Chapter 9
Theoretical Advances in Evolutionary Dynamic Optimization

Philipp Rohlfshagen, Per Kristian Lehre, and Xin Yao

Abstract. The field of evolutionary dynamic optimization is concerned with the study and application of evolutionary algorithms to dynamic optimization problems: a significant number of new algorithms have been proposed in recent years that are designed specifically to overcome the limitations faced by traditional algorithms in the dynamic domain. Subsequently, a wealth of empirical studies have been published that evaluate the performance of these algorithms on a variety of benchmark problems. However, very few theoretical results have been obtained during this time. This relative lack of theoretical findings makes it difficult to fully assess the strengths and weaknesses of the individual algorithms. In this chapter we provide a review of theoretical advances in evolutionary dynamic optimization. In particular, we argue the importance of theoretical results, highlight the challenges faced by theoreticians and summarise the work that has been done to date. We subsequently identify relevant directions for future research.

9.1 Introduction

The field of *evolutionary dynamic optimization* is concerned with the study and application of evolutionary algorithms (EAs) to the class of dynamic optimization problems (DOPs): the dependency on time of such problems poses many new challenges to the design of EAs as pointed out by numerous monographs published in early 2000 [5, 35, 56]. This raised noticeable interest in evolutionary dynamic

Philipp Rohlfshagen · Xin Yao
Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), School of Computer Science, University of Birmingham, Birmingham B15 2TT, U.K.
e-mail: philipp.r@gmail.com, X.Yao@cs.bham.ac.uk

Per Kristian Lehre
School of Computer Science, University of Nottingham, Nottingham NG8 1BB, U.K.
e-mail: perkristian.lehre@nottingham.ac.uk

optimization and a significant number of nature-inspired techniques have subsequently been proposed to address the potential shortcomings of traditional EAs in the dynamic domain. The majority of techniques, many of which are reviewed throughout this book, employ additional features, such as the preservation of population diversity, in order to efficiently *track* high quality solutions over time.

The wealth of techniques developed is well documented by the numerous reviews that have been published in the last decade, particularly in 1999/2001 [3, 4], 2005 [26] and 2011 [8]. These reviews not only highlight the significant developments in terms of new algorithms but also the simultaneous lack of theoretical results and although the degree of theoretical results in evolutionary computation is generally overshadowed by the sheer quantity of empirical results, this discrepancy is even more apparent in the case of DOPs: the only review to mention theoretical results is by Jin and Branke [26], limited to a total of four references.

This relative lack of theoretical results, caused primarily by the added difficulty of having to account for the problem's dynamics, makes it difficult to fully assess the strengths and weaknesses of the individual algorithms. Furthermore, the lack of a clearly defined framework has made it difficult for practitioners to fully express their assumptions and to generalise from specific test scenarios to a wider class of problem dynamics. In order to draw attention to this issues, this chapter provides a self-contained overview of theoretical advances in evolutionary dynamic optimization: we argue the importance of theoretical results, highlight the challenges faced by theoreticians and summarise the work that has been done to date. Finally, we subsequently identify relevant directions for future research.

The remainder of this chapter is structured as follows: in section 9.2, we provide a brief overview of evolutionary algorithms and optimization, particularly in uncertain environments. In section 9.3 we then lay the foundation for the review of theoretical results, including an introduction to runtime analysis in the dynamic domain. The review of previous work is found in section 9.4 and finally, the chapter is concluded in section 9.5 where we summarise the reviewed work, assess its implications and outline some prospects for future work.

9.2 Evolutionary Dynamic Optimization

9.2.1 Optimization Problems

An optimization problem $f : X \to Y$ is a mapping, also known as the *objective function*, from a *search space* X to the domain Y (e.g., $\mathbb{R}$); the value $f(x) \in Y$, $x \in X$ indicates the quality of x and the elements x_i are usually referred to as *design* or *decision variables*. The dimensionality of the problem is $|x| = n$ and the set of all f-values, corresponding to all elements in X is denoted as $\hat{f}$. The goal of an optimization algorithm is usually to find the *global optimum* $x^\star \in X$ such that $f(x^\star) \geq f(x)$, $\forall x \in X$ in as little time as possible. An obstacle faced by the algorithm in doing so are *local optima*, defined as points $x \in X$ such that $f(x) \geq f(z)$, $\forall z \in N(x)$ where $N(x)$ is the *neighbourhood* of x, determined by the algorithm's variation operators. It should be noted that, without loss of generality, we assume that functions are to be maximised. Furthermore, we only consider *combinatorial* optimization problems (i.e., those with a discrete search space).

9.2.2 Optimization in Uncertain Environments

Traditionally, the majority of work in evolutionary computation has concentrated on deterministic stationary optimization problems; nevertheless, a significant effort has also been devoted to problems characterised by uncertainty. In [26], Jin and Branke review four distinct types of uncertain environments as outlined next.

Noisy optimization problems are characterised by an objective function that is subject to noise. This implies that every time a point $x \in X$ is evaluated, the value $f(x)$ varies according to some additive noise z that follows some (usually normal) distribution:

$$f(x) := \int_{-\infty}^{\infty} [f(x)+z]\; p(z)\; dz = f(x),\; z \sim N(0,\sigma^2) \tag{9.1}$$

Algorithms should subsequently work on the *expected* f-values of the search points.

In *robust* optimization, it is the decision variables $x_i,\; i = 1,\dots,n$ that are subject to minor perturbations λ *after* the value $f(x)$ has been determined (e.g., manufacturing variances):

$$f(x) := \int_{-\infty}^{\infty} [f(x+\lambda)]\; p(\lambda)\; d\lambda \tag{9.2}$$

Desired solutions are those whose f-values vary within acceptable margins given minor alterations to the solution's decision variables.

The third class of problems considered by Jin and Branke [26] is *approximate optimization* (also known as *surrogate-assisted* optimization) where the objective function is too expensive to be queried continuously. A (meta-) model, which produces approximate f-values with error $e(x)$, is used instead and the algorithm only calls the original objective function intermittently:

$$f(x) := \begin{cases} f(x) & \text{if original objective function is used;} \\ f(x)+e(x) & \text{if meta-model is used.} \end{cases} \tag{9.3}$$

Here it is vital for the algorithm to determine a reasonable trade-off between the accuracy of f-values obtained by the meta-model and the computational cost required to do so.

Finally, the fourth type of uncertain optimization problems corresponds to *dynamic optimization problems* which are *deterministic* at any moment in time but may change over time. The class of DOPs is difficult to define as, in principle, any component of f may change over time and the Handbook of Approximation Algorithms and Metaheuristics [30] states that a general definition of DOPs does not exist. Jin and Branke [26] deliberately keep the problem definition as general as possible:

$$f(x) := f(x,t) \tag{9.4}$$

The dynamics of the problem correspond to the mapping $\mathcal{T} : \mathcal{F} \times \mathbb{N} \to \mathcal{F}$ such that $f(T+1) = \mathcal{T}(f(T))$. We assume that time t advances with every call to the objective function such that $T\tau \leq t < (T+1)\tau$ where $\tau \geq 1$ is the frequency of change. It follows that T is a period index for each problem instance encountered.

The majority of practitioners attempt to design algorithms that are able to *track* high quality solution as closely as possible over time. In particular, a single solution is considered insufficient and instead, the algorithm should return a *trajectory* of solutions over time. One of the main motivations driving the development of new algorithms is the *transfer of knowledge* from one problem instance encountered to the next [5]: practitioners commonly assume that successive problem instances encountered by the algorithms are correlated to one another, allowing the algorithm to outperform a random restart by making use of the search points found so far.

9.2.3 Evolutionary Algorithms

The field of evolutionary computation provides a variety of nature-inspire metaheuristics that have been utilised successfully to obtain high quality solutions to a variety of NP-hard optimization problems. In this chapter we concentrate exclusively on those algorithms that are understood to be evolutionary algorithms (EAs): EAs are population-based global search algorithms inspired loosely by the general principles of evolutionary systems. Roughly speaking, EAs attempt to obtain solutions of increasing quality by means of *selection*, *crossover* and *mutation*: selection favours those individuals in the algorithm's population (the multiset P) that represent solutions of higher quality (exploitation) whereas crossover and mutation, the algorithm's *variation operators*, generate offspring from those individuals to advance the search (exploration).

Two simple algorithms that have been analysed theoretically in the dynamic domain are the $(1+\lambda)-$EA and the $(1+1)-$EA. The former maintains at any moment in time a single parent that produces λ offspring by means of mutation. The next generation is chosen from the set of all individuals (i.e., the offspring *and* the parent). The pseudo-code for this algorithm is shown in Algorithm 1. A special (and simpler) variant of this algorithm is the (1+1) EA where the offspring population is limited to a single individual, akin of stochastic local search. Both algorithms rely solely on their mutation operator which alters elements in x with some probability p_m.

9.3 Theoretical Foundation

9.3.1 Introduction to Runtime Analysis

The theoretical foundations for EAs are less well developed than for classical algorithms which are often accompanied with rigorously proven guarantees on the quality of their solutions and bounds on the worst-case cost of obtaining them. In contrast, EAs have traditionally been evaluated empirically on selected problem

Algorithm 1 $(1+\lambda)-$EA.

```
set $t = 1$
initialise $x(0)$ uniformly at random

while terminate = false do
    for $i := 1$ to $\lambda$ do
        $x^i(t) := x(t)$
        Alter each position of $x^i(t)$ with probability $p_m$.
    end for
    $x^{best}(t) := x^i(t) \mid f(x^i(t)) \geq f(x^j(t)),\ j = 1,\ldots,\lambda$
    if $f(x^{best}(t)) \geq f(x(t))$ then
        $x(t+1) := x^{best}(t)$
    end if
    $t := t+1$
end while
```

instances, and strong guarantees about their performance are often not available. This lack of formal performance guarantees is partly because EAs are hard to analyse. In particular, they are designed to simulate some aspect of nature without regard as to whether they can be studied formally or not. In contrast, classical algorithms are often designed specifically with runtime bounds in mind.

Nevertheless, significant progress has been made during the last decade in the runtime analysis of EAs [1, 41, 42]. The *black-box* scenario is the common theoretical framework in most of these studies [17]: the algorithm is assumed to be oblivious to the structure of the function that is to be optimised (i.e., auxiliary information like gradients is not available). Information about the function can only be gained by querying for f-value of search points the algorithm chooses. However, it is assumed that the algorithm knows the class of functions $\mathcal{F}$ (problem) from which the function (problem instance) is taken. As a consequence of the No Free Lunch theorems [16, 61], it is necessary to assume that the function class $\mathcal{F}$ has some structure which can be exploited by the algorithm as otherwise it is impossible to distinguish the (average case) performance of different algorithms (see section 9.3.3).

The expected runtime of an algorithm A on a function f is the expected number of times the algorithm evaluates the objective function before the global optimum is found for the first time (the algorithm's *hitting time*). The expectation is with respect to the random choices made by the algorithm and the expected runtime on $\mathcal{F}$ is the maximum of the expected runtimes over all $f \in \mathcal{F}$. In addition to allow a precise definition of the runtime of particular algorithms, it is also possible to define the complexity of function classes, the so-called black-box complexity [17, 32]. This is the minimum expected runtime on the problem class among all black-box algorithms.

Initial runtime studies were concerned with simple EAs like the (1+1) EA on artificial pseudo-boolean functions [13, 15, 55], highlighting how different components of evolutionary algorithms impact their runtime (e.g., the crossover operator [25, 51], population size [21, 59], diversity mechanisms [19], and selection pressure [31]). This effort also considered other types of meta-heuristics (e.g., ant colony optimization [38, 40], particle swarm optimization [52, 60], and estimation of distribution algorithms [7]) as well as new problem settings (e.g., multi-objective [20, 21, 37] and continuous [23] optimization). A significant amount of work has been directed towards studying classical combinatorial optimization problems, like maximum matching [22], sorting [47], Eulerian cycles [9], minimum spanning trees [39], and more generally matroid optimization problems [44]. For NP-hard problems, the focus has been on interesting sub-classes such as for the vertex cover problem [43], on the algorithm's approximation quality [18, 58], and on fixed-parameter tractability [28, 29]. There has also been some work trying to estimate the runtime on problems close to industrial applications, in particular in software engineering [33, 34].

9.3.2 Runtime Analysis for Dynamic Functions

The rigorous analysis of an algorithm's runtime can be very challenging and it has been common practice in the past to consider relatively simple algorithms (e.g., (1+1) EA) and problems (e.g., ONEMAX; see section 9.3.4). However, in recent years, significant progress has been made and the use of new techniques (e.g., drift analysis) allowed theoreticians to obtains proofs for significantly more complex scenarios. However, runtime analysis in the dynamic domain is further complicated by the dynamics of the function which have to be taken into account in addition to the dynamics of the algorithm. The fundamental impact of this added complexity is illustrated by the need to define a new *notion of optimality* for DOPs as traditional measures are often no longer applicable. In particular, the majority of practitioners is interested in multiple solutions across the life cycle of the problem and hence one has to evaluate and quantify the notion by which the quality of an algorithm is to be judged.

In the stationary case, the goal of an optimization algorithm is usually to find the global optimum in as few steps (number of calls to the objective functions) as possible for any number of inputs n. In his work, Droste [10, 11] translates this notion of optimality directly to the dynamic case: the author considers the expected first hitting time of the (1+1) EA in the continuously changing ONEMAX problem. The expected first hitting time in this case is more accurately referred to as *expected temporal first hitting time* as we are interested in the time the *current* global optimum is first found (which, in turn, may be lost as soon as a change occurs). However, the goal in the dynamic case is usually understood to be the *tracking* of the global optimum over time [5]. In other words, the algorithm has to *repeatedly* locate the global optimum, prompting Droste to mention additional measures that may be taken into account such as the degree to which a found optimum is lost or the average distance to the nearest optimum over time [11, p 56]. The notion of distance to the optimum was also considered by Jansen and Schellbach [24] who quantified this concept via the time until the distance between the sequence obtained by the algorithm and the target point is larger than $kd_{\max}$, assuming that the initial distance is less than $d_{\max}$. Informally, this is the time until the algorithm has "lost" the target point.

In [46], the authors take the concept of dynamic runtime a step further and consider the time required by the algorithm to relocate the global optimum once it has been lost due to update of the function; this measure is called the *second hitting time*, a specific case of the more general *expected i^{th} hitting time*. Considering multiple hitting times, it is natural to take into account the duration with which the algorithm resides at the optimum, a measure called the *séjour time*. These concepts may be formalised as follows.

Definition 9.1 (Dynamic Runtime [46]). Given a search space X and a dynamic fitness function $f : X \times \mathbb{N}_0 \to \mathbb{R}$, let $x(t)$, $t \geq 0$, be the current search point at iteration t of optimization algorithm A on dynamic fitness function f. Then the *hitting times* T_j and the *séjour times* S_i of algorithm A on function f are defined as

$$T_i := \min_t \{t \geq 0 \mid \forall y \in X, f(x(Q_i+t), Q_i+t) \geq f(y, Q_i+t)\},$$
$$S_i := \min_t \{t \geq 0 \mid \exists y \in X, f(x(Z_i+t), Z_i+t) < f(y, Z_i+t)\},$$

where $i \geq 1$ and

$$Q_k := \begin{cases} 0 & \text{if } k = 1, \\ \sum_{l=1}^{k-1} (T_l + S_l) & \text{otherwise} \end{cases}$$
$$Z_k := \begin{cases} T_1 & \text{if } k = 1, \\ T_1 + \sum_{l=1}^{k-1} (S_l + T_{l+1}) & \text{otherwise} \end{cases}$$

This definition of dynamic runtime accounts for the common notion of optimality in the dynamic domain which is concerned with the algorithm's ability to *locate* and *track* the global optimum over time. To be considered efficient in the dynamic domain, it is necessary that the algorithm locates the optimum within reasonable (i.e., polynomial) time and that the second and subsequent hitting times should not be larger than the first: the first hitting time usually assumes a uniformly random distribution from which the initial search points are selected. In the dynamic case, however, once a change takes place, the algorithm has already spent τ steps optimising the function and is thus at a non-random point (e.g., the previous global optimum) when faced with a new instance of the problem. If this is not the case, a restart strategy should be favoured over continuous tracking of the optimum.

9.3.3 No Free Lunches in the Dynamic Domain

Informally, the No Free Lunch theorem for optimization (NFL; [61]) states that any two black-box algorithms a_1 and a_2 perform, on average, identically across the set of all possible functions $\mathcal{F} = Y^X$. Similarly, in the case of time-variant functions, the average performance of any two algorithms is identical across the set of all possible dynamics $\mathcal{T} : \mathcal{F} \times \mathbb{N} \to \mathcal{F}$. The following summarises these results.

Assuming optimization algorithm a only maps to points *not* previously visited, the algorithm corresponds to the mapping $a : d \in D \to \{x \mid x \notin d^x\}$ where

D is the set of all possible samples and d^x the set of unique points sampled so far. Over m iterations, the time-ordered distinct visited points correspond to $d_m \equiv \{(d^x_m(1), d^y_m(1)), \ldots, (d^x_m(m), d^y_m(m))\}$ and it is important to note that this sample contains *all* points sampled by the algorithm, not just those accepted. The performance of an algorithm a iterated m times on function f corresponds to the likelihood that a particular sample d^y_m has been obtained: $P(d^y_m \mid f, m, a)$. The NFL theorem then states that

$$\sum_f P(d^y_m \mid f, m, a_1) = \sum_f P(d^y_m \mid f, m, a_2) \tag{9.5}$$

It follows that for any performance measure $\Phi(d^y_m)$ based on the samples d^y_m, the average performance over all functions is independent of a. Wolpert and McReady [61] extend this analysis to the class of time-variant functions, highlighting two particularly interesting issues:

1. How the dynamic functions are defined.
2. How the performance of an algorithm is measured.

The authors consider the case where the algorithm starts with function f_1 and with each subsequent iteration of the algorithm, the function is transformed to a new function by the bijective mapping $\mathcal{T} : \mathcal{F} \times \mathbb{N} \to \mathcal{F}$. The authors note that an algorithm's performance in the dynamic domain is not trivially defined and propose two measures: in the first scheme, the y-value corresponding to a particular point x is determined by the function at the time the point was evaluated. In the second case, the y-values correspond to the values obtained for each x sampled according to the *final* function encountered.

Subsequently, a similar result to equation 9.5 may then be obtained. In this case, the average is over all possible dynamics $\mathcal{T}$ (rather than functions f):

$$\sum_T P(d^y_m \mid f_1, T, m, a_1) = \sum_T P(d^y_m \mid f_1, T, m, a_2) \tag{9.6}$$

9.3.4 Benchmark Problems

Numerous dynamic benchmark problems have been proposed in the past, allowing practitioners to test, evaluate and compare their algorithms. These benchmarks include tools to generate a wide range of dynamics (e.g., MOVING PEAKS[5] and DF1[35]) and dynamic variants of well-known NP-hard stationary optimization problems (e.g., Travelling Salesman Problem or Scheduling Problems). Naturally, theoreticians have concentrated on simpler problems with well-defined dynamics. The XOR DOP [62, 63] benchmark may be used to impose artificial dynamics on any pseudo-Boolean optimization problem. It is a generalisation of the dynamic pattern match problem that was used in the first attempt to analyse the runtime of an EA in the dynamic domain. Finally, a third problem that has been considered by theoreticians is a simple tracking problem in a lattice. These problems are reviewed below.

9.3.4.1 Dynamic Match Function

Standhope and Daida [49, 50] propose a simple dynamic function for an initial analysis of the behaviour of the (1+1) EA. The function is a generalisation of the well-known ONEMAX problem to the dynamic domain. The f-values in the ONEMAX problem simply correspond to the number of ones found in the solution x:

$$\text{ONEMAX}(x) = \sum_{i=1}^{n} x_i \tag{9.7}$$

In the case of the dynamic match function, the algorithm must reduce the Hamming distance $d(x,z) = \sum_{i=1}^{n} |x_i - z_i|$ to an arbitrary target pattern (match string) σ_τ that may change over time:

$$f(a,\sigma_\tau) = \sum_{i=1}^{n} \neg(a_i \oplus \sigma_{\tau i}) \tag{9.8}$$

where $\neg$ is the logical not operator and $\oplus$ logical xor. The dynamics of σ_τ are controlled by two parameters, g and d which control the number of generations between changes and the degree (Hamming distance) by which the target pattern is altered (d distinct and randomly chosen bit in σ are inverted). The values $(0,0)$ result in a stationary function whereas the values $(2,5)$ would imply that every 2 generations, the target changes by 5 bits.

9.3.4.2 The XOR DOP Problem

XOR DOP [62, 63] is the only widely accepted benchmark problem in dynamic optimization for the combinatorial domain and generates a dynamic version of any static pseudo-Boolean problem. It is a generalisation of the dynamic match function and imposes dynamics on *any* stationary pseudo-boolean function $f : \mathbb{B} \to \mathbb{R}$ by means of a bit-wise *exclusive-or* operation that is applied to each search point $x \in \{0,1\}^n$ prior to each function evaluation. The dynamic equivalent of any stationary function is simply

$$f(x(t) \ \oplus \ m(T)) \tag{9.9}$$

where $\oplus$ is the *xor* operator. The vector $m(T) \in \{0,1\}^n$, which initially is equivalent to 0^n, is a binary mask, generated by $m(T) = m(T-1) \oplus p(T)$ where $p(T) \in \{0,1\}^n$ is a randomly created template that contains exactly $\lfloor \rho n \rfloor$ ones. The value of $\rho \in [0,1]$ thus controls the magnitude of change which is specified as the Hamming distance between two binary points. It follows that ρn is the actual number of bits inverted. The period index $T = \lceil t/\tau \rceil$ is determined by the duration $\tau > 0$ between changes.

XOR DOP was analysed by Tinós and Yang [53]: if we assume that the transformation of each encoding $x(t)$ by $m(T)$ yields a vector $z(t) = x(t) \oplus m(T)$, then it is possible to rewrite this expression as $z^n(t) = A(T)x^n(t)$ where $x \in \{0,1\}^n$ is normalised to $x^n(t) \in \{-1,1\}^n$ and where $A(T)$ is a linear transformation:

$$A(\pi) = \begin{bmatrix} A_1(\pi) & 0 & \dots & 0 \\ 0 & A_2(\pi) & \dots & 0 \\ & & \ddots & \\ 0 & 0 & \dots & A_n(T) \end{bmatrix}$$

where

$$A_i(\pi) = \begin{cases} 1 & \text{if } m_i(T) = 0 \\ -1 & \text{if } m_i(T) = 1 \end{cases}$$

for $i = 1, 2, \dots, n$. It follows that XOR DOP does not alter the underlying function but instead *rotates* each search point x prior to each function evaluation.

This analysis was extended in [54] using a dynamical system analysis. In particular, the authors showed that XOR DOP corresponds to a DOP with permutations: the class of DOPs with permutation are those with dynamics that permute the assignment between elements in X and those in $\hat{f}$. The authors subsequently showed that the xor operator may be replaced with a single mutational step performed whenever the function is meant to change. In other words, given two dynamic processes, one governed by XOR DOP, the other by an initial mutational step, "if both evolutionary processes have the same initial population and parameters, and the fitness function in the first change cycle for the first process is equal to the fitness function in the second process, then the evolution of the population in the two processes is identical, i.e., the two evolutionary processes are equivalent.".

9.3.4.3 Tracking Problem

Weicker [57] considered a simple *tracking problem* modelled on the integer lattice as a sequence of target points $a_1, a_2, \dots \in \mathbb{Z}^2$ together with a time-variant objective function $f : \mathbb{Z}^2 \times \mathbb{N} \to \mathbb{R}$ which is to be minimised. The first argument to the objective function is a point in the lattice, and the second argument is the time parameter. The function is defined for all x and t as $f(x,t) := \|x - a_t\|$, where $\|\cdot\|$ is the ℓ_1-norm. Essentially, $f(x,t)$ is the Manhattan-distance between the point x and the current target point at time t.

The sequence $a_1, a_2, \dots$, representing a moving target, is unknown to the algorithm, and can be deterministic, stochastic, or chosen by an adversary. The only assumption made about the sequence is that for some parameter $d_{\max}$, $\|a_t - a_{t+1}\| \leq d_{\max}$ holds for all $t \geq 0$ (i.e., the speed of the target point is no more than $d_{\max}$). The special case $d_{\max} = 0$ corresponds a static optimization problem.

Informally, the objective of an algorithm in the tracking problem is to obtain a sequence of search points $x_1, x_2, \dots \in \mathbb{Z}^2$ that are close to the sequence of target points. Various aspects of this informal objective have been formalised. First, the algorithm needs to obtain a search point that is within acceptable distance to the target point, then it must track the moving target. Jansen and Schellbach [24] considered the *first hitting-time* $T_{\max_d}(n)$, defined as the number of function evaluations until a search point x_t has been obtained for which $\|x_t - a_t\| \leq d_{\max}$, assuming that the algorithm is provided with an initial search point x_1 for which $\|x_1 - a_1\| = n$.

9.4 Runtime Analysis for Dynamic Functions

9.4.1 First Hitting Times for Pattern Match

The (1+1) EA has been analysed on several variants of the dynamic ONEMAX (match function), extending the work carried out previously on its stationary counterpart (e.g., [2, 14, 36]). The dynamic variants of the ONEMAX differ in their transitions from one instance to the next, a property which may drastically affect the runtime of the algorithm.[1]

The first consideration of a non-empirical analysis is due to Stanhope and Daida [50] who consider the (1+1) EA with simplified mutation operator on the dynamic pattern match function with intergenerational updates. The mutation operator considered inverts exactly r bit, chosen uniformly at random; the mutation rate r remains constant throughout the algorithm execution. The authors first consider the case $(0,0)$ (i.e., a stationary function) and then generalise to the dynamic case (d,g). The authors use a hypergeometric random variable to describe the probability distribution over neighbouring search points that may be generated by the mutation operator. The dynamics of the pattern matching function are modelled as an additional mutational step that inverts d bits every g generations (c.f., analysis of XOR DOP in section 9.3.4). The authors subsequently derive a distribution function on the fitness if a selected individual. The transition probabilities are validated empirically by a comparison to Monte-Carlo generated fitness distributions and amongst other things, Stanhope and Daida showed that even small perturbations in the fitness function could have a significantly negative impact on the performance of the (1+1) EA.

In a sequence of papers [11, 12], Droste considered the dynamic ONEMAX problem. The initial model considered the target sequence modified by a single, uniformly chosen, bit-flip with probability p in each iteration. The goal of the study was to determine values of p for which the (1+1) EA has polynomial expected first hitting time. Since the parameter setting $p = 0$ corresponds to the static ONEMAX-problem, for which the (1+1) EA has expected first hitting time $O(n \log n)$ [15], it is clear that exponential first hitting time can only occur for strictly larger values. It should be noted that the (1+1) EA considered by Droste [10] was adapted specifically to the dynamic domain by calling the objective function twice during each iteration to prevent the use of outdated f-values.

Droste found that the first hitting time remains polynomial as long as $p = O(\log n/n)$. At this rate, the target sequence is modified $O(\log^2(n))$ times in expectation during a time interval of $n \log n$ iterations. This rate turned out to be critical, as the expected first hitting time becomes exponential for $p = \omega(\log(n)/n)$.

The dynamic ONEMAX-model considered by Droste can be generalised. Instead of only flipping one bit with a certain probability, one can define a random operator $\mathcal{M}$ that acts on the target sequence in each iteration. One such natural operator, is

[1] As pointed out in [49], the pattern match function is equivalent to the application of XOR DOP to the ONEMAX function. Furthermore, the runtime analysis may be simplified if the dynamics are viewed as an additional mutation operator that acts directly on x (depending of magnitude and frequency of change; see section 9.3.4.2).

to flip each bit position in the target sequence with some probability p' in each iteration. While the second model leads to a more involved analysis, the results are essentially the same in the two models. Note first that by setting $p' := p/n$, the expected number of bit-flips to the target sequence per time step is the same in the both models. Droste found that the expected first hitting time remains polynomial as long as $p' = O(\log(n)/n^2)$, whereas the expected first hitting time becomes exponential as soon as $p'' = \omega(\log(n)/n^2)$.

9.4.2 Analysis of Frequency and Magnitude of Change

In [46], the authors look at the two most prominent attributes of most DOPs, the *magnitude of change* and the *frequency of change*. The former is generally regarded as the *relatedness* of two successive problem instances, $f(T)$ and $f(T+1)$ and a common assumption is that smaller magnitudes of change are easier to adapt to, primarily by "transferring knowledge from the past" [26, p 311]. The authors attempt to shed light on the question whether this is always the case or whether examples exist where a large magnitude of change may make it easier for the algorithm to relocate the global optimum.

For the magnitude of change, a specially designed function called MAGNITUDE is proposed: informally, this bi-modal function features a local optimum (LOCAL) surrounded by a valley of low f-values (TRAP). Beyond the valley is a region (ZERO) that leads a path that leads to the global optimum (GLOBAL). This stationary function is subsequently made dynamic using the XOR DOP framework to yield a dynamic MAGNITUDE function. The authors subsequently found that for (1+1) EA on MAGNITUDE with an update time $\tau \geq n^2 \log n$, and a magnitude of change θ, the second hitting T_2 satisfies

1. For small magnitudes of change, i.e. when $1 \leq \theta \leq q - cn$,
$$\mathbf{E}[T_2] = e^{\Omega(n)}$$
2. For large magnitudes of change, i.e. when $3q \leq \theta \leq n$,
$$\mathbf{Pr}\left[T_2 \leq n^2 \log n\right] = 1 - e^{-\Omega(n)}.$$

The proof idea for the runtime of the (1+1) EA follows directly from the function definition and is based on two concepts: the behaviour of the algorithm during each update period (i.e., in time of stagnation) and the impact of the dynamics on the algorithm, given the algorithm is either at LOCAL or GLOBAL. It is assumed that the time between changes is sufficiently long for the algorithm to reach one of the two optima with high probability; depending on the magnitude of change, different behaviours emerge. The initial search point may be in TRAP or ZERO. The probability to be on PATH is exceedingly small. If the algorithm starts in TRAP, it will be led away from the other regions of the search space towards the point LOCAL. Subsequently, if a small change occurs, the rotated search point will still be in the region TRAP and is hence attracted again to the local optimum. If, on the other hand, the initial search point is not in Trap, the algorithm is led to the beginning of the path which leads directly to Global. The situation at Global is similar to the one at Local. If the magnitude of change is small, the search point will be rotated into the Trap region. If the magnitude of change is large, on the other hand, the search point will *jump* across the trap into Zero or Path. Similarly, if the algorithm is at Local and a large change takes place, the search point is rotated beyond the boundary of the Trap region.

The authors also looked at the frequency of change and similarly to the result above, showed that a high frequency of change may allow the algorithm to locate the function's global optimum whereas a low frequency of change does not. More specifically, it is shown that the dynamic optimization problem called Balance is hard for the (1+1) EA at low frequencies, and easy at high frequencies. Informally, the function is defined as follows: the algorithm is drawn towards the global optimum along the z-axis while the dynamics "tilt" the plane along the x-axis, elevating the f-values at different parts across the y-axis. This potentially draws the algorithm towards a trap-region that allows the algorithm to only get within a specific distance to the global optimum. If the algorithm is not trapped, on the other hand, the global optimum may be found by incremental improvements to the search points sampled.

The function is also made dynamic using the XOR DOP framework but a specially designed mask m is used to alter the search points in specific ways:

$$m(T) := \begin{cases} 0^{n/2} \cdot 0^{n/2} & \text{if } T \bmod 2 = 0, \text{ and} \\ 0^{n/2} \cdot 1^{n/2} & \text{otherwise.} \end{cases}$$

Hence, only the suffix of the point x is affected, and the magnitude of change is equivalent to $n/2$.

Theorem 9.1 ([46]). *The expected first hitting time of (1+1) EA on* Balance *with update time τ is*

$$\mathbf{E}[T] = \begin{cases} n^{\Omega(\sqrt{n})} & \text{if } \tau > 40n, \text{ and} \\ O(n^2) & \text{if } \tau = 2. \end{cases}$$

The idea for the proof shows that the algorithm will balance along the centre of the vertical axis when the frequency of change is high, while the algorithm is likely to fall into one of the trap regions when the frequency of change is sufficiently low. This can be proved by analysing the horizontal and vertical *drift*. Informally, the drift of a search point is the distance the search point moves per iteration. The horizontal drift corresponds to the change in number of leading 1-bits in the prefix, and the vertical drift corresponds to the change in number of 1-bits in the suffix. As long as the trap region has not been reached, the position along the vertical axis can be changed by flipping any of at least $n/16$ bits, and no other bits. In contrast, in order to reduce the distance to the optimum along the horizontal axis, it is necessary to flip the single left-most 0-bit, an event that happens with much lower probability. Therefore, the vertical drift is much larger than the horizontal drift. If the frequency of change is sufficiently low, then the current search point will have enough time to reach one of the trap regions before the optimum is found. On the other hand, if the frequency of change is sufficiently high, then the search point will not have time to reach the trap region during one period. In the following period, the vertical drift will be in the opposite direction, and the vertical displacement of the search point is off-set. These informal ideas can be turned into a rigorous analysis using the simplified drift theorem.

9.4.3 Tracking the Optimum in a LATTICE

Jansen and Schellbach [24] analysed the performance of the $(1+\lambda)$ EA on the tracking problem described in section 9.3.4.3. An offspring is generated by adding to the parent K vectors sampled uniformly at random with replacement from the set $\{(\pm 1, 0), (0, \pm 1)\}$, where K is a Poisson distributed random variable with parameter 1. Hence, the offspring has expected distance 1 from the parent. The analysis assumes that the time steps of the objective function is synchronised with the generation counter of the EA. Hence, in each generation $t \geq 1$, the algorithm evaluates the distance between λ offspring and the current target point a_t. Intuitively, a larger population size should be beneficial for the EA within this scenario, because each generation provides more information about the position of the current target point.

For the special case of $d_{\max} = 0$, (i.e., a static optimization problem), they obtained the following asymptotically tight bound on the first hitting time.

Theorem 9.2 ([24])

$$\mathbf{E}\left[T_{\lambda,0}(n)\right] = \Theta\left(\lambda \cdot \left(1 + \frac{n \cdot \log\log\lambda}{\log\lambda}\right)\right).$$

As the algorithm makes λ function evaluations per generation, and needs to overcome a distance of n, the result informally means that the speed of the algorithm is on the order of $\Theta(\log\lambda / \log\log\lambda)$ per generation. Increasing the population size λ decreases the expected number of generations needed to reach the (static) target point.

The potential difficulty of the tracking problem increases with the parameter $d_{\max}$. In the worst case, the target point moves in the opposite direction of the current search point. Intuitively, if $d_{\max}$ is significantly lower than the speed $\Theta(\log\lambda / \log\log\lambda)$ of the algorithm, then one would expect the algorithm to be able to reach the target point. The following theorem confirms this intuition.

Theorem 9.3 ([24]). *Let* $b := 4/e$, $n' := n - d_{max}$, $\tilde{c} > 1$, *and* $s := \left\lfloor \frac{\log_b \lambda}{2\tilde{c}\log_b \log_b \lambda} \right\rfloor$. *For* $d_{max} \leq (2/3 - o(1/\lambda))s$, *it holds*

$$\mathbf{Pr}\left[T_{\lambda,dmax}(n) = O\left(\lambda(1 + \frac{n'\log\log\lambda}{\log\lambda})\right)\right] = 1 - 2^{-\Omega(n'/s)}$$

Once the algorithm is within distance $d_{\max}$ of the moving target point, it is intuitive that the algorithm does not loose track of the target point. The following theorem shows that the expected time until the target point is lost grows exponentially with the population size.

Theorem 9.4 ([24]). *Let s be as in Theorem 9.3. For $d_{max} \leq (2/3)s$, and any integer $k \geq 2$, the expected number of generations until the $(1+\lambda)$ EA has a distance to the target of at least kd_{max} after having a distance of at most d_{max} is bounded below by $e^{\Omega(k\sqrt{\lambda})}$.*

In contrast, when $d_{\max}$ is significantly higher than $\Theta(\log\lambda/\log\log\lambda)$, and the target point moves in an adverserial way, it is to be expected that the tracking problem becomes hard for the (1+λ) EA. Jansen and Schellbach [24] provide some theoretically motivated arguments that support this view.

9.5 Conclusions

9.5.1 Summary and Implications

The number of contributions in evolutionary dynamic optimization has risen dramatically in recent years and a wealth of novel evolutionary algorithms (EAs) have been suggested that attempt to *track* the global optimum of some dynamic function over time. However, theoretical results on the expected runtimes of these algorithms are almost non-existent and almost all findings are based exclusively on empirical data. This imbalance may make it difficult to validate an algorithm's performance and to identify a broader class of functions the algorithm may work well on. Furthermore, the lack of theoretical results may lead to incorrect empirical validation of common assumptions about the dynamic domain. In this chapter, we reviewed previous theoretical results in an attempt to highlight some of the gaps in our understanding of evolutionary dynamic optimization. These results may be summarised as follows.

The results by Droste [11, 12] showed how the rate of change plays a crucial role in the algorithm's (temporal) first hitting time. In a more general sense, this is a first step in understanding how subtle differences in the problem's dynamics make the problem tractable or not. This result extended the earlier study by Stanhope and Daida [50] that even small perturbations in the fitness function could have a significantly negative impact on the performance of the (1+1) EA. The results by Rohlfshagen et al. [46] have shown that it is possible to show examples where common assumptions (i.e., that a larger magnitude of change / higher frequency of change makes a DOP harder) break down. This has important ramifications regarding the treatment of such problems and how one generalise empirical results from specific test cases to more general classes of DOPs. Similarly, Chen et al. [6] proved that adaptive and self-adaptive mutations may not perform as well as one might have thought in a dynamic environment. A fixed and non-adaptive scheme can sometime be just as good as any adaptive schemes in a dynamic environment. Finally, Jansen and Schellbach [24] is the only work to consider an offspring population. Within the framework considered, the authors showed that increasing the algorithm's population size decreases the expected number of generations needed to reach the target point and that the expected time until the target point is lost grows exponentially with the population size.

These theoretical studies have some important implications. First of all, they highlight the difficulty in defining the problem itself, including ways to unambiguously describe some of its properties such as the magnitude of change. Furthermore, the performance of an algorithm may be measured in numerous different ways and there are subtle differences between each approach; it is important to better understand these differences as they clearly have an impact on how one evaluates a particular algorithm on a particular problem.

9.5.2 Future Work

As the review in section 9.4 has shown, the scope of existing theoretical results is limited. Nevertheless, the progress to date is essential to further developments in the field and build the basis for future work. We believe the following constitutes important directions for future theoretical work in evolutionary dynamic optimization especially with regard to the significant advances made recently in the runtime analysis of EAs in the stationary domains.

1. **Framework and problem complexity.** A theoretical framework is required that allows practitioners and theoreticians to unambiguously describe different instances of DOPs. This framework would subsequently allow for the classification of different types of DOPs and may subsequently facilitate an analysis of problem complexity. In particular, currently it is not possible to identify and distinguish between those types of DOPs that are easier to solve than stationary problems and those that are harder; empirical evidence from biology seems to suggest that certain types of dynamics allows for faster rates of adaptation (e.g., [27]).
2. **Notion of optimality.** As the review has highlighted, numerous different notions of optimality/runtime may be applied in the dynamic domain and their relationship remains to be established. A general definition of runtime analysis in the dynamic domain should also be grounded in practical requirements and hence be able to account for trajectory-based performance measures as used in most practical applications.
3. **Populations.** One of the main motivations behind the application of EAs to DOPs is their use of populations. In particular, it is thought that the sampling of multiple search points simultaneously allows for better rates of adaptation in the new environments. The work by Jansen and Schellbach [24] provides an initial analysis of the role played by populations yet further examples are required where populations are provably beneficial.
4. **Diversity.** Diversity is considered one of the key issues that determines the performance of an EA on a particular DOP and the majority of algorithms developed aim to maintain high levels of diversity throughout the algorithm's execution. However, it is clear that not all types of diversity are equally useful and

hence a better understanding is required to identify mechanisms that are able to produce useful levels of diversity given a particular DOP.

5. **Crossover.** Related to the issues of populations and diversity comes a better understanding of crossover operators; so far, only EAs with mutation have been considered yet almost all population-based algorithms developed for DOPs also employ crossover operators. Nevertheless, there is little evidence that substantiates the impact of crossover on the algorithm's performance. There is evidence from biology that the evolution of sexual reproduction (i.e., crossover) is directly linked to uncertainty in the environment (e.g., [45, 48]).
6. **Beyond EAs and toy problems.** There are many additional population-based algorithms, such as ant colony optimization and particle swarm optimization, that have already been considered from a practical point of view. Furthermore, it is important to extend the theoretical treatment from simple artificial problems to simple dynamics variants of well-known NP hard problems such as the travelling salesman problem.

Acknowledgements. This work was partially supported by the Engineering and Physical Sciences Research Council (EPSRC) of UK under Grant EP/E058884/1.

References

[1] Auger, A., Doerr, B. (eds.): Theory of randomized search heuristics. World Scientific Publishing (2011)

[2] Back, T.: Optimal mutation rates in genetic search. In: Forrest, S. (ed.) Proc. 5th Int. Conf. Genetic Algorithms, pp. 2–8. Morgan Kaufmann (1993)

[3] Branke, J.: Evolutionary algorithms for dynamic optimization problems - a survey. Tech. Rep. 387, Insitute AIFB, University of Karlsruhe (1999)

[4] Branke, J.: Evolutionary approaches to dynamic environments - updated survey. In: GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems, pp. 27–30 (2001)

[5] Branke, J.: Evolutionary Optimization in Dynamic Environments. Kluwer (2002)

[6] Chen, T., Chen, Y., Tang, K., Chen, G., Yao, X.: The impact of mutation rate on the computation time of evolutionary dynamic optimization. arXiv preprint arXiv:1106.0566 (2011)

[7] Chen, T., Lehre, P.K., Tang, K., Yao, X.: When is an estimation of distribution algorithm better than an evolutionary algorithm? In: Proc. 2009 IEEE Congr. Evol. Comput., pp. 1470–1477. IEEE (2009)

[8] Cruz, C., González, J.R., Pelta, D.A.: Optimization in dynamic environments: a survey on problems, methods and measures. Soft Comput. 15(7), 1427–1448 (2011)

[9] Doerr, B., Klein, C., Storch, T.: Faster evolutionary algorithms by superior graph representation. In: Proc. 1st IEEE Symp. Foundations of Comput. Intell., pp. 245–250 (2007)

[10] Droste, S.: Analysis of the (1+1) ea for a dynamically changing objective function. Tech. rep., Universität Dortmund (2001)

[11] Droste, S.: Analysis of the (1+1) ea for a dynamically changing onemax-variant. In: Proc. 2002 IEEE Congr. Evol. Comput., pp. 55–60 (2002)

[12] Droste, S.: Analysis of the (1+1) ea for a dynamically bitwise changing onemax. In: Cantú-Paz, E., et al. (eds.) GECCO 2003. LNCS, vol. 2723, pp. 909–921. Springer, Heidelberg (2003)

[13] Droste, S., Jansen, T., Wegener, I.: On the optimization of unimodal functions with the (1 + 1) evolutionary algorithm. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 13–22. Springer, Heidelberg (1998)

[14] Droste, S., Jansen, T., Wegener, I.: A rigorous complexity analysis of the (1+1) evolutionary algorithm for linear functions with boolean inputs. Evol. Comput. 6(2), 185–196 (1998)

[15] Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) Evolutionary Algorithm. Theoretical Computer Science 276, 51–81 (2002)

[16] Droste, S., Jansen, T., Wegener, I.: Optimization with randomized search heuristics the (a)nfl theorem, realistic scenarios, and difficult functions. Theoretical Computer Science 287 (2002)

[17] Droste, S., Jansen, T., Wegener, I.: Upper and lower bounds for randomized search heuristics in black-box optimization. Electronic Colloquium on Computational Complexity (ECCC) 48, 2003 (2004)

[18] Friedrich, T., Hebbinghaus, N., Neumann, F., He, J., Witt, C.: Approximating covering problems by randomized search heuristics using multi-objective models. In: Proc. 9th Annual Conf. Genetic and Evol. Comput., pp. 797–804 (2007)

[19] Friedrich, T., Oliveto, P.S., Sudholt, D., Witt, C.: Analysis of diversity-preserving mechanisms for global exploration. Evol. Comput. 17(4), 455–476 (2009)

[20] Giel, O.: Zur analyse von randomisierten suchheuristiken und online-heuristiken. Ph.D. thesis, Universität Dortmund (2005)

[21] Giel, O., Lehre, P.K.: On the effect of populations in evolutionary multi-objective optimisation. Evol. Comput. 18(3), 335–356 (2010)

[22] Giel, O., Wegener, I.: Evolutionary algorithms and the maximum matching problem. In: Alt, H., Habib, M. (eds.) STACS 2003. LNCS, vol. 2607, pp. 415–426. Springer, Heidelberg (2003)

[23] Jägersküpper, J.: Probabilistic analysis of evolution strategies using isotropic mutations. Ph.D. thesis, Universität Dortmund (2006)

[24] Jansen, T., Schellbach, U.: Theoretical analysis of a mutation-based evolutionary algorithm for a tracking problem in lattice. In: Beyer, H.G.G. (ed.) Proc. 2005 Genetic and Evol. Comput. Conf., pp. 841–848. ACM (2005)

[25] Jansen, T., Wegener, I.: Real royal road functions–where crossover provably is essential. Discrete Applied Mathematics 149(1-3), 111–125 (2005)

[26] Jin, Y., Branke, J.: Evolutionary optimization in uncertain environment - a survey. IEEE Trans. Evol. Comput. 9(3), 303–317 (2005)

[27] Kashtan, N., Noor, E., Alon, U.: Varying environments can speed up evolution. PNAS 104(34), 13,711–13,716 (2007)

[28] Kratsch, S., Lehre, P.K., Neumann, F., Oliveto, P.S.: Fixed parameter evolutionary algorithms and maximum leaf spanning trees: A matter of mutation. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 204–213. Springer, Heidelberg (2010)

[29] Kratsch, S., Neumann, F.: Fixed-parameter evolutionary algorithms and the vertex cover problem. In: Proc. 2009 Genetic and Evol. Comput. Conf., pp. 293–300 (2009)

[30] Leguizamon, G., Blum, C., Alba, E.: Handbook of approximation algorithms and metaheuristics, pp. 24.1–24.X. CRC Press (2007)

[31] Lehre, P.K.: Negative drift in populations. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 244–253. Springer, Heidelberg (2010)

[32] Lehre, P.K., Witt, C.: Black-box search by unbiased variation. In: Proc. 12th Annual Conf. Genetic and Evol. Comput., pp. 1441–1448. ACM, New York (2010)

[33] Lehre, P.K., Yao, X.: Runtime analysis of search heuristics on software engineering problems. Frontiers of Computer Science in China 3(1), 64–72 (2009)

[34] Lehre, P.K., Yao, X.: Runtime analysis of the (1+1) EA on computing unique input output sequences. Inform. Sci. (2011)

[35] Morrison, R.W.: Designing Evolutionary Algorithms for Dynamic Environments. Springer, Berlin (2004) ISBN 3-540-21231-0

[36] Muhlenbein, H.: How genetic algorithms really work i: Mutation and hillclimbing. In: Manner, R., Manderick, B. (eds.) Proc. 2nd Int. Conf. Parallel Problem Solving from Nature, pp. 15–25 (1992)

[37] Neumann, F.: Combinatorial optimization and the analysis of randomized search heuristics. Ph.D. Thesis, Christian-Albrechts-Universität zu Kiel (2006)

[38] Neumann, F., Sudholt, D., Witt, C.: A few ants are enough: Aco with iteration-best update. In: Proc. 12th Annual Conf. Genetic and Evol. Comput., pp. 63–70. ACM (2010)

[39] Neumann, F., Wegener, I.: Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. Theoretical Computer Science 378(1), 32–40 (2007)

[40] Neumann, F., Witt, C.: Runtime analysis of a simple ant colony optimization algorithm. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 618–627. Springer, Heidelberg (2006)

[41] Neumann, F., Witt, C.: Bioinspired Computation in Combinatorial Optimization. Natural Computation Series. Springer (2010)

[42] Oliveto, P.S., He, J., Yao, X.: Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results. Int. J. of Automation and Computing 4(1), 100–106 (2007)

[43] Oliveto, P.S., He, J., Yao, X.: Analysis of population-based evolutionary algorithms for the vertex cover problem. In: Proc. 2008 IEEE World Congr. Comput. Intell., pp. 1563–1570 (2008)

[44] Reichel, J., Skutella, M.: Evolutionary algorithms and matroid optimization problems. Algorithmica 57(1), 187–206 (2010)

[45] Ridley, M.: The Red Queen: Sex and the Evolution of Human Nature. Penguin Books Ltd. (1993)

[46] Rohlfshagen, P., Lehre, P.K., Yao, X.: Dynamic evolutionary optimisation: An analysis of frequency and magnitude of change. In: Proc. 2009 Genetic and Evol. Comput. Conf., pp. 1713–1720 (2009)

[47] Scharnow, J., Tinnefeld, K., Wegener, I.: Fitness landscapes based on sorting and shortest paths problems. In: Guervós, J.J.M., Adamidis, P.A., Beyer, H.-G., Fernández-Villacañas, J.-L., Schwefel, H.-P. (eds.) PPSN 2002. LNCS, vol. 2439, pp. 54–63. Springer, Heidelberg (2002)

[48] Smith, J.M.: The Evolution of Sex. Cambridge University Press, Cambridge (1978)

[49] Stanhope, S.A., Daida, J.M.: Optimal mutation and crossover rates for a genetic algorithm operating in a dynamic environment. In: Porto, V.W., Waagen, D. (eds.) EP 1998. LNCS, vol. 1447, pp. 693–702. Springer, Heidelberg (1998)

[50] Stanhope, S.A., Daida, J.M. (1+1) genetic algorithm fitness dynamics in a changing environments. In: Proc. 1999 IEEE Congr. Evol. Comput., vol. 3, pp. 1851–1858 (1999)

[51] Storch, T., Wegener, I.: Real royal road functions for constant population size. Theoretical Computer Science 320(1), 123–134 (2004)

[52] Sudholt, D., Witt, C.: Runtime analysis of binary pso. In: GECCO 2008: Proc. 10th Annual Conf. Genetic and Evol. Comput., pp. 135–142. ACM, New York (2008)

[53] Tinos, R., Yang, S.: Continuous dynamic problem generators for evolutionary algorithms. In: Proc. 2007 IEEE Congr. Evol. Comput., pp. 236–243 (2007)

[54] Tinós, R., Yang, S.: An analysis of the XOR dynamic problem generator based on the dynamical system. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 274–283. Springer, Heidelberg (2010)

[55] Wegener, I., Witt, C.: On the analysis of a simple evolutionary algorithm on quadratic pseudo-boolean functions. Journal of Discrete Algorithms 3(1), 61–78 (2005)

[56] Weicker, K.: Evolutionary algorithms and dynamic optimization problems. Der Andere Verlag (2003)

[57] Weicker, K.: Analysis of local operators applied to discrete tracking problems. Soft Comput. 9(11), 778–792 (2005)

[58] Witt, C.: Worst-case and average-case approximations by simple randomized search heuristics. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 44–56. Springer, Heidelberg (2005)

[59] Witt, C.: Population size versus runtime of a simple evolutionary algorithm. Theoretical Computer Science 403(1), 104–120 (2008)

[60] Witt, C.: Why standard particle swarm optimisers elude a theoretical runtime analysis. In: Proc. 10th Int. Workshop Foundations of Genetic Algorithms, pp. 13–20. ACM, New York (2009)

[61] Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**(1) (1997)

[62] Yang, S.: Non-stationary problem optimization using the primal-dual genetic algorithms. In: Sarker, R., Reynolds, R., Abbass, H., Tan, K.C., McKay, R., Essam, D., Gedeon, T. (eds.) Proc. 2003 IEEE Congr. Evol. Comput., vol. 3, pp. 2246–2253 (2003)

[63] Yang, S.: Memory-based immigrants for genetic algorithms in dynamic environments. In: Proc. 2005 Genetic and Evol. Comput. Conf., pp. 1115–1122. ACM (2005)