Unpacking and Understanding Evolutionary Algorithms

Xin Yao

CERCIA, School of Computer Science
University of Birmingham
Edgbaston, Birmingham B15 2TT, UK
x.yao@cs.bham.ac.uk
http://www.cs.bham.ac.uk/~xin

Abstract. Theoretical analysis of evolutionary algorithms (EAs) has made significant progresses in the last few years. There is an increased understanding of the computational time complexity of EAs on certain combinatorial optimisation problems. Complementary to the traditional time complexity analysis that focuses exclusively on the problem, e.g., the notion of NP-hardness, computational time complexity analysis of EAs emphasizes the relationship between algorithmic features and problem characteristics. The notion of EA-hardness tries to capture the essence of when and why a problem instance class is hard for what kind of EAs. Such an emphasis is motivated by the practical needs of insight and guidance for choosing different EAs for different problems. This chapter first introduces some basic concepts in analysing EAs. Then the impact of different components of an EA will be studied in depth, including selection, mutation, crossover, parameter setting, and interactions among them. Such theoretical analyses have revealed some interesting results, which might be counter-intuitive at the first sight. Finally, some future research directions of evolutionary computation will be discussed.

1 Introduction

Evolutionary computation refers to the study of computational systems that are inspired by natural evolution. It includes four major research areas, i.e., evolutionary optimisation, evolutionary learning, evolutionary design and theoretical foundations of evolutionary computation.

1.1 Evolutionary Optimisation

Evolutionary optimisation includes a wide range of topics related optimisation, such as global (numerical) optimisation, combinatorial optimisation, constraint handling, multi-objective optimisation, dynamic optimisation, etc. Many evolutionary algorithms (EAs) have been used with success in a variety of application domains that rely on optimisation. For example, in the area of global optimisation, fast evolutionary programming (FEP) and improved FEP [1] were used successfully in modelling and designing new aluminium alloys [2]. Self-adaptive differential evolution with neighbourhood search (SaNSDE) [3] was used to calibrate building thermal models [4].

In the area of combinatorial optimisation, memetic algorithms were designed for tackling capacitated arc routing problems [5], which were inspired by the real world problem of route optimisation for gritting trucks in winter road maintenance [6]. Multi-objective EAs (MOEAs) were used very effectively for module clustering in software engineering [7] and optimal testing resource allocation in modular software systems [8]. Advantages of using the MOEAs were clearly demonstrated in these two cases.

1.2 Evolutionary Learning

Evolutionary learning appears in many different forms, from the more classical learning classifier systems to various hybrid learning systems, such as neural-based learning classifier systems [9], evolutionary artificial neural networks [10], evolutionary fuzzy systems [11], co-evolutionary learning systems [12], etc. While most of the learning problems considered in evolutionary learning are also investigated in the broader domain of machine learning, co-evolutionary learning has stood out as a learning paradigm that is rather unique to evolutionary computation.

1.3 Evolutionary Design

Evolutionary design is closely related to optimisation, especially in engineering domains, such as digital filter design [13] and shape design [14]. However, there is one different consideration in evolutionary design, which is evolutionary discovery. There has been a strong interest in using EAs as a discovery engine, in discovering novel designs, rather than just treating EAs as optimisers. A lot of work has appeared in using interactive evolutionary computation for creative design, e.g., traditional Batik design [15] and others.

1.4 Theoretical Foundations of Evolutionary Computation

In spite of numerous successes in real-world applications of EAs, theories of evolutionary computation have not progressed as fast as its applications. However, there have been significant advances in the theoretical foundation of evolutionary computation in the last decade or so. There have been a number of theoretical analyses of different fitness landscapes in terms of problem characterisation, as well as rigorous analysis of computation time used by an EA to solve a problem.

In global (numerical) optimisation, the analysis of EA's convergence has routinely been done. It was an active research topic in 1990s. Later on, convergence rates (to a local optimum) were also studied in depth. In recent years, there has been a significant growth in the computational time complexity analysis of EAs for combinatorial optimisation problems, which really bridges the gap between the analysis of EAs in the evolutionary computation field and the analysis of heuristics in theoretical computer science in general. After all, EAs are algorithms and can/should be analysed just like we analyse any other algorithms.

This book chapter will focus on the computational time complexity analysis of EAs for combinatorial optimisation problems. It will not cover other theories of evolutionary computation, which are equally important for the field. Even for the computational time complexity analysis of EAs, we will not be able to cover everything within a limited numner of pages. The choice of the topics covered in this chapter is highly biased by the author's own experience. One of the objectives of this chapter is to illustrate that there are some interesting theoretical results in evolutionary computation, which may help us in understanding why and when EAs work/fail. Such theoretical results are of interest in their own right. They also help to guide the design of better algorithms in the future.

The rest of this chapter is organised as follows. Section 2 introduces drift analysis as an easy-to-understand approach to analyse computational time complexity of EAs. General conditions under which an EA solves (or fails to solve) a problem within a polynomial time will be given. Section 3 presents a problem classification, which tries to identify hard and easy problem instances for a given EA. Such problem classification helps us to get a glimpse at potential characteristics that make a problem hard/easy for a given EA. Section 4 analysed the role of population in EAs. It is interesting to discover that a common belief — the large the population, the more powerful an EA would be — is not necessarily true. There are proven cases where a large population could be harmful. Section 5 investigates the impact of crossover on EA's computation time on the unique input output problem, a problem that occurs in finite state machine testing. This was the first time that crossover was analysed in depth on a non-artificial problem. All previous analyses on crossover were done on artificial problems. Section 6 examines the interactions of different components of an EA. Rather than analysing search operators (such as crossover and mutation) or selection mechanisms individually, this section is focused on the interactions between mutation and selection. It is shown that even parameter settings can have a significant on EA's computation, even when exactly the same EA was used. Section 7 discusses some recent results on analysing estimation of distribution algorithms (EDAs), which have rarely been studied in terms of computational time complexity analysis. A more general problem classification is also given. Finally, Section 8 concludes this chapter with some remarks and future research directions.

2 Evolutionary Algorithms and Drift Analysis

This section reviews some basic techniques used in analysing EAs as given previously [16]. The combinatorial optimization problem considered in this chapter can be described as follows: Given a finite state space S and a function $f(\mathbf{x}), \mathbf{x} \in S$, find

$$\mathbf{x}^* = \arg\max\{f(\mathbf{x}); \mathbf{x} \in S\},$$

where $\mathbf{x}^*$ is a state with the maximum function value, i.e., $f_{\max} = f(\mathbf{x}^*)$.

The EA for solving the combinatorial optimization problem can be described as follows:

1. *Initialization*: generate, either randomly or heuristically, an initial population of $2N$ individuals, $\xi_0 = (\mathbf{x}_1, \cdots, \mathbf{x}_{2N})$, and let $k \leftarrow 0$, where $N > 0$ is an integer. For any population ξ_k, define $f(\xi_k) = \max\{f(\mathbf{x}_i); \mathbf{x}_i \in \xi_k\}$.
2. *Generation*: generate a new intermediate population by crossover and mutation (or any other operators for generating offspring), and denote it as $\xi_{k+1/2}$.
3. *Selection*: select and reproduce $2N$ individuals from the combined population of $\xi_{k+1/2}$ and ξ_k, and obtain another new intermediate population ξ_{k+s}.
4. If $f(\xi_{k+s}) = f_{\max}$, then terminate the algorithm; otherwise let $\xi_{k+1} = \xi_{k+s}$ and $k \leftarrow k + 1$, and go to step 2.

The EA framework given above is very general because it allows for any initialization methods, any search operators and any selection mechanisms, to be used. The only difference from some EAs is that selection is applied after, not before, the search operators. However, the main results introduced in this chapter, i.e., Theorems 1 and 2, are independent of any such implementation details. In fact, the results in this section hold for virtually any stochastic search algorithms. They serve as the basis for many more specific results using specific EAs on specific problems.

2.1 Modelling EAs Using Stochastic Processes

Assume $\mathbf{x}^*$ is an optimal solution, and let $d(\mathbf{x}, \mathbf{x}^*)$ be the distance between a solution $\mathbf{x}$ and $\mathbf{x}^*$, where $\mathbf{x} \in S$. If there are more than one optimal solution (that is, a set S^*), we use $d(\mathbf{x}, S^*) = \min\{d(\mathbf{x}, \mathbf{x}^*) : \mathbf{x}^* \in S^*\}$ as the distance between individual $\mathbf{x}$ and the optimal set S^*. For convenience, we can denote the distance as $d(\mathbf{x})$, which satisfies $d(\mathbf{x}^*) = 0$ and $d(\mathbf{x}) > 0$ for any $\mathbf{x} \notin S^*$.

Given a population $X = \{\mathbf{x}_1, \cdots, \mathbf{x}_{2N}\}$, let

$$d(X) = \min\{d(\mathbf{x}) : \mathbf{x} \in X\},$$

which is used to measure the distance of this population to the optimum. The drift of the random sequence $\{d(\xi_k), k = 0, 1, \cdots\}$ at time k is defined by

$$\Delta(d(\xi_k)) = d(\xi_{k+1}) - d(\xi_k).$$

Define the stopping time of an EA as $\tau = \min\{k; d(\xi_k) = 0\}$, which is the **first hitting time** on an optimal solution. Our interest now is to investigate the relationship between the expected first hitting time and the problem size n, i.e., the computational time complexity of EAs in our context. In this chapter, we will establish the conditions under which an EA is guaranteed to find an optimal solution in polynomial time on average and conditions under which an EA takes at least exponential time on average to find an optimal solution. Such theoretical results help us to gain a better understanding of when and why an EA is expected to work well/poorly.

2.2 Conditions for Polynomial Average Computation Times

Theorem 1 ([16]). *If $\{d(\xi_k); k = 0, 1, 2, \cdots\}$ satisfies the following two conditions,*

1. *there exists a polynomial of problem size n, $h_0(n) > 0$, such that*
$$d(\xi_k) \leq h_0(n)$$
for any population ξ_k, and
2. *for any $k \geq 0$, if $d(\xi_k) > 0$, then there exists a polynomial of problem size n, $h_1(n) > 0$, such that*
$$E[d(\xi_k) - d(\xi_{k+1}) \mid d(\xi_k) > 0] \geq \frac{1}{h_1(n)},$$

then starting from any initial population ξ_0 with $d(\xi_0) > 0$,
$$E[\tau \mid d(\xi_0) > 0] \leq h(n),$$
where $h(n)$ is a polynomial of problem size n.

The first condition in the theorem implies that all populations occurred during the evolutionary search process are reasonably close to the optimum, i.e., their distances to the optimum is upper bounded by a polynomial in problem size. The second condition implies that, on average, the EA always drifts towards the optimum with at least some reasonable distance, i.e., the drifts are lower bounded by $\frac{1}{h_1(n)}$, where $h_1(n) > 0$ is a polynomial. The theorem basically says that the stochastic process defined by the EA can reach the optimum efficiently (in polynomial time) if the search is never too far away from the optimum and the drift towards the optimum is not too small.

Using the same intuition and analytical methods, as first proposed by Hajek [17], we can establish conditions under which an EA will take at least exponential time to reach an optimum.

2.3 Conditions for Exponential Average Computation Time

Theorem 2 ([16]). *Assume the following two conditions hold:*

1. *For any population ξ_k with $d_b < d(\xi_k) < d_a$, where $d_b \geq 0$ and $d_a > 0$,*
$$E[e^{-(d(\xi_{k+1}) - d(\xi_k))} \mid d_b < d(\xi_k) < d_a] \leq \rho < 1,$$
where $\rho > 0$ is a constant.
2. *For any population ξ_k with $d(\xi_k) \geq d_a$, $d_a > 0$,*
$$E[e^{-(d(\xi_{k+1}) - d_a)} \mid d(\xi_k) \geq d_a] \leq D,$$
where $D \geq 1$ is a constant.

If $d(\xi_0) \geq d_a$, $D \geq 1$ and $\rho < 1$, then there exist some $\delta_1 > 0$ and $\delta_2 > 0$ such that

$$E[\tau \mid d(\xi_0) \geq d_a] \geq \delta_1 e^{\delta_2(d_a - d_b)}$$

The first condition in the above theorem indicates that (d_b, d_a) is a very difficult interval to search. When this condition is satisfied, $d(\xi_{k+1}) > d(\xi_k)$. In other words, the offspring population is on average drifting away from the optimum, rather than getting closer to it. The second condition indicates that a population in the interval $[d_a, +\infty)$ will not, on average, drift towards the optimum too much because it is always quite far away from the optimum, i.e., $(d(\xi_{k+1})) \geq d_a - \ln D$.

Although the above two general theorems were first proved more than a decade ago [16], they still serve as foundations of many later results for EAs on specific problems, e.g., the subset sum problem [16], maximum matching [18, 19], vertex cover [20], unique input-output sequence [21], etc. The analytical techniques, i.e., drift analysis, advocated here is very intuitive and offer a general approach to analysing different EAs on different problems, which avoids the need to develop different and complicated analytical techniques for different EAs and problems.

3 Problem Classification: EA-hard vs EA-easy

Traditional complexity theories, such as NP-hardness, characterise the inherent complexity of a problem, regardless of any algorithms that might be used to solve them. However, we might not always encounter the worst case in practical cases. For a hard problem, we are interested in understanding what instance classes are hard and what instances are actually easy. When we analyse an algorithm, we want to know which problem instance classes are more amenable to this algorithm and which are not. Different instance classes of a problem pose different challenges to different algorithms. In evolutionary computation, we are particularly interested in problem characteristics that make the problem hard or easy for a given algorithm. A problem instance class may be very hard for one algorithm, but easy for another. Analysing the relationship between problem characteristics and algorithmic features will shed light on the essential question of when to use which algorithm in solving a difficult problem instance class. In order to emphasise such an algorithm-specific complexity concept, we introduce EA-hard and EA-easy problem instance classes in this section. For simplicity, we will just use the term problems to mean problem instance classes here.

Given an EA, we can divide all optimisation problems into two classes based on the mean number of generations (i.e., the mean first hitting time) needed to solve the problems [22].

Easy Class: For the given EA, starting from *any* initial population, the mean number of generations needed by the EA to solve the problem, i.e., $E[\tau|\xi_0]$, is *at most* polynomial in the problem size.

Hard Class: For the given EA, starting from *some* initial population, the mean number of generations needed by the EA to solve the problem, i.e., $E[\tau|\xi_0]$, is *at least* exponential in the problem size.

Theorem 3 ([22]). *Given an EA, a problem belongs to the* **EA-easy Class** if and only if *there exists a distance function $d(\xi_k)$, where ξ_k is the population at generation k, such that for any population ξ_k with $d(\xi_k) > 0$,*

1. *$d(\xi_k) \leq g_1(n)$, where $g_1(n)$ is polynomial in the problem size n, and*
2. *$E[d(\xi_k) - d(\xi_{k+1})|\xi_k] \geq c_{low}$, where $c_{low} > 0$ is a constant.*

Although the above theorem is closely related to Theorem 1, it shows stronger necessary and sufficient conditions. Similarly, the following theorem, related to Theorem 2, establishes the necessary and sufficient conditions for a problem to be hard for a given EA.

Theorem 4 ([22]). *Given an EA, a problem belongs to the* **EA-hard Class** if and only if *there exists a distance function $d(\xi_k)$, where ξ_k is the population at generation k, such that*

1. *for some population ξ_{k_1}, $d(\xi_{k_1}) \geq g_2(n)$, where $g_2(n)$ is exponential in the problem size n, and*
2. *for any population ξ_k with $d(\xi_k) > 0$, $E[d(\xi_k) - d(\xi_{k+1})|\xi_k] \leq c_{up}$, where $c_{up} > 0$ is a constant.*

The above two theorems can be used to verify whether a problem is hard/easy for a given EA. The key steps are to prove whether the two conditions hold. These conditions give us some important insight into problem characteristics that make a problem hard/easy for a given EA.

4 Is a Large Population Always Helpful?

Population has always been regarded as a crucial element of EAs. There have been numerous empirical studies that showed the benefit of a large population size. Whenever a problem becomes more challenging, one tends to increase the population size in an attempt to make the EA more 'powerful'. However, such an intuition might not be correct in all cases. He and Yao [23] first compared (1+1) EAs and (N+N) EAs theoretically. They showed cases where (N+N) EAs are indeed more efficient than (1+1) EAs, i.e., populations do help. They also showed somewhat surprising cases where (N+N) EAs might actually perform worse than (1+1) EAs, i.e., having a population actually makes an EA less efficient.

More recently, Chen *et al.* [24] investigated the population issue further and used the solvable rate as an improved performance measure of EAs. The solvable rate is a more precise performance measure than the mean first hitting time, because it considers a probability distribution, rather than just a mean.

Let $\tau = \min\{t|\mathbf{x}^* \in \xi_t\}$ be the first hitting time, where $\mathbf{x}^*$ is the global optimum and ξ_t is the population at the tth generation. The solvable rate κ is defined by

$$\kappa = P\left(\tau \prec Poly(n)\right),$$

where the event $\tau \prec Poly(n)$ means that there exists some polynomial function (of the problem size n) $h(n)$ such that $\tau < h(n)$ for any $n > n_0 > 0$.

Consider the following multi-modal TrapZeros problem with its global optimum at $\mathbf{x}^* = (1, ..., 1)$.

$$\textsc{TrapZeros}(\mathbf{x}) \triangleq \begin{cases} 2n + \sum_{i=1}^{n} \prod_{j=1}^{i}(1 - x_j), & \text{if } (x_1 = 0) \wedge (x_2 = 0); \\ 3n + \sum_{i=1}^{n} \prod_{j=1}^{i} x_j, & \text{if } (x_1 = 1) \wedge (x_2 = 1) \wedge \left(\prod_{i=3}^{\ln^2 n+2} x_i = 1\right); \\ n + \sum_{i=1}^{n} \prod_{j=1}^{i} x_j, & \text{if } (x_1 = 1) \wedge (x_2 = 1) \wedge \left(\prod_{i=3}^{\ln^2 n+2} x_i = 0\right); \\ 0, & \text{if } (x_1 = 0) \wedge (x_2 = 1); \\ 1, & \text{if } (x_1 = 1) \wedge (x_2 = 0); \\ 0, & \text{Otherwise.} \end{cases}$$

Consider the following $(N + N)$ EA used to solve the above problem.

Initialization: The N initial individuals are generated uniformly at random. Generation counter $k := 0$.

Mutation: For each individual in population ξ_k, one offspring is generated by flipping each bit independently with a uniform probability $1/n$, where n is the problem size (chromosome length). The offspring population is $\xi_k^{(m)}$.

Selection: Select the best N individuals from $\xi_k \cup \xi_k^{(m)}$ to form the next generation ξ_{k+1}. $k := k + 1$ and go to the mutation step.

This algorithm is very generic except for the lack of crossover, which we will discuss in the next section. The following results compare the EA's performance theoretically when $N = 1$ and $N > 1$.

Theorem 5 ([24]). *The first hitting time of the* $(1 + 1)$ *EA on* TrapZeros *is* $O(n^2)$ *with the probability of* $\frac{1}{4} - O\left(\frac{\ln^2 n}{n}\right)$. *In other words, the solvable rate of the* $(1 + 1)$ *EA on* TrapZeros *is at least* $\frac{1}{4} - O\left(\frac{\ln^2 n}{n}\right)$.

This theorem shows that $(1 + 1)$ EA can solve the problem in polynomial time with an almost constant probability.

Theorem 6 ([24]). *The first hitting time of the* $(N + N)$ *EA, where* $N = O(\ln n)$ *and* $N = \omega(1)$, *on* TrapZeros *is* $O\left(\frac{n^2}{N}\right)$ *with a probability of* $1/Poly(n)$, *where* $1/Poly(n)$ *refers to some positive function (of the problem size n), whose reciprocal is bounded from above by a polynomial function of the problem size n. In other words, the solvable rate of the* $(N + N)$ *EA on* TrapZeros *is at least* $1/Poly(n)$.

When the population size increases from 1 to greater than 1, but not too much greater (i.e., $N = O(\ln n)$), there is no significant gain in terms of EA's computation time, although the upper bound is decreased marginally from $O(n^2)$ to $O\left(\frac{n^2}{N}\right)$. Note that we do not have a near constant solvable rate anymore when the population size is greater than 1.

Theorem 7 ([24]). *The first hitting time of the $(N+N)$ EA, where $N = \Omega(n/\ln n)$, on* TrapZeros *is super-polynomial with an overwhelming probability. In other words, the solvable rate of the $(N+N)$ EA on* TrapZeros *is super-polynomially close to 0.*

Surprisingly, when the population size is very large, i.e., $N = \Omega(n/\ln n)$, the $(N+N)$ EA is no longer able to solve the TrapZeros in polynomial time. A large population size is actually harmful in this case!

Although the above study [24] was carried out on a specific problem using a specific type of EAs, it has actually revealed some interesting problem characteristics under which the $(N+N)$ EA may perform poorly: when the basin of attraction for a local optimum has relatively high fitness in comparison with most areas in the entire search space, a large population may be harmful, since it may lead to a large probability of finding individuals at the local basin. The search process towards and staying at the local basin can quickly eliminate other promising individuals that could lead to the global optimum later. When such congregation at the local basin happens, only large search step sizes can help to find promising individuals again, resulting in a long computation time towards the global optimum.

The weakness of the $(N+N)$ EA without crossover on the above problem characteristic can partially be tackled by employing larger search step sizes. Either an appropriately designed crossover operator or some adaptive/self-adaptive mutation schemes could work well with a large population in this case, as long as they can provide large search step sizes in exploring the correct attraction basin even if the whole population has been trapped in a local basin.

5 Impact of Crossover

The previous section used an artificial problem to gain some insight into the role of population in EAs. Crossover was not considered. This section introduces a real-world problem and analyses when crossover can be beneficial in improving EA's computation time.

Unique input-output sequences (UIO) have important applications in conformance testing of finite state machines (FSMs) [25]. In spite of much experimental work, few theoretical results exist [21, 26]. One significant result that does exist is the rigorous analysis of crossover's impact on EA's performance on one type of UIO problems [27].

Following [27], a finite state machine (FSM) is defined as a quintuple, $M = (I, O, S, \delta, \lambda)$, where $I(O)$ is the set of input (output) symbols, S is the set of states, $\delta : S \times I \to S$ is the state transition function, and $\lambda : S \times I \to O$ is the output function. A unique input-output sequence (UIO) for a state s in M is a string x over I such that $\lambda(s, x) \neq \lambda(t, x), \forall t, t \neq s$. In other words, x identifies state s uniquely. Although the shortest UIO in the general case can be exponentially long with respect to the number of states [25], our objective here is to search for an UIO of length n for state s in an FSM, where the fitness of an input sequence is defined as a function of the state partition tree induced by the input sequence [26]. In other words, given an FSM M with m states, the associated fitness function $UIO_{M,s} : I^n \to \mathcal{N}$ is defined as

$$UIO_{M,s}(x) := m - \gamma_M(s,x),$$

where

$$\gamma_M(s,x) := \left|\{t \in S | \lambda(s,x) = \lambda(t,x)\}\right|.$$

For theoretical analysis, a special FSM instance class, i.e., the TwoPaths problem, is introduced here [27].

For instance size n and constant ϵ, $0 < \epsilon < 1$, a TwoPaths FSM has input and output symbols $I := \{0,1\}$ and $O := \{a,b,c\}$, respectively, and $2(n+1)$ states $S = R \cup Q$, where $R := \{s_1, s_2, \ldots, s_{n+1}\}$and $Q := \{q_1, q_2, \ldots, q_{n+1}\}$. The output function λ is

$$\lambda(q_i,x) := \begin{cases} c, \text{ if } i = n+1 \text{ and } x = 0 \\ a, \text{ otherwise} \end{cases}$$

$$\lambda(s_i,x) := \begin{cases} b, \text{ if } i = n+1 \text{ and } x = 1 \\ a, \text{ otherwise} \end{cases}$$

The state transition function δ is

$$\delta(s_i,0) := \begin{cases} q_{(1-\epsilon)n+3}, & \text{if } i = (1-\epsilon)n+1, \\ s_1, & \textit{otherwise} \end{cases}$$

$$\delta(s_i,1) := \begin{cases} q_1, & \text{if } i = n+1 \\ s_{i+1}, & \textit{otherwise} \end{cases}$$

$$\delta(q_i,1) := q_1$$

$$\delta(q_i,0) := \begin{cases} s_1, & \text{if } i = n+1 \\ q_{i+1}, & \textit{otherwise} \end{cases}$$

We can use the following $(N+1)$ steady state EA (SSEA) [27] to solve the above problem.

Initialisation: Initialise N individuals uniformly at random from $\{0,1\}^n$ to form the initial population P_0. $i = 0$.

Reproduction: Perform one of the following two choices

1-point Crossover: With probability $p_c(n)$, select $\mathbf{x}$ and $\mathbf{y}$ uniformly at random from population P_i. Select k from $\{1, \ldots, n\}$ uniformly at random. Perform 1-point crossover between $\mathbf{x}$ and $\mathbf{y}$ and obtain

$$\mathbf{x}' := x_1 x_2 \cdots x_{k-1} y_k y_{k+1} \cdots y_n,$$

$$\mathbf{y}' := y_1 y_2 \cdots y_{k-1} x_k x_{k+1} \cdots x_n.$$

If $\max\{f(\mathbf{x}'), f(\mathbf{y}')\} \geq \max\{f(\mathbf{x}), f(\mathbf{y})\}$, then $\mathbf{x} := \mathbf{x}', \mathbf{y} := \mathbf{y}'$.

Mutation Only: With probability $1 - p_c(n)$, select $\mathbf{x}$ from P_i uniformly at random. Flip each bit of $\mathbf{x}$ independently with probability $1/n$. If the result is no worse than $\mathbf{x}$, the mutant replaces $\mathbf{x}$.

$i := i + 1$**:** and go to the Reproduction step.

Given the UIO problem and the SSEA as described above, the following results show clearly the significant impact crossover has on SSEA's performance.

Theorem 8 ([27]). *For a sufficiently large constant $c > 0$, if the $(N+1)$ SSEA with a constant crossover probability $p_c > 0$ and population size N, $2 \leq N = Poly(n)$, is restarted every cN^2n^2 generations on* TwoPaths*, then the expected first hitting time is $O(N^2n^2)$.*

In other words, the optimum can be found within polynomial time as long as crossover is used. The following theorem shows that it is no longer possible to find the optimum in polynomial time if crossover is not used.

Theorem 9 ([27]). *If the crossover probability $p_c = 0$, the probability that the $(N + 1)$ SSEA with population size $N = Poly(n)$ finds the optimum of* TwoPaths *within 2^{cn} generations, where c is a constant, is upper-bounded by $e^{-\Omega(n)}$.*

It is important to note that these two theorems only state the benefits of this crossover operator for the TwoPaths problem. The conclusions should not be generalised to other problems without new proofs, because different search operators are effective on different problems. There are problems on which crossover will not be beneficial.

6 Interaction between Operators/Parameters

The performance of an EA is determined by its operators, parameters and interactions among them. While there have been studies on individual operators, such as crossover described in the previous section, and parameters, such as population size as discussed in Section 4, only one study [28] exists, which analyses the interaction of two operators, i.e., mutation and selection. It was shown in this work that neither mutation nor selection alone could determine the performance of an EA [28]. It was their combined effect that determined EA's performance. While this might sound intuitive, it was the first time that a rigorous analysis was given.

Let's investigate a non-elitist population-based EA with the linear ranking scheme [28], which captures many features of the EAs used in practice.

Initialisation: Generate N individuals at random for the initial population P_0. Each individual $P_0(i)$ is generated by sampling $\{0,1\}^n$ uniformly at arandom, $i \in \{1, 2, \ldots, N\}$. $t := 0$.

Evolutionary Cycle: Repeat the following until certain stopping criterion is met.

1. Sort P_t according to fitness f such that

$$f(P_t(1)) \geq f(P_t(2)) \geq \cdots \geq f(P_t(N)).$$

2. For $i \in \{1, 2, \ldots, N\}$,

```
    (a) Sample $r$ from $\{1, \ldots, N\}$ using the linear ranking scheme.
    (b) $P_{t+1} := P_t(r)$.
    (c) Flip each bit in $P_{t+1}(i)$ with probability $\chi/n$.
3. $t := t + 1$.
```

In the above algorithm, N is the population size and χ determines the mutation probability. Both are fixed during evolution. To illustrate the importance of selection-mutation balance in this algorithm, the following problem is considered.

For any constants $\sigma, \delta, 0 < \delta < \sigma < 1 - 3\delta$, and integer $k \geq 1$, the fitness function considered here is [28]

$$\textsc{SelPres}_{\sigma,\delta,k}(x) := \begin{cases} 2n, & \text{if } x \in X^*_\sigma, \\ \sum_{i=1}^n \prod_{j=1}^i x_j, & \text{otherwise} \end{cases}$$

where the set of optimal solutions X^*_σ contain all bitstrings $\mathbf{x} \in \{0,1\}^n$ satisfying

$$\|x[1, k+3]\| = 0,$$

$$\|x[k+4, (\sigma - \delta)n - 1]\| = 1,$$

$$\|x[(\sigma + \delta)n, (\sigma + 2\delta)n - 1]\| \leq 2/3.$$

Theorem 10 ([28]). *For any constant integer $k \geq 1$, let T be the runtime of the non-elitist population-based EA with linear ranking selection. Its population size N satisfies $n \leq N \leq n^k$. It has a constant selection pressure of η, where $1 < \eta \leq 2$. The bit-wise mutation rate is χ/n. On function* $\textsc{SelPres}_{\sigma,\delta,k}$*, for any constant $\epsilon > 0$,*

1. *If $\eta < \exp(\chi(\sigma - \delta)) - \epsilon$, then for some constant $c > 0$,*
$$Pr(T \geq e^{cn}) = 1 - e^{-\Omega(n)}.$$
2. *If $\eta = \exp(\chi\sigma)$, then*
$$Pr(T \leq n^{k+4}) = 1 - e^{-\Omega(n)}.$$
3. *If $\eta > \frac{2\exp(\chi(\sigma+3\delta))-1}{1-\delta}$, then*
$$E(T) = e^{\Omega(n)}.$$

A couple of observations can be made from the above theorem. First, the theorem shows an interesting relationship between selection pressure η and mutation rate χ. Neither determines the EA's computation time by itself. If selection pressure is high, it can be compensated by a high mutation rate to achieve the balance between the two, i.e., $\eta = \exp(\chi\sigma)$. If selection pressure is too low, we can lower the mutation rate accordingly to maintain an efficient EA. This theorem also suggests that trying to increase the mutation rate in order to increase evolvability and the ability of escaping from local optima may not work well for some problems, unless selection pressure is also increased appropriately.

Second, the EA's computation time is very sensitive to the ratio between η and χ. The ratio needs to be in a very narrow range around $\eta = \exp(\chi\sigma)$ to achieve EA's efficiency, i.e., polynomial computation time. Given a mutation rate, either a slightly small selection pressure or a moderately larger one will lead to exponential computation time. This is a rather unique example that unpacks the relationship between the EA and the problem, and sheds light into how the parameter interactions affect EA's performance on this problem.

7 Estimation of Distribution Algorithms (EDAs)

Although estimation of distribution algorithms (EDAs) were proposed and studied in the field of evolutionary computation, they are actually very different from other EAs. Instead of using any search operators, EDAs rely on model-building and sampling.

Initialisation: Generate N individuals using the initial probability distribution. $t := 0$.

Iterations: Repeat the following until the stopping criterion is met.

1. M individuals are selected from the population of N individuals;
2. A probability distribution is estimated from these M individuals;
3. N individuals are sampled from this estimated probability distribution;
4. $t := t + 1$.

Similar to Section 3, given an EDA, we can classify all problem instance classes into hard and easy cases [29].

EDA-easy Class. For a given EDA, a problem is EDA-easy *if and only if*, with the probability of $1 - 1/SuperPoly(n)$, the first hitting time needed to reach the global optimum is polynomial in the problem size n.

EDA-hard Class. For a given EDA, a problem is ED-hard *if and only if*, with the probability of $1/Poly(n)$, the first hitting time needed to reach the global optimum is superpolynomial in the problem size n.

Note the hardness definition here is EDA-dependent, because we are interested in the relationship between algorithms and problems. The above definition is similar to but different from that in Section 3 because the probabilities are used here, not mean first hitting times as in Section 3.

We define formally an optimisation problem as $I = (\Omega, f)$, where Ω is the search space and f the fitness function. $\mathcal{P} = (\Omega, f, \mathcal{A})$ indicates an algorithm $\mathcal{A}$ on a fitness function f in the search space Ω. P_t^* indicates the probability of generating the global optimum in one sampling at the t-th generation.

Given a function $f(n)$, where $f(n) > 1$ always holds and when $n \to \infty, f(n) \to \infty$, denote

1. $f(n) \prec Poly(n)$ and $g(n) = \frac{1}{f(n)} \succ \frac{1}{Poly(n)}$ if and only if $\exists a, b > 0, n_0 > 0 : \forall n > n_0, f(n) \leq an^b$.

2. $f(n) \succ SuperPoly(n)$ and $g(n) = \frac{1}{f(n)} \prec \frac{1}{SuperPoly(n)}$ if and only if $\forall a, b > 0 : \exists n_0 : \forall n > n_0, f(n) > an^b$.

Theorem 11 ([29]). *For a given $\mathcal{P} = (\Omega, f, \mathcal{A})$, if the population size N of the EDA $\mathcal{A}$ is polynomial in the problem size n, then*

1. *if problem I is* **EDA-easy** *for $\mathcal{A}$, then $\exists t' \le \lceil E[\tau(\mathcal{P}) | \tau(\mathcal{P}) \prec Poly(n)] \rceil + 1$ such that*
$$P^*_{t'} \succ \frac{1}{Poly(n)};$$
2. *if $\forall t \prec Poly(n), P^*_t \prec \frac{1}{SuperPoly(n)}$, then problem I is* **EDA-hard** *for $\mathcal{A}$.*

Because the hardness definition used here is algorithm dependent. A problem that is easy for one EDA can be hard for another EDA or another EA. Chen *et al.* [29] described one problem that is EA-easy but EDA-hard. An example of EA-hard bu EDA-easy problems is yet to be found. Such theoretical comparison of problem hardness under different algorithms can often lead to a better understanding of the algorithms and shed light on the issue of what algorithmic features are most effective in tackling certain problem characteristics.

8 Concluding Remarks

Although most research in evolutionary computation relies on computational studies, there have been an increasing number of theoretical results in recent years. Significant progresses in analysing the computational time complexity of EAs have been made. Not only have there been a large number of papers on evolutionary computation theories in journals in evolutionary computation, artificial intelligence and theoretical computer science, there are also published books [30, 31]. One of the three sections, Section C, of the well-established *Theoretical Computer Science (TCS)* journal is entirely devoted to Natural Computing. According to Elsevier (`http://www.journals.elsevier.com/theoretical-computer -science/most-cited-articles/`), two of the top three most cited TCS papers published since 2007 are on evolutionary computation theories. This chapter only reviewed a tiny part of the results in evolutionary computation theory.

In spite of significant progresses, there is still much work to be done in developing better theories for evolutionary computation. There are several future research directions that seem to be particularly attractive and important.

First, the analysis of EDAs has been very few. The work by Chen *et al.* [29] investigated UMDAs only and on two artificial problems. More work is needed to analyse other EDAs on non-artificial problems. In particular, it will be very interesting to study when an EDA is likely to outperform an EA and why [32]. It is also interesting to analyse the impact of different probabilistic models on EDA's performance. Is it possible to improve EDA's performance by using a more powerful probabilistic model? When will a more powerful probabilistic model help?

Second, many real world problems are dynamic. Yet the analysis of EAs on dynamic problems is lagging behind applications. The existing work in this topic area is still in its infancy [33, 34]. There is a need for more theoretical work to complement computational studies in this area.

Third, all the work reviewed here focused on the time of finding the global optimal solution. In practice, good approximate solutions are often sufficient. Theoretical analysis of evolutionary approximation algorithms has shown some promising results [18–20, 35]. It has been shown that EAs from a random initial population can perform just as well as tailored heuristics for certain combinatorial optimisation problems. Can we find an example that an EA finds an approximate solution to a problem more efficiently than a human-designed heuristic?

Fourth, there has been some interest in algorithm portfolios in evolutionary computation [36, 37]. Computational studies have shown very promising results. However, it is unclear whether or not such type of algorithms offers any theoretical advantages over conventional ones. This is certainly an interesting challenge for theoretical research.

Fifth, the work reviewed in this chapter is all related to combinatorial optimisation. Yet EAs are equally often used in global (numerical) optimisation. There has been excellent work on the convergence and convergence rates of various EAs. However, theoretical analysis of EA's scalability has been few, in spite of recent surge in the interest of large scale optimisation [38–40]. It is still unclear what the relationship is between the optimisation time and the problem size (in terms of dimensionality) for different EAs on different problems. In fact, it is not entirely clear what a good measure for the optimisation time should be. The convergence time may not be very interesting from a practical point of view as we may not find the exact global optimum in finite time. It is more interesting to analyse the computation time towards a near optimum. Maybe we should explore the potential links to Blum *et al.*'s seminal work [41].

Acknowledgement. This work was partially supported by an EPSRC grant (No. EP/I010297/1).

References

1. Yao, X., Liu, Y., Lin, G.: Evolutionary programming made faster. IEEE Transactions on Evolutionary Computation 3, 82–102 (1999)
2. Li, B., Lin, J., Yao, X.: A novel evolutionary algorithm for determining unified creep damage constitutive equations. International Journal of Mechanical Sciences 44(5), 987–1002 (2002)
3. Yang, Z., Tang, K., Yao, X.: Self-adaptive differential evolution with neighborhood search. In: Proceedings of the 2008 IEEE Congress on Evolutionary Computation (CEC 2008), pp. 1110–1116. IEEE Press, Piscataway (2008)
4. Yang, Z., Li, X., Bowers, C., Schnier, T., Tang, K., Yao, X.: An efficient evolutionary approach to parameter identification in a building thermal model. IEEE Transactions on Systems, Man, and Cybernetics — Part C (2012), doi:10.1109/TSMCC.2011.2174983

5. Tang, K., Mei, Y., Yao, X.: Memetic algorithm with extended neighborhood search for capacitated arc routing problems. IEEE Transactions on Evolutionary Computation 13, 1151–1166 (2009)
6. Handa, H., Chapman, L., Yao, X.: Robust route optimisation for gritting/salting trucks: A CERCIA experience. IEEE Computational Intelligence Magazine 1, 6–9 (2006)
7. Praditwong, K., Harman, M., Yao, X.: Software module clustering as a multi-objective search problem. IEEE Transactions on Software Engineering 37, 264–282 (2011)
8. Wang, Z., Tang, K., Yao, X.: Multi-objective approaches to optimal testing resource allocation in modular software systems. IEEE Transactions on Reliability 59, 563–575 (2010)
9. Dam, H.H., Abbass, H.A., Lokan, C., Yao, X.: Neural-based learning classifier systems. IEEE Transactions on Knowledge and Data Engineering 20, 26–39 (2008)
10. Yao, X., Islam, M.M.: Evolving artificial neural network ensembles. IEEE Computational Intelligence Magazine 3, 31–42 (2008)
11. Cordón, O., Gomide, F., Herrera, F., Hoffmann, F., Magdalena, L.: Ten years of genetic fuzzy systems: current framework and new trends. Fuzzy Sets and Systems 141(1), 5–31 (2004)
12. Chong, S.Y., Tino, P., Yao, X.: Measuring generalization performance in co-evolutionary learning. IEEE Transactions on Evolutionary Computation 12, 479–505 (2008)
13. Salcedo-Sanz, S., Cruz-Roldán, F., Heneghan, C., Yao, X.: Evolutionary design of digital filters with application to sub-band coding and data transmission. IEEE Transactions on Signal Processing 55, 1193–1203 (2007)
14. Zhang, P., Yao, X., Jia, L., Sendhoff, B., Schnier, T.: Target shape design optimization by evolving splines. In: Proc. of the 2007 IEEE Congress on Evolutionary Computation (CEC 2007), pp. 2009–2016. IEEE Press, Piscataway (2007)
15. Li, Y., Hu, C., Yao, X.: Innovative batik design with an interactive evolutionary art system. J. of Computer Sci. and Tech. 24(6), 1035–1047 (2009)
16. He, J., Yao, X.: Drift analysis and average time complexity of evolutionary algorithms. Artificial Intelligence 127, 57–85 (2001)
17. Hajek, B.: Hitting time and occupation time bounds implied by drift analysis with applications. Adv. Appl. Probab. 14, 502–525 (1982)
18. He, J., Yao, X.: Maximum cardinality matching by evolutionary algorithms. In: Proceedings of the 2002 UK Workshop on Computational Intelligence (UKCI 2002), Birmingham, UK, pp. 53–60 (September 2002)
19. He, J., Yao, X.: Time complexity analysis of an evolutionary algorithm for finding nearly maximum cardinality matching. J. of Computer Sci. and Tech. 19, 450–458 (2004)
20. Oliveto, P., He, J., Yao, X.: Analysis of the (1+1)-ea for finding approximate solutions to vertex cover problems. IEEE Transactions on Evolutionary Computation 13, 1006–1029 (2009)
21. Lehre, P.K., Yao, X.: Runtime analysis of the (1+1) ea on computing unique input output sequences. Information Sciences (2010), doi:10.1016/j.ins.2010.01.031
22. He, J., Yao, X.: A study of drift analysis for estimating computation time of evolutionary algorithms. Natural Computing 3, 21–35 (2004)
23. He, J., Yao, X.: From an individual to a population: An analysis of the first hitting time of population-based evolutionary algorithms. IEEE Transactions on Evolutionary Computation 6, 495–511 (2002)

24. Chen, T., Tang, K., Chen, G., Yao, X.: A large population size can be unhelpful in evolutionary algorithms. Theoretical Computer Science (2011), doi:10.1016/j.tcs.2011.02.016
25. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines — a survey. Proceedings of the IEEE 84(8), 1090–1123 (1996)
26. Lehre, P.K., Yao, X.: Runtime analysis of (1+1) ea on computing unique input output sequences. In: Proc. of the 2007 IEEE Congress on Evolutionary Computation (CEC 2007), pp. 1882–1889. IEEE Press, Piscataway (2007)
27. Lehre, P.K., Yao, X.: Crossover can be constructive when computing unique input-output sequences. Soft Computing 15, 1675–1687 (2011)
28. Lehre, P.K., Yao, X.: On the impact of mutation-selection balance on the runtime of evolutionary algorithms. IEEE Transactions on Evolutionary Computation (2011), doi:10.1109/TEVC.2011.2112665
29. Chen, T., Tang, K., Chen, G., Yao, X.: Analysis of computational time of simple estimation of distribution algorithms. IEEE Transactions on Evolutionary Computation 14, 1–22 (2010)
30. Neumann, F., Witt, C.: Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational Complexity. Springer, Berlin (2010)
31. Auger, A., Doerr, B. (eds.): Theory of Randomized Search Heuristics: Foundations and Recent Developments. World Scientific, Singapore (2011)
32. Chen, T., Lehre, P.K., Tang, K., Yao, X.: When is an estimation of distribution algorithm better than an evolutionary algorithm? In: Proceedings of the 2009 IEEE Congress on Evolutionary Computation, pp. 1470–1477. IEEE Press, Piscataway (2009)
33. Droste, S.: Analysis of the (1+1) ea for a dynamically changing onemax-variant. In: Proceedings of the 2002 IEEE Congress on Evolutionary Computation, pp. 55–60. IEEE Press, Piscataway (2002)
34. Rohlfshagen, P., Lehre, P.K., Yao, X.: Dynamic evolutionary optimisation: An analysis of frequency and magnitude of change. In: Proceedings of the 2009 Genetic and Evolutionary Computation Conference, pp. 1713–1720. ACM Press, New York (2009)
35. Yu, Y., Yao, X., Zhou, Z.-H.: On the approximation ability of evolutionary optimization with application to minimum set cover. Artificial Intelligence (2012), doi:10.1016/j.artint.2012.01.001
36. Fukunaga, A.S.: Genetic algorithm portfolios. In: Proceedings of the 2000 IEEE Congress on Evolutionary Computation, pp. 16–19. IEEE Press, Piscataway (2000)
37. Peng, F., Tang, K., Chen, G., Yao, X.: Population-based algorithm portfolios for numerical optimization. IEEE Transactions on Evolutionary Computation 14, 782–800 (2010)
38. Yang, Z., Tang, K., Yao, X.: Large scale evolutionary optimization using cooperative coevolution. Information Sciences 178, 2985–2999 (2008)
39. Yang, Z., Tang, K., Yao, X.: Scalability of generalized adaptive differential evolution for large-scale continuous optimization. Soft Computing 15, 2141–2155 (2011)
40. Li, X., Yao, X.: Cooperatively coevolving particle swarms for large scale optimization. IEEE Transactions on Evolutionary Computation (2011), doi:10.1109/TEVC.2011.2112662
41. Blum, L., Shub, M., Smale, S.: On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. Bulletin of the American Mathematical Society 21, 1–46 (1989)