forms of temporary resource limitations; these ERCs were periodic ERCs, commitment ERCs, commitment composite ERCs, and random ERCs. To analyze the impact of these ERCs on evolutionary search, we engaged in a theoretical and experimental investigation.

The theoretical investigation has used Markov chains to study the effect of periodic ERCs on two selection strategies and reproduction schemes commonly-used within EAs. One conclusion of the analysis was that (binary) tournament selection is in general more robust to the restrictions imposed by the ERC than fitness proportionate selection, which tends to perform better only when the difference in the fitness between fit and poor individuals is large. Another conclusion was that a non-*é*litist steady state reproduction scheme is superior to a standard generational one during the activation periods of an ERC and when the restriction imposed by an ERC on the optimization is severe, i.e. when the activation period and the constraint time frame is long and little or no recovery time is available.

The experimental investigation has studied the effect of the other ERC types on four EAs that differed in their reproduction scheme and in whether they employ *élitism* or not. The experimental analysis confirmed the findings from the theoretical analysis and revealed parameter settings of the individual ERCs that seem to be challenging to an EA. Commitment ERCs with a constraint schemata of low order and/or constraint schemata that represent near optimal genetic material seem to be challenging as they may cause a population, if activated at the appropriate optimization stage, to become uniform with respect to poor order-defining bits. Commitment composite ERCs might be challenging if an optimizer is not given the opportunity to try out many composites and/or the number of storage cells is small as this may slow down the convergence and cause a population to get trapped at local optima. A lack of *élitism* is rather harmful in the presence of this ERC type. This is certainly the case when dealing with random ERCs. With this ERC type, the convergence is slowed down too, especially if the average number of constrained bits is large or the activation period long.

As all ERC types reduce the convergence speed and/or require an optimizer to wait for mutation to introduce again diversity into the population among previously constrained bit positions, generally speaking, the performance reduces as less recovery time is available. But we also observed examples where recovery time was less important than the preparation time, and this was easily understood from considering the interaction between the EA, problem and ERC. Resource constraints did not uniformly degrade performance either: sometimes they caused useful additional diversity, as we saw with 3-SAT and the random ERCs; alternatively the constraints can be aligned with the optimization goal, which will also tend to help search.

An initial suite of test problems resulting from this study including the communication channels of all ERCs is available online.[10]

Further research into the effects of ERCs is needed. One of our main intentions is to consider non-homogeneous experimental costs and to understand how these yield challenging problem instances. Another is to eventually develop effective and efficient search policies, or components for their design, for the optimization of ERCOPs. Here, inspiration may be drawn, for example, from strategies used in online optimization, and dynamic scheduling, and memory-based EA methods including diploid representations [55]; online optimization can be helpful in the decision of upcoming resource orders, while scheduling and memory-based methods can be helpful if (future) activation periods are known, respectively, to make use of previously-found but temporarily non-evaluable individuals. Another important avenue we are pursuing is to consider learning techniques, such as reinforcement learning [56], in combination with prediction models to make search strategies more versatile and robust to environmental changes; for example, one can learn for which decisions an ERC was activated and/or negative performance impact was associated with it. Finally, due to the nature of experimental optimization, ERCOPs may not only be subject to ERCs but also to noise, uncertainty and uncontrolled factors. Moreover, the aim is often to optimize several (conflicting) objectives simultaneously. Another future aspect is to extend search policies developed for ERC to account also for these additional factors.

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

[1] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 67–82, 1997.

[2] J. D. Knowles, "Closed-loop evolutionary multiobjective optimization," *IEEE Computational Intelligence Magazine*, vol. 4, no. 3, pp. 77–91, 2009.

[3] Z. S. Davies, R. J. Gilbert, R. J. Merry, D. B. Kell, M. K. Theodorou, and G. W. Griffith, "Efficient improvement of silage additives by using genetic algorithms," *Applied and Environmental Microbiology*, vol. 66, no. 4, pp. 1435–1443, 2000.

[4] D. Weuster-Botz and C. Wandrey, "Medium optimization by genetic algorithm for continuous production of formate dehydrogenase," *Process Biochemistry*, vol. 30, no. 6, pp. 563–571, 1995.

[5] R. D. King, K. E. Whelan, F. M. Jones, P. G. K. Reiser, C. H. Bryant, S. H. Muggleton, D. B. Kell, and S. G. Oliver, "Functional genomic hypothesis generation and experimentation by a robot scientist," *Nature*, vol. 427, pp. 247–252, 2004.

[6] S. O'Hagan, W. B. Dunn, J. D. Knowles, D. Broadhurst, R. Williams, J. J. Ashworth, M. Cameron, and D. B. Kell, "Closed-loop, multiobjective optimization of two-dimensional gas chromatography/mass spectrometry for serum metabolomics," *Analytical Chemistry*, vol. 79, no. 2, pp. 464–476, 2007.

[10] http://www.cs.manchester.ac.uk/~allmendr/software.html

[7] S. O'Hagan, W. B. Dunn, M. Brown, J. D. Knowles, and D. B. Kell, "Closed-loop, multiobjective optimization of analytical instrumentation: gas chromatography/time-of-flight mass spectrometry of the metabolomes of human serum and of yeast fermentations," *Analytical Chemistry*, vol. 77, no. 1, pp. 290–303, 2005.

[8] P. K. Wong, F. Yu, A. Shahangian, G. Cheng, R. Sun, and C.-M. Ho, "Closed-loop control of cellular functions using combinatory drugs guided by a stochastic search algorithm," *Proceedings of the National Academy of Sciences*, vol. 105, pp. 5105–5110, 2008.

[9] D. Calzolari, S. Bruschi, L. Coquin, J. Schofield, J. D. Feala, J. C. Reed, A. D. McCulloch, and G. Paternostro, "Search algorithms as a framework for the optimization of drug combinations," *PLoS Computational Biology*, vol. 4, p. 1, 2008.

[10] O. Gobin, A. M. Joaristi, and F. Schüth, "Multi-objective optimization in combinatorial chemistry applied to the selective catalytic reduction of NO with C_3H_6," *Journal of Catalysis*, 2007.

[11] O. M. Shir, T. Bäck, H. Rabitz, and M. J. J. Vrakking, "On the evolution of laser pulses under a dynamic quantum control environment," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2008, pp. 2127–2134.

[12] O. M. Shir, M. Emmerich, T. Bäck, and M. J. J. Vrakking, "The application of evolutionary multi-criteria optimization to dynamic molecular alignment," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2007, pp. 4108–4115.

[13] J. H. Holland, *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.

[14] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973.

[15] H.-P. Schwefel, "Evolutionsstrategie und numerische optimierung," Ph.D. dissertation, Technical University of Berlin, 1975.

[16] H.-G. Beyer, "Evolutionary algorithms in noisy environments: Theoretical issues and guidelines for practice," *Computer Methods in Applied Mechanics and Engineering*, vol. 186, pp. 239–267, 2000.

[17] J. D. Knowles, D. W. Corne, and A. Reynolds, "Noisy multiobjective optimization on a budget of 250 evaluations," in *Proceedings of the International Conference on Evolutionary Multi-Criterion Optimization*, 2009, pp. 36–50.

[18] Z. Michalewicz and M. Schoenauer, "Evolutionary algorithms for constrained parameter optimization problems," *Evolutionary Computation*, vol. 4, no. 1, pp. 1–32, 1996.

[19] C. A. C. Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art," *Computer Methods in Applied Mechanics and Engineering*, vol. 191, no. 11-12, pp. 1245–1287, 2002.

[20] J. Branke, *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, 2001.

[21] K. Miettinen, *Nonlinear multiobjective optimization*. Kluwer Academic Publishers, 1999.

[22] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, 2001.

[23] J. D. Knowles, D. W. Corne, and K. Deb, *Multiobjective Problem Solving from Nature*. Springer, 2008.

[24] I. Rechenberg, "Case studies in evolutionary experimentation and computation," *Computer Methods in Applied Mechanics and Engineering*, 2000.

[25] H.-P. Schwefel, "Experimentelle optimierung einer Zweiphasendüse," Bericht 35 des AEG-Forschungsinstituts Berlin zum Projekt MHD-Staustrahlrohr, 1968.

[26] E. Byrne, "Optimising the flow of experiments to a robot scientist with multi-objective evolutionary algorithms," in *Proceedings of Genetic and Evolutionary Computation Conference*, 2007, pp. 2429–2436.

[27] G. E. P. Box, J. S. Hunter, and W. G. Hunter, *Statistics for Experimenters: Design, Innovation, and Discovery*, 2nd ed. Wiley, 2005.

[28] D. C. Montgomery, *Design and Analysis of Experiments*. Wiley, 1976.

[29] R. L. Mason, R. F. Gunst, and J. L. Hess, *Statistical Design and Analysis of Experiments: With Applications to Engineering and Science*. Wiley, 1989.

[30] P. A. N. Bosman, "Learning, anticipation and time-deception in evolutionary online dynamic optimization," in *Proceedings of Genetic and Evolutionary Computation Conference*, 2005, pp. 39–47.

[31] P. A. N. Bosman and H. L. Poutré, "Learning and anticipation in online dynamic optimization with evolutionary algorithms: the stochastic case," in *Proceedings of Genetic and Evolutionary Computation Conference*, 2007, pp. 1165–1172.

[32] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, 1999.

[33] T. T. Nguyen and X. Yao, "Benchmarking and solving dynamic constrained problems," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2009, pp. 690–697.

[34] G. E. P. Box, "Evolutionary operation: A method for increasing industrial productivity," *Applied Statistics*, vol. 6, no. 2, pp. 81–101, 1957.

[35] C. R. Reeves and J. E. Rowe, *Genetic algorithms–Principles and Perspectives: A guide to GA theory*. Kluwer Academic Publishers, 2003.

[36] J. R. Norris, *Markov Chains (Cambridge Series in Statistical and Probabilistic Mathematics)*. Cambridge University Press, 1998.

[37] T. Nakama, "Theoretical analysis of genetic algorithms in noisy environments based on a markov model," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2008, pp. 1001–1008.

[38] J. Horn, "Finite Markov chain analysis of genetic algorithms with niching," in *Proceedings of the International Conference on Genetic Algorithms*, 1993, pp. 110–117.

[39] J. He and X. Yao, "From an individual to a population: an analysis of the first hitting time of population-based evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 495–511, 2002.

[40] J. L. Doob, *Stochastic Processes*. Wiley, 1953.

[41] D. E. Goldberg and P. Segrest, "Finite Markov chain analysis of genetic algorithms," in *Proceedings of the International Conference on Genetic Algorithms*, 1987, pp. 1–8.

[42] S. W. Mahfoud, "Finite markov chain models of an alternative selection strategy for the genetic algorithm," *Complex Systems*, vol. 7, pp. 155–170, 1991.

[43] M. D. Vose and G. E. Liepins, "Punctuated equilibria in genetic search," *Complex Systems*, vol. 5, pp. 31–44, 1991.

[44] A. Nix and M. D. Vose, "Modeling genetic algorithms with Markov chains," *Annals of Mathematics and Artificial Intelligence*, vol. 5, pp. 79–88, 1992.

[45] T. E. Davis and J. C. Principe, "A Markov chain framework for the simple genetic algorithm," *Evolutionary Computation*, vol. 1, no. 3, pp. 269–288, 1993.

[46] G. Syswerda, "A study of reproduction in generational and steady state genetic algorithms," in *Foundations of Genetic Algorithms*, 1991, pp. 94–101.

[47] K. A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation, University of Michigan, 1975.

[48] D. Whitley, "The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best," in *Proceedings of the International Conference on Genetic Algorithms*, 1989, pp. 116–121.

[49] H. Mühlenbein and D. Schlierkamp-Voosen, "Optimal interaction of mutation and crossover in the breeder genetic algorithm," in *Proceedings of the International Conference on Genetic Algorithms*, 1993, p. 648.

[50] C. V. Hoyweghen, D. E. Goldberg, and B. Naudts, "From twomax to the ising model: Easy and hard symmetrical problems," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2002, pp. 626–633.

[51] M. Pelikan and D. E. Goldberg, "Genetic algorithms, clustering, and the breaking of symmetry," in *Proceedings of Parallel Problem Solving from Nature*, 2000, pp. 385–394.

[52] T. Friedrich, P. S. Oliveto, D. Sudholt, and C. Witt, "Theoretical analysis of diversity mechanisms for global exploration," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2008, pp. 945–952.

[53] G. Syswerda, "Uniform crossover in genetic algorithms," in *Proceedings of the International Conference on Genetic Algorithms*, 1989, pp. 2–9.

[54] J. Horn, "The nature of niching: Genetic algorithms and the evolution of optimal, cooperative populations," Ph.D. dissertation, University of Illinois, Urbana, Illinois, 1997.

[55] D. E. Goldberg and R. Smith, "Nonstationary function optimization using genetic algorithm with dominance and diploidy," in *Proceedings of the International Conference on Genetic Algorithms*, 1987, pp. 59–68.

[56] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.