Soft Comput (2011) 15:1675–1687
DOI 10.1007/s00500-010-0610-2

FOCUS

Crossover can be constructive when computing unique input–output sequences

Per Kristian Lehre · Xin Yao

Published online: 9 June 2010

Abstract Unique input–output (UIO) sequences have important applications in conformance testing of finite state machines (FSMs). Previous experimental and theoretical research has shown that evolutionary algorithms (EAs) can compute UIOs efficiently on many FSM instance classes, but fail on others. However, it has been unclear how and to what degree EA parameter settings influence the runtime on the UIO problem. This paper investigates the choice of acceptance criterion in the $(1 + 1)$ EA and the use of crossover in the $(\mu + 1)$ Steady State Genetic Algorithm. It is rigorously proved that changing these parameters can reduce the runtime from exponential to polynomial for some instance classes of the UIO problem.

Keywords Finite state machines · Unique input–output sequences · Evolutionary algorithms · Runtime analysis · Crossover operator

1 Introduction

Evolutionary algorithms (EAs) are general purpose optimisation algorithms. Although problem domain knowledge is useful, EAs can in principle be applied with little problem domain knowledge, only requiring the user to provide the algorithm with a set of candidate solutions and a way of measuring the quality of each candidate solution. This generality allows EAs to be applied in diverse problem domains, as has been documented extensively. In practice, the application of EAs is often not straightforward as it is often necessary to adjust the parameter settings to the problem at hand. Due to a poor understanding in how and why genetic operators influence the search process, this parameter tuning is often expensive.

Theoretical research like runtime analysis will seldom provide optimal parameter settings for specific real world problems. However, it may provide insight into how and why EAs work and sometimes fail. In particular, a theoretical analysis can point out simple general cases where the choice of a genetic operator has a particularly important effect on the runtime. Equipped with an understanding of how EAs behave in such archetypical cases, a practitioner will be able to make better informed decisions as to how to choose parameter settings on a specific real world problem. This paper analyses rigorously the influence of genetic operators on the problem of computing unique input–output (UIO) sequences from deterministic finite state machines (FSMs), a computationally hard problem from the software engineering domain (Lehre and Yao 2007). UIOs have important applications in conformance testing of FSMs (Lee and Yannakakis 1996). Similarly to other approaches in search-based software engineering (Clarke et al. 2003), the UIO problem has been reformulated as an optimisation problem and tackled with EAs (Derderian et al. 2006, Guo et al. 2004). Experimental results show that EAs can construct UIOs efficiently on some instances. Guo et al. (2004) compared an evolutionary approach with a random search strategy, and found that the two

A preliminary version of this paper appeared in Proceedings of the 7th International Conference on Simulated Evolution and Learning (SEAL'08) (Lehre and Yao 2008).

P. K. Lehre (✉) · X. Yao
The Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), School of Computer Science, The University of Birmingham, Edgbaston, Birmingham B15 2TT, UK
e-mail: P.K.Lehre@cs.bham.ac.uk

X. Yao
e-mail: X.Yao@cs.bham.ac.uk

approaches have similar performance on a small FSM, while the evolutionary approach outperforms random search on a larger FSM. Derderian et al. (2006) presented an alternative evolutionary approach, confirming Guo et al's results.

Previous theoretical results confirm that EAs can outperform random search on the UIO problem (Lehre and Yao 2007). The expected running time of (1 + 1) EA on a counting FSM instance class is $O(n \log n)$, while random search needs exponential time (Lehre and Yao 2007). The UIO problem is NP-hard (Lee and Yannakakis 1996), so one can expect that there exist EA-hard instance classes. It has been proved that a combination lock FSM is hard for the (1 + 1) EA (Lehre and Yao 2007). To reliably apply EAs to the UIO problem, it is necessary to distinguish easy from hard instance classes. Theoretical results indicate that there is no sharp boundary between these categories in terms of runtime. For any polynomial n^k, there exist UIO instance classes where the (1 + 1) EA has running time $\Theta(n^k)$ (Lehre and Yao 2007).

Do EA settings have any significant impact on the chance of finding UIOs efficiently? Guo et al. (2005) hypothesise that crossover is helpful, without giving further evidence than two example sequences that recombine into a UIO. This paper provides, for the first time, rigorous theoretical results that prove that crossover can be essential for finding UIO in polynomial time. The results also show how modifying the acceptance criterion of an EA can have a similarly drastic impact on the runtime. The remaining of this section provides preliminaries, followed by the main results in Sects. 2 and 3.

In this paper, we will consider deterministic Mealy FSMs which can be defined as follows.

Definition 1 (*Finite state machine*) A finite state machine (FSM) M is a quintuple, $M = (I, O, S, \delta, \lambda)$, where I is the set of input symbols, O is the set of 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.

When receiving input symbol x, the machine makes the transition from its current state s to a next state $\delta(s, x)$ and outputs symbol $\lambda(s, x)$. The functions λ and δ are generalised to the domain of input sequences in the obvious way.

Definition 2 (*Unique input–output sequence*) A unique input–output sequence (UIO) for a state s in an FSM M is a string x over the input alphabet I such that $\lambda(s, x) \neq \lambda(t, x)$ for all states $t, t \neq s$.

Definitions 1 and 2 are illustrated in Fig. 1. An edge (s_i, s_j) labelled i/o defines the transition $\delta(s_i, i) = s_j$ and the output $\lambda(s_i, i) = o$. The input sequence 0202 is a UIO for state s_1, because only starting from state s_1 will the FSM output the sequence *aaab*. The single input symbol 2 is a UIO for state s_2, because only state s_2 has output b on input 2. This FSM does not have any distinguishing sequence, because for every input symbol x in the FSM, there exist at least two states s and t such that $\lambda(s, x) = \lambda(t, x)$ and $\delta(s, x) = \delta(t, x)$.

To compute UIOs with EAs, candidate solutions are represented as strings over the input alphabet of the FSM, which is here restricted to $I = \{0, 1\}$ (Guo et al. 2004). Although the shortest UIOs in the general case can be exponentially long with respect to the number of states (Lee and Yannakakis 1996), all the instance classes presented here have UIOs of length n. The objective in this paper is to search for an UIO of length n for state s_1 in various FSMs, where the fitness of a input sequence is defined as a function of the state partition tree induced by the input sequence (Guo et al. 2004; Lehre and Yao 2007).

Definition 3 (*UIO fitness function*) Given a finite state machine M with m states, the associated fitness function $\mathrm{UIO}_{M,s} : I^n \to \mathbb{N}$ is defined as

$$\mathrm{UIO}_{M,s}(x) := m - \gamma_M(s, x),$$

where the function γ_M is defined as

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

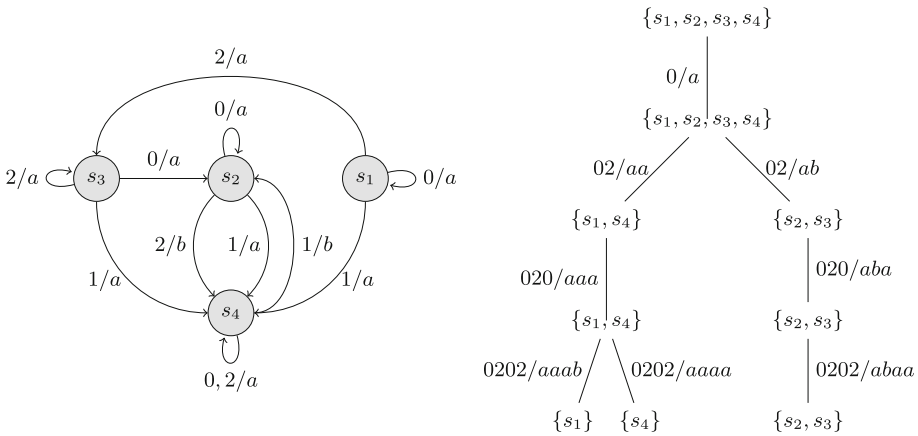

Fig. 1 FSM with corresponding state partition tree for the input sequence 0202

The *instance size* of fitness function UIO_{M,s_1} is here defined as the length of the input sequence n. The value of $\gamma_M(s,x)$ is the number of states in the leaf node of the state partition tree containing node s, and is in the interval from 1 to m. If the shortest UIO for state s in FSM M has length no more than n, then $\text{UIO}_{M,s}$ has an optimum of $m-1$, where m is the number of states. The following obvious lemma will be useful when characterising fitness functions corresponding to FSMs.

Lemma 1 *For any FSM $M=(I,O,S,\delta,\lambda)$ and pair of states $s,t\in S$ and pair of input sequences $x,y\in I^*$, if $\lambda(s,xy)=\lambda(t,xy)$ then $\lambda(s,x)=\lambda(t,x)$.*

Proof If $\lambda(s,xy)=\lambda(s,x)\cdot\lambda(\delta(s,x),y)$ equals $\lambda(t,xy)=\lambda(t,x)\cdot\lambda(\delta(t,x),y)$, then $\lambda(s,x)=\lambda(t,x)$. □

The goal of analysing the runtime of a search algorithm on a problem is to derive expressions showing how the number of iterations the algorithm uses to find the optimum depends on the problem instance size. The time is here measured as the number of fitness evaluations.

Definition 4 [*Runtime (Droste et al. 2002; He and Yao 2004)*] Given a class $\mathcal{F}$ of fitness functions $f_i:S_i\to\mathbb{R}$, the *runtime* $T_{A,\mathcal{F}}(n)$ of a search algorithm A is defined as

$$T_{A,\mathcal{F}}(n):=\max\{T_{A,f}\mid f\in\mathcal{F}_n\},$$

where $\mathcal{F}_n$ is the subset of functions in $\mathcal{F}$ with instance size n, and $T_{A,f}$ is the number of times algorithm A evaluates the cost function f until the optimal value of f is evaluated for the first time.

For a randomised search algorithm A, the runtime $T_{A,\mathcal{F}}(n)$ is a random variable. Runtime analysis of randomised search heuristics estimates properties of the distribution of $T_{A,\mathcal{F}}(n)$, in particular, the *expected runtime* $\mathbf{E}[T_{A,\mathcal{F}}(n)]$, and the *success probability*, which for a time bound $t(n)$ is defined as $\mathbf{Pr}[T_{A,\mathcal{F}}(n)\le t(n)]$. The analysis uses standard notation (e.g., O,Ω and Θ) for asymptotic growth of functions (Cormen et al. 2001).

2 Impact of acceptance criterion

The (1 + 1) EA is a simple single-individual-based algorithm, which in each iteration replaces the current search point x by a new search point x' if and only if $f(x')\ge f(x)$. The variant $(1+1)^*$ EA adopts the more restrictive acceptance criterion $f(x')>f(x)$. There exists an artificial pseudo-boolean function Spc where (1 + 1) EA is efficient while $(1+1)^*$ EA fails (Jansen and Wegener 2001). Here, it is shown that the same result holds on the UIO problem for the Ridge FSM instance class.

Definition 5 *((1 + 1) EA)*

Choose x uniformly from $\{0,1\}^n$.
Repeat
  $x':=x$.
  Flip each bit of x' with probability $1/n$.
  If $f(x')\ge f(x)$,
    then $x:=x'$.

Definition 6 (Ridge *FSM*) For instance sizes $n\ge 2$, define a Ridge FSM with input and output symbols $I:=\{0,1\}$ and $O:=\{a,b\}$, respectively, and $2n$ states $S:=Q\cup R$, where $Q:=\{q_1,q_2,\ldots,q_n\}$ and $R:=\{s_1,s_2,\ldots,s_n\}$. For all states q_i and s_i, $1\le i\le n$, define the transition function as $\delta(q_i,0):=q_i$, $\delta(s_i,1):=s_1$, and

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

$$\delta(s_i,0):=\begin{cases}q_1 & \text{if } i=n,\text{ and}\\ s_{i+1} & \text{otherwise.}\end{cases}$$

And for all states q_i and s_i, $1\le i\le n$, define the output function λ as $\lambda(q_i,0):=a$, $\lambda(s_i,1):=a$, and

$$\lambda(q_i,1):=\begin{cases}b & \text{if } i=n,\\ a & \text{otherwise.}\end{cases}$$

$$\lambda(s_i,0):=\begin{cases}b & \text{if } i=n,\\ a & \text{otherwise.}\end{cases}$$

The Ridge FSM instance class is illustrated in Fig. 2. The fitness function $\text{UIO}_{\text{Ridge},s_1}$ can be expressed as a pseudo-boolean function.

Proposition 1 *The fitness function $\text{UIO}_{\text{Ridge},s_1}$ associated with the Ridge FSM instance class of size n takes the values*

$$\text{Ridge}(x)=\begin{cases}2n-1 & \text{if } x=0^n,\\ \sum_{i=1}^n x_i+\sum_{i=1}^n\prod_{j=1}^i(1-x_j) & \text{otherwise.}\end{cases}$$

Proof The transitions (s_n,q_1) and (q_n,s_1) are called *distinguishing* because they have unique input/output behaviours, whereas all other states output a on any input symbol. Clearly, for any two states s and t and input sequence x, if neither state s nor state t reaches any distinguishing transition on input sequence x, then $\lambda(s,x)=\lambda(t,x)=a^{\ell(x)}$.

For input sequences x of length n that are different from 0^n, we make two claims. First, the number of states, among the states q_i, $1\le i\le n$, with different output than state s_1 equals $\text{OneMax}(x):=\sum_{i=1}^n x_i$. And second, the number of states, among the states s_i, $2\le i\le n$, with different output than state s_1 equals $\text{Lz}(x):=\sum_{i=1}^n\prod_{j=1}^i(1-x_j)$. If both claims hold, then the proposition follows. The first claim can be proved similarly to the characterisation of the Easy FSM instance class in (Lehre and Yao 2007) (see Proposition 1).

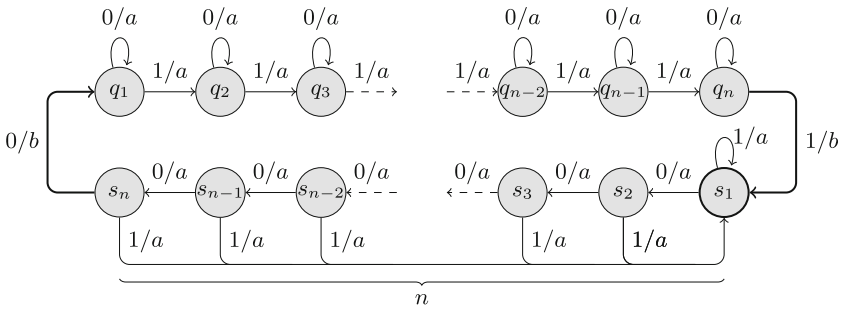

Fig. 2 RIDGE FSM instance class

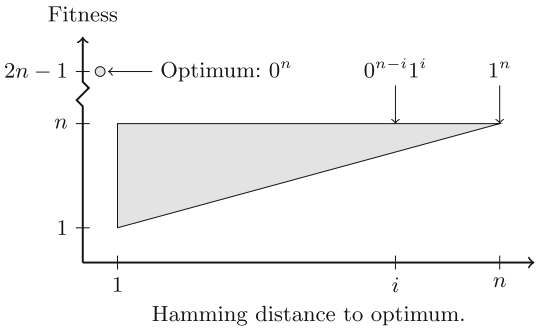

Fig. 3 Illustration of fitness values in $\text{UIO}_{\text{RIDGE},s_1}$

For the second claim, note that all states s_i, where $1 \leq i \leq n$, collapse to state s_1 on input symbol 1. Hence, input symbols subsequent to input symbol 1 will not produce any distinguishing output. Therefore, for any state $s_i, 2 \leq i \leq n$, if $\lambda(s_1, 0^j 1z) \neq \lambda(s_i, 0^j 1z)$, then $\lambda(s_1, 0^j) \neq \lambda(s_i, 0^j)$. The only distinguishing transition that s_i can reach on input 0^j is (s_n, q_1). To reach this transition from state s_i, it is necessary to have at least $n - i$ 0-bits in the input sequence. Hence, on input 0^j, a state s_j has different output from s_1 if and only if $j > n - i$. The number of states $s_i, 2 \leq i \leq n$, with different output from state s_1 on input $0^j 1z$ is therefore j. □

The fitness function associated with the RIDGE FSM is illustrated in Fig. 3. Except for 0^n which is the only UIO of length n for state s_1, the fitness function is the sum of LZ and ONEMAX. In the figure, each search point is mapped to a point in the plane. The horizontal position of a search point is given by the Hamming distance to the optimum, and the vertical position is given by the fitness of the search point. It is clear from the fitness function RIDGE that all non-optimal search points are within the triangle in the figure. The search points $0^i 1^{n-i}, 0 \leq i < n$, have identical fitness, forming a neutral path of length $n - 1$. The runtime analysis for RIDGE is similar to that of SPC in (Jansen and Wegener 2001). When reaching the path, (1 + 1) EA will make a random walk until it hits the global optimum. $(1+1)\text{EA}^*$ will get stuck on the path, only accepting the optimal search point. If the distance to the optimum is large, then it takes long until $(1+1)\text{EA}^*$ mutates the right bits. The function SPC maximises this distance by embedding an explicitly defined trap. In contrast, RIDGE does not have such an explicitly defined trap. Even without the trap, one can prove that $(1+1)\text{EA}^*$ is still likely to reach the path far from the optimum because $(1+1)^*$ EA optimises ONEMAX quicker than LZ.

Lemma 2 *Let $x = 0^i 1\alpha$ and $y = 0^j 1\beta$ be two bitstrings of length n, with $i, j \geq 0$. If there are more 0-bits in substring β than in substring α, then* $\text{RIDGE}(x) > \text{RIDGE}(y)$.

Proof Let bitstrings x and y be on the form $x = 0^i 1\alpha$, and $y = 0^j 1\beta$, where $i, j \geq 0$. Assume substring α contains less 0-bits than substring β. Then

$$n - i - 1 - \text{ONEMAX}(\alpha) < n - j - 1 - \text{ONEMAX}(\beta)$$
$$\text{ONEMAX}(\alpha) > \text{ONEMAX}(\beta) + j - i.$$

So the fitness of search point x is

$$\begin{aligned}\text{RIDGE}(x) &= i + 1 + \text{ONEMAX}(\alpha) \\ &> i + 1 + \text{ONEMAX}(\beta) + j - i \\ &= \text{RIDGE}(y).\end{aligned}$$

□

Theorem 1 *The expected time until (1 + 1) EA finds an UIO of length n for state s_1 in* RIDGE *FSM using fitness function* $\text{UIO}_{\text{RIDGE},s_1}$ *is bounded from above by $O(n^3)$.*

Proof By Proposition 1, any non-optimal search point x can be expressed on the form $x = 0^i 1z$, where i is an integer $0 \leq i < n$, and z is a bitstring of length $n - 1 - i$ that will be referred to as the *suffix* of x. Let j denote the number of 0-bits in the suffix z of the current search point. By Lemma 2, the value of j will never increase. The search process is divided into two phases. The process is in Phase 1 when the suffix z contains at least one 0-bit, i.e. $j > 0$, and the process is in Phase 2 when z does not contain any 0-bit, i.e. $j = 0$.

We first estimate the expected duration of Phase 1. The probability of decreasing the value of j in an iteration is at least the probability of mutating one out of the j 0-bits in the suffix, and none of the remaining bits, i.e. $(j/n) \cdot (1 - 1/n)^{n-1} \geq j/en$, where e is Euler's number. So the expected number of iterations needed to remove the at most $n - 1$ 0-bits and end Phase 1 is $en \sum_{j=1}^{n-1} 1/j = O(n \ln n)$.

We then estimate the expected duration of Phase 2. During this phase, only search points on the form 0^i1^{n-i} will be accepted. Hence, the changing value of i can be considered as a random walk on the integer interval between 0 and n. The optimum is found when the random walk hits the value $i = n$. This process has been analysed by Jansen and Wegener (2001), showing an upper bound of $O(n^3)$ iterations. The expected duration of both phases is therefore $O(n^3)$. □

Theorem 2 *The probability that (1 + 1)* EA has found an UIO of length n for state s_1 in* Ridge *FSM using fitness function* $\text{UIO}_{\text{Ridge},s_1}$ *in less than $n^{(1-\delta)n}$ steps, is bounded from above by* $\exp(-\Omega(n^{1-\epsilon}))$ *for any constants $\delta, \epsilon > 0$.*

Proof The search process is divided into two phases in the same way as in the proof of Theorem 1. We first claim that with probability $1 - \exp(-\Omega(n^{1-\epsilon}))$, the first 1-bit will occur before position $\delta n/2$ when the process enters Phase 2. We say that there is a *failure* in Phase 1 if Phase 2 starts with the first 1-bit after position $\delta n/2$. Following the arguments in Droste et al. (2002) for lower bounding the runtime of (1 + 1) EA on LeadingOnes, it can be shown that there exists a constant c_1 such that the probability that there are more than $\delta n/2$ leading 0-bits in step $c_1 n^2$ is $e^{-\Omega(n)}$. By Lemma 2, the number of 0-bits in the tail will not increase. Hence, following the same arguments as in the proof of Theorem 1, it can be shown that the expected time until all 0-bits are removed in the suffix is less than $(c_2/2)n \ln n$ for some constant c_2. Divide the first $c_1 n^2$ steps into $c_1 n/(c_2 \ln n)$ intervals, each of duration $c_2 n \ln n$ steps. By Markov's inequality (Motwani and Raghavan 1995), the probability that Phase 1 is not finished after one interval is less than $1/2$. Furthermore, the probability that Phase 1 is not finished after $c_1 n/(c_2 \ln n)$ intervals is $2^{-c_1 n/(c_2 \ln n)} = \exp(-\Omega(n^{1-\epsilon}))$ for any constant $\epsilon > 0$.

If there is no failure in Phase 1, then Phase 2 starts with a search point on the form 0^i1^{n-i}, with $i < \delta n/2$. From this point, the selection operator will only accept the optimum, which is the only search point with strictly higher fitness. To reach optimum 0^n from search point 0^i1^{n-i}, it is necessary to flip at least $(1 - \delta/2)n$ 1-bits in one iteration, an event which occurs with probability less than $n^{-(1-\delta/2)n}$. So, by a union bound, in runs without failures, the probability that the optimum is found within $n^{(1-\delta)n}$ iterations is less than $n^{(1-\delta)n} n^{-(1-\delta/2)n} = n^{-\delta n/2} = e^{-\Omega(n)}$.

Hence, the probability that no failure has occurred and the optimum has not been found after $n^{(1-\delta)n}$ steps is at least $1 - \exp(-\Omega(n^{1-\epsilon}))$ for any constant $\epsilon > 0$. □

3 Impact of crossover

Although the (1 + 1) EA is efficient on several instance classes, one can hypothesise that there exist FSMs for which this EA is too simplistic. In particular, when is it necessary to use crossover and a population in computing UIOs?

There exists theoretical evidence that crossover is essential on at least some problems, including several artificial pseudo-boolean functions (Jansen and Wegener 1999, 2002; Storch and Wegener 2004; Dietzfelbinger et al. 2003). Jansen and Wegener (1999, 2002) were among the first to rigorously analyse the impact of crossover, showing that the operator can reduce the runtime exponentially on a variant of the Jump problem (Droste et al. 2002). However, a crossover event between the right types of individuals on this problem is only likely to occur when the population is sufficiently diverse. To guarantee population diversity, they assumed an artificially low crossover probability. For the Ising model, a small runtime gap was proven for rings (Fischer and Wegener 2005), and an exponential runtime gap was proven for trees (Sudholt 2005). Oliveto et al. (2008) proved that crossover is essential on an instance class of the vertex cover problem, but this result depends again on an artificially low crossover probability. Doerr et al. (2008) studied the all pairs shortest path problem. Under certain restrictions, they showed that introducing a problem-specific crossover operator can reduce the expected runtime from $\Theta(n^4)$ to $O(n^{3.5+\varepsilon})$. Through an improved analysis, Doerr and Theile have recently improved this result, showing that an EA using crossover solves the problem in time $O(n^{3.25} \log^{1/4} n)$ with high probability (Doerr and Theile 2009). Although this runtime gap is small, the result is important because it shows for the first time that a crossover operator can be beneficial, not only on specific problem instances, but in general on a practical combinatorial optimisation problem.

We present an instance class of the UIO problem and a steady state EA where crossover decreases the runtime exponentially. Furthermore, the result holds for any constant crossover probability. More specifically, when reducing the crossover probability from any positive constant ($p_c > 0$) to no crossover ($p_c = 0$), the runtime increases exponentially. The proof idea is to construct a fitness function where the individuals in the population can follow two different paths, each leading to a separate local optimum. The local optima are separated by the maximal Hamming distance. The global optimum is positioned in the middle between these local optima and can be efficiently reached with an appropriate one-point crossover between the local optima. The paths are constructed to make it unlikely that mutation alone will produce the global optimum. It is worth noting that our analysis is based on specific types of crossover and mutation.

The definition of the TwoPaths FSM instance class is given in Definition 7, and is parametrised by a constant ϵ that can take any value in the open interval from 0 to 1. The

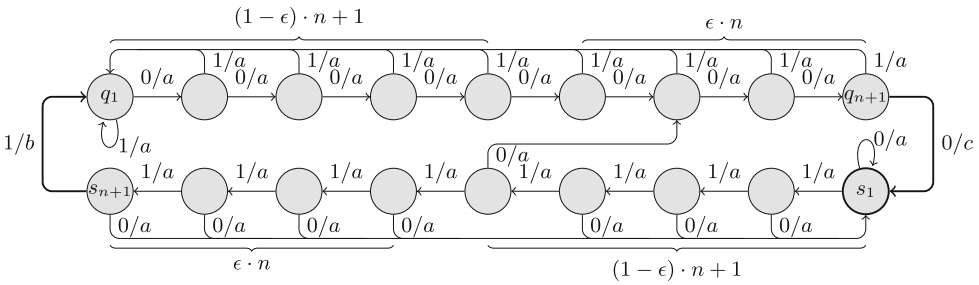

Fig. 4 TwoPaths FSM instance class

TwoPaths FSM instance class has a ternary output alphabet and is illustrated in Fig. 4.

Definition 7 For instance sizes $n \geq 2$ and a constant $\epsilon, 0 < \epsilon < 1$, define a TwoPaths FSM with input and output symbols $I := \{0,1\}$ and $O := \{a,b,c\}$, respectively, and $2(n+1)$ states $S = Q \cup R$, 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 defined as

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

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

The state transition function δ is defined as

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

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

$$\delta(q_i,1) := q_1, \quad \text{and,}$$

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

First, we characterise the function $\text{UIO}_{\text{TwoPaths},s_1}$ associated with the TwoPaths FSM instance class.

Proposition 2 *Let ϵ be any constant $0 < \epsilon < 1$. On input sequences of length n, the function $\text{UIO}_{\text{TwoPaths},s_1}$ takes the following values,*

$$\text{TwoPaths}(x) = \begin{cases} 2n+1 & \text{if } x = x^* \\ \text{Lo}(x)+1 & \text{if } x \in A \setminus \{x^*\} \\ \text{Lo}(x) & \text{if } x_1 = 1 \text{ and } x \notin A \\ \text{Lz}(x)+1 & \text{if } x_1 = 0 \text{ and } \text{Lz}(x) \geq \epsilon n, \\ \text{Lz}(x) & \text{if } x_1 = 0 \text{ and } \text{Lz}(x) < \epsilon n, \end{cases}$$

where

$$x^* := 1^{(1-\epsilon)\cdot n} 0^{\epsilon\cdot n},$$

$$A := \left\{ 1^i 0^{\epsilon n} \alpha \mid \alpha \in \{0,1\}^{(1-\epsilon)n-i} \right\},$$

and

$$\text{Lo}(x) := \sum_{i=1}^{n} \prod_{j=1}^{i} x_j,$$

$$\text{Lz}(x) := \sum_{i=1}^{n} \prod_{j=1}^{i} (1-x_j).$$

Proof Similarly to the proof of Proposition 1, the transitions (s_{n+1}, q_1) and (q_{n+1}, s_1) are called *distinguishing* because they have unique input/output behaviours, whereas all other states output a on any input symbol. Clearly, for any two states s and t and input sequence x, if neither state s nor state t reaches any distinguishing transition on input sequence x, then $\lambda(s,x) = \lambda(t,x) = a^{\ell(x)}$.

Consider first input sequences that have a leading 1-symbol. By arguments similar to those used in the proof of Proposition 1, the number of states in R that reach the distinguishing transition (s_{n+1}, q_1) on input 1^j equals j. Furthermore, all states $q \in Q$ collapse into state q_1 on input 1, therefore none of these states will reach a distinguishing state on any input sequence $1z$ of length n.

We make the claim that any state $s_i \in R$ reaches the distinguishing transition (q_{n+1}, s_1) if and only if the input sequence is on the form $x = 1^{(1-\epsilon)\cdot n+1-i} 0^{\epsilon\cdot n} \alpha$. To prove this claim, consider first input sequences of length n on the form $x = 1^j 0 \alpha$ where $j \neq (1-\epsilon)\cdot n+1-i$. If $0 \leq j < (1-\epsilon)\cdot n+1-i$, then $\delta(s_i, 1^j 0) = s_1$, and from state s_1, it is impossible to reach state q_{n+1} with the remaining bits α which by assumption must be shorter than n. On the other hand, if $j > (1-\epsilon)\cdot n+1-i$, then on input 1^j, we reach a state beyond $s_{(1-\epsilon)\cdot n+1}$ from which the shortest distance to state q_{n+1} is longer than n. Consider next input sequences of length n that are on the form $x = 1^{(1-\epsilon)\cdot n+1-i} 0^j 1 \alpha$ with $0 \leq j < \epsilon\cdot n$, then $\delta(s_i, 1^{(1-\epsilon)\cdot n-i} 0^j 1) = q_1$, and it is impossible to reach state q_{n+1} from state q_1 with the remaining substring α which is shorter than n.

The claim therefore holds, and hence, on input sequence $x^* = 1^{(1-\epsilon)\cdot n} 0^{\epsilon\cdot n}$, only state s_1 among states R reaches the distinguishing transition, and none of the states in Q reaches the distinguishing transition. This implies that this input sequence is a UIO, i.e. the fitness of input sequence

x^* is $2n+1$. For other sequences x with a leading 1-symbol, the number of states with different output than s_1 now equals $\mathrm{Lo}(x)+1$ if $x \in A \setminus \{x^*\}$ and $\mathrm{Lo}(x)$ if $x \notin A$.

Consider then input sequences that have a leading 0-symbol. Again, following the arguments similar to those used in the proof of Proposition 1, the number of states in Q that reach the distinguishing transition (q_{n+1}, s_1) on input $0z$ equals $\mathrm{Lz}(0z)$. Furthermore, by the claim above, the number of states in R with different output than state s_1 on input 0^j is 1 if $j \geq \epsilon n$ and 0 otherwise. Therefore, the total number of states with different output than state s_1 on input $0z$ is $\mathrm{Lz}(0z)+1$ if $\mathrm{Lz}(0z) \geq \epsilon n$ and $\mathrm{Lz}(0z)$ otherwise. □

The problem contains two local optima, 0^n and 1^n, and one global optimum $1^{(1-\epsilon)\cdot n}0^{\epsilon \cdot n}$. If all the individuals reach the same local optimum, then the crossover operator will not be helpful. An essential challenge with the idea behind TwoPaths is therefore to ensure that both local optima are reached. In addition to a large population size, some sort of diversity mechanism might therefore be necessary. Friedrich et al. (2008) have shown that the choice of diversity mechanism can have a major impact on the runtime of EAs. Here, we will consider a steady state GA where population diversity is ensured through the acceptance criteria, as in the *deterministic crowding* diversity mechanism (Friedrich et al. 2008).

Definition 8 ($(\mu+1)$ *SSGA*)

Sample a population P of μ points u.a.r. from $\{0,1\}^n$.
repeat
  with prob. $p_c(n)$,
    Sample x and y u.a.r. from P.
    Sample k u.a.r. from $\{1,...,n\}$.
    $x' := x_1 \cdot x_2 \cdots x_{k-1} \cdot y_k \cdot y_{k+1} \cdots y_n$.
    $y' := y_1 \cdot y_2 \cdots y_{k-1} \cdot x_k \cdot x_{k+1} \cdots x_n$.
    if $\max\{f(x'), f(y')\} \geq \max\{f(x), f(y)\}$
      then $x := x'$ *and* $y := y'$.
  otherwise
    Sample x u.a.r. from P.
    $x' := x$.
    Flip each bit of x' with prob. $1/n$.
    if $f(x') \geq f(x)$
      then $x := x'$.

$(\mu+1)$ SSGA with crossover probability $p_c = 0$ degenerates into μ parallel runs of the $(1+1)$ EA. The algorithm $(\mu+1)$ SSGA is similar to $(\mu+1)$ RLS introduced in (Oliveto et al. 2008), but has a different acceptance criterion. The $(\mu+1)$ RLS accepts both offspring if the best offspring is at least as good as the worst parent, hence allowing the best individual in the population to be replaced with a less fit individual. The $(\mu+1)$ SSGA adopts a more restrictive acceptance criterion, only accepting the offspring if the best offspring is at least as good as the best parent. Each individual in a $(\mu+1)$ SSGA population can be associated with a lineage which interacts little with other lineages, thus facilitating the runtime analysis.

Definition 9 (*SSGA lineage*) Let x be any individual that was added to the population by mutating an individual y. Then individual y is called the *parent* of individual x. Let $z = \alpha_1 \cdot \beta_2$ be any individual that was added to the population by crossover between two individuals $x = \alpha_1 \cdot \alpha_2$ and $y = \beta_1 \cdot \beta_2$. If $\alpha_1 = \beta_1$, then individual y is the parent of individual x, and if $\alpha_1 \neq \beta_1$, then individual x is the parent of individual z. The *lineage* of an individual in the population is the sequence of search point associated with the parent relations.

Definition 10 (TwoPaths *prefix*) Let x be any search point where $x_1 = x_2 = \cdots = x_i$ and $x_i \neq x_{i+1}$ for an integer i where $1 \leq i \leq n$. Then i is the *prefix length* of search point x, the substring $x_1 \cdots x_i$ is called the *prefix* of search point x, and the substring $x_{i+1} \cdots x_n$ is called the *suffix* of search point x.

Lemma 3 *Define the set $A^* := (A \cup B) \setminus \{x^*\}$, where the set A is as in defined in Proposition 2 and the set B is defined as*

$$B := \big\{ b^m \bar{b} \alpha 0^{\epsilon n} \beta \mid b \in \{0,1\}, \\ \alpha, \beta \in \{0,1\}^*, \text{ and} \\ \ell(\alpha) + \ell(\beta) = n - m - 1 - \epsilon n \big\}.$$

Let i be an integer $1 \leq i \leq n$. Let x be a search point with prefix length strictly shorter than i and which is not member of set A^*. Let y be another search point with prefix length at least i. Let x' be the descendant from x, and y' the descendant from y in any crossover between x and y. If $(\mu+1)$ SSGA on TwoPaths *accepted the crossover products x' and y', then $x_i = x'_i$.*

Proof The proof is by contradiction, assuming that $x_i \neq x'_i$ and that the crossover products were accepted. Assume in addition that the crossover point occurred in position k. The lemma trivially holds if $i < k$. So let us assume that $i \geq k$. Defining $j := i - k$, the crossover products can be written as

$$y' = y_1 \cdots y_1 \quad \cdot \overline{y_1} \cdot x_{k+j+1} \cdots x_n, \quad \text{and}$$
$$x' = x_1 \cdots x_{k+j-1} \cdot y_1 \cdot y_{k+j+1} \cdots y_n.$$

The conditions of the lemma imply that $f(y) \geq k+j$. The lemma will be proved by showing that both $f(y') < f(y)$ and

$f(x') < f(y)$ hold, which contradicts that the crossover products were accepted.

Consider first the case where $y_1 = y_2 = \cdots y_{k+j} = 1$. This implies that $x_{k+j} = \overline{y_{k+j}} = 0$. Since $x \notin A^*$, we must also have $y' \notin A^*$, which by Proposition 2 implies that $f(y') = k + j - 1 < f(y)$. To see that also $f(x') < f(y)$ holds, notice that it is impossible that $x_1 = x_2 = \cdots = x_{k+j-1} = 1$, because then by Definition 9, the descendant satisfies $x' = x$ and the lemma would hold. But it is also impossible that $x_1 = x_2 = \cdots = x_{k+j-1} = 0$, because then x would have prefix length at least $k + j = i$, which contradicts with a condition in the lemma. Both search points x and x' therefore have prefix lengths strictly shorter than $k + j - 1$, and by Proposition 2, we then have $f(x') \leq k + j - 1 < f(y)$.

Consider next the case where $y_1 = y_2 = \cdots y_{k+j} = 0$. This implies that $x_{k+j} = \overline{y_{k+j}} = 1$. Proposition 2 implies that $f(y') < f(y)$. To see that also $f(x') < f(y)$, notice that it is impossible that $x_1 = x_2 = \cdots = x_{k+j-1} = 0$, because then by Definition 9, the descendant satisfies $x' = x$ and the lemma would hold. But it is also impossible that $x_1 = x_2 = \cdots = x_{k+j-1} = 1$, because then x would have prefix length at least $k + j = i$, which contradicts with a condition in the lemma. Both search points x and x' therefore have prefix lengths strictly shorter than $k + j - 1$, and by Proposition 2, we then have $f(x') \leq k + j - 1 < f(y)$. □

Lemma 4 *For any generation $t \geq 0$, if no individual until generation t was member of set A^*, and an individual $x(t)$ in generation has prefix length strictly shorter than i, then* $\mathbf{Pr}[x_i(t) = 1] = \mathbf{Pr}[x_i(t) = 0] = 1/2$.

Proof The proof is by induction on generation number t. The lemma obviously holds for generation $t = 0$, hence assume that the lemma also holds for generation $t = k$. If a mutation occurs in generation k, then the parent $x(k)$ must have had prefix length strictly shorter than i, and by the induction hypothesis $\mathbf{Pr}[x_i(k) = 1] = \mathbf{Pr}[x_i(k) = 0]$. We then have

$$\begin{aligned}\mathbf{Pr}[x_i(k+1) = 1] &= \mathbf{Pr}[\text{bit } x_i \text{ mutated}] \cdot \mathbf{Pr}[x_i(k) = 0] \\ &\quad + (1 - \mathbf{Pr}[x_i \text{ mutated}]) \cdot \mathbf{Pr}[x_i(k) = 1] \\ &= \mathbf{Pr}[x_i(k) = 1].\end{aligned}$$

Assume that a crossover between individuals $x(k)$ and $y(k)$ occurs in generation k. If the crossover point was higher than i, then clearly $\mathbf{Pr}[x_i(k+1)] = \mathbf{Pr}[x_i(k)]$. If the crossover point was equal or less than position i, and the corresponding bit $y_i(k)$ was in the suffix of bitstring $y(k)$, then it follows from the induction hypothesis that $\mathbf{Pr}[x_i(k+1) = 1] = \mathbf{Pr}[y_i(k)] = 1/2$. Finally, if the bit $y_i(k)$ occurs in the prefix of bitstring $y(k)$, then by Lemma 3, the crossover occurs only if $x_i(k) = y_i(k)$. Hence,

$$\begin{aligned}\mathbf{Pr}[x_i(k+1) = 1] &= \mathbf{Pr}[x_i(k) = 1 \cap y_i(k) = 1] \\ &\quad + \mathbf{Pr}[x_i(k) = 1 \cap y_i(k) = 0] \\ &= \mathbf{Pr}[x_i(k) = 1].\end{aligned}$$

The lemma now holds for all t by induction. □

Proposition 3 *The probability that any of the initial e^{cn} generations of $(\mu + 1)$ SSGA on* TwoPaths *with population size $\mu = poly(n)$ contains a non-optimal individual in set A^* is exponentially small $e^{-\Omega(n)}$.*

Proof Denote by p_t the probability that there exists an individual in generation t that is member of set A^*, conditional on the event that none of the previous $t - 1$ generations contained such an individual. Then by Lemma 4, the probability that any block of bits of length ϵn in the suffix contains only 0-bits is $2^{-\epsilon n}$. There are at most $O(\mu n)$ such suffix-blocks in the population, hence the probability p_t is bounded by $p_t \leq O(\mu n) \cdot 2^{-\epsilon n} = e^{-\Omega(n)}$ if $\mu = poly(n)$. By union bound, the probability that within e^{cn} generations, there exists such an individual is less than $\sum_{t=0}^{e^{cn}} p_t \leq e^{cn} \cdot e^{-\Omega(n)} = e^{-\Omega(n)}$ for a sufficiently small constant c. □

Lemma 5 *For any lineage, if no individual along the lineage is a member of set A as defined in Proposition 2, then the fitness along the lineage is monotonically increasing.*

Proof The proof is by contradiction, assuming that there exists a lineage where y' is a direct descendant of y, and $f(y') < f(y)$. Search point y' could not have been accepted in a mutation step, hence search y' must have been created through crossover between y and some other search point x, producing the offspring x' and y'. Assume w.l.o.g. that these individuals can be expressed on the form

$$\begin{array}{llll} x := \alpha_1 \cdot \alpha_2 & x' := \alpha_1 \cdot \beta_2 & \ell(\alpha_1) = \ell(\beta_1) \\ y := \beta_1 \cdot \beta_2 & y' := \beta_1 \cdot \alpha_2 & \ell(\alpha_2) = \ell(\beta_2) \end{array}$$

By the definition of a lineage, and the assumption that the offspring were accepted, the following three conditions must hold

$$f(x') \geq f(x) \tag{1}$$

$$f(x') \geq f(y) > f(y'), \text{ and,} \tag{2}$$

$$\alpha_1 \neq \beta_1. \tag{3}$$

The proof is divided into the two main cases $y_1 = 1$ and $y_1 = 0$. These two cases are further divided into the subcases (a) $x_1 = 0$ and (b) $x_1 = 1$. We will show that all these cases lead to a contradiction.

Case 1 Consider the case where $y_1 = 1$. By the condition of the proposition, $y, y' \notin A$, so the fitnesses of the two

individuals y and y' are given by Proposition 2 as $f(y) = \mathrm{Lo}(\beta_1\beta_2)$ and $f(y') = \mathrm{Lo}(\beta_1\alpha_2)$. Hence

$$f(y) > f(y') \tag{4}$$

$$\mathrm{Lo}(\beta_1\beta_2) > \mathrm{Lo}(\beta_1\alpha_2) \tag{5}$$

This inequality is only satisfied when $\mathrm{Lo}(\beta_1) = \ell(\beta_1)$ and $\mathrm{Lo}(\beta_2) \geq 1$, so in both sub-cases, the following two inequalities must hold

$$f(y) \geq \ell(\beta_1) + \mathrm{Lo}(\beta_2) \geq \ell(\beta_1) + 1, \quad \text{and}$$
$$f(y') \geq \ell(\beta_1) + \mathrm{Lo}(\alpha_2).$$

Consider the first sub-case where $x_1 = 0$. Then since $\mathrm{Lo}(\beta_2) \geq 1$, we have $f(x') \leq \mathrm{Lz}(\alpha_1\beta_2) + 1 \leq \mathrm{Lz}(\alpha_1) + 1$. From condition (2), it follows that

$$f(x') > f(y')$$
$$\mathrm{Lz}(\alpha_1) + 1 > \ell(\beta_1) + \mathrm{Lo}(\alpha_2)$$

This inequality is only satisfied when $\mathrm{Lz}(\alpha_1) = \ell(\beta_1)$ and $\mathrm{Lz}(\alpha_2) \geq 1$. Note that since $\mathrm{Lz}(\beta_2) = 0$, we have $\mathrm{Lz}(\alpha_1\beta_2) \geq \epsilon n$ only if also $\mathrm{Lz}(\alpha_1\alpha_2) \geq \epsilon n$. By Proposition 2, the difference between the fitness of individual x and the fitness of individual x' is therefore bounded by

$$\begin{aligned} f(x) - f(x') &\geq \mathrm{Lz}(\alpha_1\alpha_2) - \mathrm{Lz}(\alpha_1\beta_2) \\ &\geq \ell(\beta_1) + \mathrm{Lz}(\alpha_2) - \ell(\beta_1) \\ &\geq 1, \end{aligned}$$

which contradicts with condition (1). Hence, Case 1a leads to a contradiction.

In the second sub-case, where $x_1 = 1$, condition (3) requires that the prefixes of x and y must be different. In other words, the prefix of x must contain at least one 0-bit, so $\mathrm{Lo}(\alpha_1) < \ell(\beta_1) = \mathrm{Lo}(\beta_1)$ by Inequality (5). The fitness of x' in this sub-case is therefore bounded by $f(x') \leq \ell(\beta_1) \leq f(y')$, which contradicts with condition (2). Hence, Case 1b also leads to a contradiction.

Case 2 Consider the case where $y_1 = 0$. In this case, we claim it must hold that

$$\mathrm{Lz}(\beta_1) = \ell(\beta_1), \quad \text{and}$$
$$\mathrm{Lz}(\beta_2) \geq 1.$$

The claim trivially holds in the sub-case where $\mathrm{Lz}(\beta_1) < \epsilon n$ and $\mathrm{Lz}(\beta_1\beta_2) \geq \epsilon n$. In the other sub-cases, where $\mathrm{Lz}(\beta_1) \geq \epsilon n$ or $\mathrm{Lz}(\beta_1\beta_2) < \epsilon n$, condition (2) and Proposition 2 imply that

$$f(y) > f(y')$$
$$\mathrm{Lz}(\beta_1\beta_2) > \mathrm{Lz}(\beta_1\alpha_2),$$

so the claim must also hold.

By the claim and Proposition 2, the fitnesses of individuals y and y' in case 2 therefore satisfy

$$f(y') \geq \mathrm{Lz}(\beta_1\alpha_2) \geq \ell(\beta_1), \quad \text{and}$$
$$f(y) \geq \mathrm{Lz}(\beta_1\beta_2) \geq \ell(\beta_1) + 1.$$

In the first sub-case where $x_1 = 0$, condition (3) requires that the prefixes of x and y must be different. In other words, the prefix of x must contain at least one 1-bit, so $\mathrm{Lz}(\alpha_1) < \mathrm{Lz}(\beta_1) = \ell(\beta_1)$. The fitness of x' in this sub-case is therefore bounded by $f(x') \leq \ell(\beta_1) \leq f(y')$ which contradicts with condition (2). Hence, Case 2a leads to a contradiction.

Consider the second sub-case $x_1 = 1$. Assume first that $x' \in A$, where A is as defined in Proposition 2. It cannot be that $\mathrm{Lo}(\alpha_1) < \ell(\beta_1)$, because this would lead to the following contradiction with condition (2)

$$f(x') = \mathrm{Lo}(\alpha_1\beta_2) + 1 \leq \ell(\beta_1) \leq f(y').$$

Hence, it must hold that $\mathrm{Lo}(\alpha_1) = \ell(\beta_1)$, which by the assumptions $x' \in A$ and $\mathrm{Lz}(\beta_2) \geq 1$ implies that $\mathrm{Lz}(\beta_2) \geq \epsilon n$. However, from this follows another contradiction with condition (2)

$$\begin{aligned} f(x') &= \mathrm{Lo}(\alpha_1\beta_2) + 1 \\ &= \mathrm{Lo}(\alpha_1) + 1 \\ &< \ell(\beta_1) + \epsilon n \leq f(y). \end{aligned}$$

Finally, assuming that $x' \notin A$ would also contradict with condition (2), because

$$f(x') = \mathrm{Lo}(\alpha_1\beta_2) \leq \ell(\beta_1) < f(y).$$

Hence, Case 2b also leads to a contradiction. It has now been shown that all of the sub-cases where $f(y') < f(y)$ lead to a contradiction. From this, one must conclude that $f(y') \geq f(y)$, i.e. the fitness along the lineage is monotonically increasing. □

To show that the population will be relatively evenly distributed between the two local optima, it is sufficient to prove that there is a constant probability that a lineage will always stay on the same path as it started.

Lemma 6 *For $n \geq 4$, and any lineage x, the probability that lineage x reached the local optimum without accepting a search point in which the first bit has been flipped, is bounded from below by $1/12$.*

Proof By Proposition 3, with probability $1 - e^{-\Omega(n)}$, no element of the lineage is member of set $A^* \supset A$ as defined in Proposition 3. Hence, by Lemma 5, the fitness along the lineage is monotonically increasing.

Denote by $v \in \{0, 1\}$ the leading bit in the lineage in the first iteration. Let $p_i, 1 \leq i \leq n$ be the probability that the lineage acquires at least i leading v-bits without accepting a search point where the initial bit has been flipped. We prove by induction on i that probability p_i is bounded from below by

$$p_i \geq \frac{1}{4(1+ei/n)}, \tag{6}$$

which is sufficient to prove the lemma because it implies that $p_n \cdot (1-e^{-\Omega(n)}) > 1/12$.

Inequality (6) clearly holds for $i = 3$, because the probability of getting three identical leading bits in the initial generation is 1/4. Suppose that the inequality also holds for $i = k, 3 \leq k < n$. We show that the inequality must also hold for $i = k+1$. The probability of reaching $k+1$ leading v-bits without flipping the first bit, equals the probability of the event that the lineage acquires k leading v-bits, and then the number of leading v-bits is increased before k leading v-bits are flipped simultaneously.

$$\begin{aligned} p_{k+1} &\geq p_k \cdot \frac{1/en}{1/en + 1/n^k} \\ &\geq \frac{1}{4} \cdot \frac{1}{(1+ek/n)(1+e/n^{k-1})} \\ &\geq \frac{1}{4} \cdot \frac{1}{(1+e(k+1)/n)}, \end{aligned}$$

when $n \geq 4 > e+1$ and $k \geq 3$. By induction, Inequality (6) now holds for all $k, 1 \leq k \leq n-1$. □

The success probability of the $(\mu+1)$ SSGA when using a positive constant crossover probability can now be estimated easily.

Theorem 3 *The probability that the $(\mu+1)$ SSGA with constant crossover rate $p_c > 0$ and population size $\mu \geq 2$ and $\mu = poly(n)$ has found the optimum of* TwoPaths *within $c\mu^2 n^2$ generations is $1 - e^{-\Omega(\mu)} - e^{-\Omega(n)}$ for some constant $c > 0$.*

Proof The optimisation process will be divided into two phases. Phase 1 lasts the first $c\mu n^2$ generations. A failure occurs in Phase 1 if the population does not contain both 1^n and 0^n when the phase ends. Phase 2 lasts for the next $c\mu n^2(\mu-1)$ generations. A failure occurs in Phase 2 if the optimum has not been found by the end of the phase. Clearly, if no failure occurs, then the optimum has been found.

We assume that no individual during Phase 1 is member of the set A^* defined Lemma 3, which by Proposition 3 holds with probability $1 - e^{-\Omega(n)}$. The phase is considered failed if this assumption does not hold. By Lemma 6, for any of the two local optima 0^n or 1^n, the probability that a lineage reaches this optimum is at least $(1/2) \cdot (1/12) = 1/24$. Hence, the probability that both of the two optima will be reached by at least one individual is at least $1 - (1 - 1/24)^\mu = 1 - e^{-\Omega(\mu)}$. We now focus on any pair of lineages that reach different local optima. The probability that a given lineage is selected for mutation in one generation is $(1-p_c) \cdot (1/\mu)$. Hence, by Chernoff bounds (Motwani and Raghavan 1995), the probability that during $(1-p_c) \cdot (2c') \cdot \mu n^2$ generations, the lineage is mutated less than $(1-p_c) \cdot c' n^2$ times is $e^{-\Omega(n)}$. By Lemma 5, the fitness along the lineage is monotonically increasing, and the progress is similar to that of $(1+1)$ EA on the LeadingOnes function. Following the arguments in Droste *et al.* (2002), for a sufficiently large constant c', the probability that the lineage has not reached the local optimum within $(1-p_c) \cdot (c'/2) \cdot \mu n^2$ generations is $e^{-\Omega(n)}$, and the probability that both local optima has not been reached within $c\mu n^2$ generations is $e^{-\Omega(n)}$ for $c := (1-p_c) \cdot c'$. By union bound, the probability of failure during Phase 1 is $e^{-\Omega(\mu)} + e^{-\Omega(n)}$.

Assuming no failure in Phase 1, the probability during each generation in Phase 2 of crossing over two individuals 0^n and 1^n to produce the optimum is at least $p_c/(\mu^2 n)$. Hence, the probability that Phase 2 is not finished within $c\mu n^2(\mu-1)$ generations, i.e. of a failure in Phase 2, is no more than

$$\left(1 - \frac{p_c}{\mu^2 n}\right)^{c\mu n^2(\mu-1)} = e^{-\Omega(n)}.$$

Hence, by union bound, the combined failure probability during both phases is bounded from above by $e^{-\Omega(\mu)} + e^{-\Omega(n)}$. □

It is a common practice in applications of EAs to conduct multiple independent runs of the algorithm, either by launching several parallel runs in a multi-processing environment, or by sequentially restarting the EA in certain time intervals. The following result gives the expected runtime of the $(\mu+1)$ SSGA assuming an appropriately chosen restart strategy that depends both on the problem size n and the population size μ.

Theorem 4 *For a sufficiently large constant $c > 0$, if the $(\mu+1)$ SSGA with constant crossover probability $p_c > 0$ and population size $\mu, 2 \leq \mu = poly(n)$, is restarted every $c\mu^2 n^2$ generations on* TwoPaths, *then the expected time until the optimum is found is $O(\mu^2 n^2)$.*

Proof The expected runtime is no more than the duration of one run, i.e. $c\mu^2 n^2$, multiplied by the expected number of restarts needed to find the optimum. By Theorem 3, for a sufficiently large constant c, the probability of reaching the optimum within one run is $1 - e^{-\Omega(\mu)} - e^{-\Omega(n)}$. For $\mu \geq 2$, the expected number of restarts needed is therefore $1/(1 - e^{-\Omega(\mu)} - e^{-\Omega(n)}) = O(1)$. □

There is a trade-off between the success probability and the duration of each run. According to Theorem 3, the probability of finding the optimum in one run increases with increasing population size. However, the duration of one run, following the restarting strategy in Theorem 4, also increases with population size. The following corollary shows that a constant, small population size is

sufficient to yield a short, quadratic expected runtime when using the restart strategy.

Corollary 1 *For a sufficiently large constant $c > 0$, if the $(\mu + 1)$ SSGA with constant crossover probability $p_c > 0$ and population size $\mu = 2$ is restarted every $4cn^2$ generations on* TwoPaths, *then the expected time until the optimum is found is $O(n^2)$.*

Proof The corollary follows from Theorem 4. □

The previous results show that the $(\mu + 1)$ SSGA can find a UIO for state s_1 on the TwoPaths FSM instance class efficiently, assuming that one applies the crossover operator. Finally, we will analyse the runtime of $(\mu + 1)$ SSGA without the crossover operator, i.e. with crossover probability $p_c = 0$. The proof idea is to focus on a single lineage, since the lineages are independent, and distinguish between two conditions. If the lineage has at least $(1 - \epsilon)n$ leading 0-bits, then all these must be flipped into 1-bits. If there is at least one 1-bit among the first $(1 - \epsilon)n$ bits, then with high probability, a large number of 1-bits must be flipped in the tail of the search point.

Theorem 5 *The probability that the $(\mu + 1)$ SSGA with crossover probability $p_c = 0$ and population size $\mu = poly(n)$ finds the optimum of* TwoPaths *within 2^{cn} generations is bounded from above by $e^{-\Omega(n)}$, where c is a constant.*

Proof With crossover probability $p_c = 0$, the population is only updated by mutations and the algorithm is essentially μ parallel runs of (1 + 1) EA. Consider any lineage, and divide the current search point x into an $(1 - \epsilon)n$ bits long prefix, and an ϵn bits long suffix v, such that $x = u \cdot v$. The global optimum has prefix $1^{(1-\epsilon)n}$ and suffix $0^{\epsilon n}$.

If the run at some point reaches a search point with prefix $u = 0^{(1-\epsilon)n}$, then subsequent search points are only accepted if they have prefix $0^{(1-\epsilon)n}$ or $1^{(1-\epsilon)n}$. The probability of reaching the optimum in any such iteration is therefore bounded above by $n^{-(1-\epsilon)n}$, and the success probability within e^{cn} iterations is by union bound no more than $n^{-(1-\epsilon)n} \cdot e^{cn} = e^{-\Omega(n)}$.

For runs where the current search point has at least one 1-bit in the prefix, we will use the simplified drift theorem (see the appendix) to bound the time until the suffix contains only 0-bits. Let the state $i \in \{0, \ldots, N\}$ be the number of 1-bits in the suffix, with $N := \epsilon n$. Furthermore, define $a := 0$ and $b := \epsilon n/10$. To derive a lower bound, we optimistically assume that any bit-flip from 1 to 0 in the suffix is accepted, an assumption which can only speed up the process. The remaining part of the analysis is now practically identical to the analysis of (1 + 1) EA on Needle in (Oliveto and Witt 2008). Assuming $a < i < b$, the expected drift in the process is

$$\begin{aligned}\mathbf{E}[\Delta(i)] &= \frac{\epsilon n - i}{n} - \frac{i}{n} \\ &= \frac{\epsilon n - 2i}{n} \\ &\geq (4/5)\epsilon,\end{aligned}$$

and condition 1 of the drift theorem holds with $\beta = (4/5)\epsilon$. In order to decrease the number of 1-bits in the suffix by j, it is necessary to flip j 1-bits simultaneously, an event which happens with probability

$$\binom{(1-\epsilon)\cdot n}{j} \cdot n^{-j} \leq \frac{1}{j!} \leq 2^{-j+1},$$

so condition 2 of the theorem holds with $\delta = r = 1$. Hence, the probability that a given lineage reaches the optimum within 2^{cn} iterations, is bounded from above by $e^{-\Omega(n)}$, for some constant c. Finally, the probability that any lineage reaches the optimum within 2^{cn} generations, is bounded from above by $\mu \cdot e^{-\Omega(n)}$. □

It is easy to see that when not using crossover, the $(\mu + 1)$ SSGA is highly unsuccessful on the TwoPaths problem, even when using the restarting strategy.

Corollary 2 *The probability that in at least one of $2^{c'n}$ runs, each run lasting 2^{cn} generations, $(\mu + 1)$ SSGA with crossover probability $p_c = 0$ and population size $\mu = poly(n)$ finds the optimum of* TwoPaths, *is $e^{-\Omega(n)}$, for some constants c' and c.*

Proof Let c be a constant that satisfies the conditions of Theorem 5. Then by Theorem 5 and a union bound, the probability that the optimum has been found within $2^{c'n}$ restarts is no more than $2^{c'n} \cdot e^{-\Omega(n)} = e^{-\Omega(n)}$ for a sufficiently small constant c'. □

Here, we have studied a problem characteristic where crossover is beneficial. The crossover operator allows the $(\mu + 1)$ SSGA to overcome a large gap in the search space by recombining two local optima into the global optimum. In contrast, the probability of overcoming a gap with bitwise mutation is exponentially small with respect to the gap width. Although the analysis only considers a specific instance class of the UIO problem, similar results can be envisaged on instance classes that are constructed similarly to the TwoPaths FSM. The TwoPaths FSM can be decomposed into two smaller FSMs, corresponding to the set of states R and Q. It is easy for the (1 + 1) EA to construct the UIO 0^n for state s_1, when only taking into account the set of states Q. Similarly, the UIO 1^n for state s_1 can easily be constructed when only taking into account the states in R. The crossover operator becomes useful when constructing an UIO for the combined FSM. Similar characteristics may be obtained by combining other sub-components that individually are easy for the (1 + 1) EA.

4 Conclusion

This paper has investigated the impact of the acceptance criterion in (1 + 1) EA and the crossover operator in $(\mu+1)$ SSGA when computing UIOs from FSMs. The objective is to identify simple, archetypical cases where these EA parameter settings have a particularly strong effect on the runtime of the algorithm. The results are obtained by first characterising the fitness landscapes induced by the UIO problem instance classes (in Proposition 1 and Proposition 2), and then by analysing the behaviour of the EAs on these fitness landscapes. This analysis can therefore be transferred to other problem domains than FSM testing when the corresponding fitness landscapes have similar characteristics to those studied here. Such vigorous theoretical analysis is essential in gaining insight into fundamental issues in applying meta-heuristic algorithms to software engineering problems. It can shed light on when to use (or not to use) what kind of meta-heuristic algorithms (e.g., EAs). It also helps to characterise a problem (e.g., UIO in FSM testing) so that a deeper understanding of the problem can be obtained.

The first part of the paper describes the Ridge FSM instance class which induces a search space with a neutral path of equally fit search points. Runtime analysis shows that the variant of (1 + 1) EA which only accepts strictly better search points will get stuck on the path, while the standard (1 + 1) EA which also accepts equally fit search points will find the UIO in polynomial time. This result shows how apparently minor modification in EAs can have an exponentially large runtime impact when computing UIOs.

The second part of the paper considers the impact of crossover when computing UIOs with the $(\mu+1)$ SSGA. The result shows that on the TwoPaths FSM instance class, the SSGA finds the UIO in polynomial time with high probability as long as the crossover probability is any constant $p_c > 0$. For single runs of the $(\mu+1)$ SSGA, the probability of finding the UIO increases with increasing population size, but a larger population size also imposes an overhead in terms of increased runtime. However, it is shown that by using an appropriate restart strategy, the UIO can be found in expected quadratic time with a small population size. On the other hand, with crossover probability $p_c = 0$, the runtime of $(\mu+1)$ SSGA increases exponentially, even when using a restart strategy. This result means that when computing UIOs, the crossover operator can be essential, and simple EAs including the (1 + 1) EA can be inefficient. This result is important because although crossover is assumed important in EAs, few theoretical results on non-artificial problem domains have confirmed that this is the case.

Acknowledgments The authors would like to thank Pietro Oliveto for useful comments. This work was supported by EPSRC under grant no. EP/D052785/1.

Appendix

The appendix states a result obtained elsewhere which was used and cited in runtime analysis of EAs in this paper.

Theorem 6 [Simple drift theorem (Oliveto and Witt 2008)] *Let $X_t, t \geq 0$, be the random variables describing a Markov process over the state space $S := \{0, 1, \ldots, N\}$, and denote $\Delta_t(i) := (X_{t+1} - X_t \mid X_t = i)$ for $i \in S$ and $t \geq 0$. Suppose there exists an interval $[a, b]$ of the state space and three constants $\beta, \delta, r > 0$ such that for all $t \geq 0$*

1. $\mathbf{E}[\Delta_t(i)] \geq \beta$ *for* $a < i < b$, *and*
2. $\mathbf{Pr}[\Delta_t(i) = -j] \leq 1/(1+\delta)^{j-r}$ *for* $i > a$ *and* $j \geq 1$, *then there is a constant* $c^* > 0$ *such that for*

$$T^* := \min\{t \geq 0 : X_t \leq a \mid X_0 \geq b\}$$

it holds that $\mathbf{Pr}\left[T^* \leq 2^{c^*(b-a)}\right] = 2^{-\Omega(b-a)}$.

References

Clarke J, Dolado JJ, Harman M, Hierons R, Jones B, Lumkin M, Mitchell B, Rees K, Roper M, Shepperd MJ (2003) Reformulating software engineering as a search problem. IEE Proc Softw 150(3):161–175

Cormen TH, Leiserson CE, Rivest RL, Stein C (2001) Introduction to algorithms, 2nd edn. McGraw Hill, New York

Derderian KA, Hierons RM, Harman M, Guo Q (2006) Automated unique input output sequence generation for conformance testing of fsms. Comput J 49(3):331–344

Dietzfelbinger M, Naudts B, Van Hoyweghen C, Wegener I (2003) The analysis of a recombinative hill-climber on h-iff. IEEE Trans Evol Comput 7(5):417–423

Doerr B, Happ E, Klein C (2008) Crossover can provably be useful in evolutionary computation. In: Proceedings of the 10th annual conference on genetic and evolutionary computation (GECCO' 2008), New York, NY, USA. ACM, pp 539–546

Doerr B, Theile M (2009) Improved analysis methods for crossover-based algorithms. In: Proceedings of the 11th annual conference on Genetic and evolutionary computation (GECCO'2009), New York, NY, USA, ACM, pp 247–254

Droste S, Jansen T, Wegener I (2002) On the analysis of the (1 + 1) evolutionary algorithm. Theor Comput Sci 276:51–81

Fischer S, Wegener I (2005) The one-dimensional Ising model: Mutation versus recombination. Theor Comput Sci 344(2–3): 208–225

Friedrich T, Oliveto PS, Sudholt D, Witt C (2008) Theoretical analysis of diversity mechanisms for global exploration. In: Proceedings of the 10th annual conference on Genetic and evolutionary computation (GECCO'2008), New York, NY, USA, ACM, pp 945–952

Guo Q, Hierons RM, Harman M, Derderian KA (2004) Computing unique input/output sequences using genetic algorithms. In: Proceedings of the 3rd international workshop on formal

approaches to testing of software (FATES'2003), vol 2931 of *LNCS*, pp 164–177

Guo Q, Hierons RM, Harman M, Derderian KA (2005) Constructing multiple unique input/output sequences using metaheuristic optimisation techniques. IEE Proc Softw 152(3):127–140

He J, Yao X (2004) A study of drift analysis for estimating computation time of evolutionary algorithms. Nat Comput 3(1):21–35

Jansen T, Wegener I (1999) On the analysis of evolutionary algorithms—a proof that crossover really can help. In: Proceedings of 7th annual European symposium on algorithms (ESA'99), vol 1643 of *LNCS*, pp 700

Jansen T, Wegener I (2001) Evolutionary algorithms—how to cope with plateaus of constant fitness and when to reject strings of the same fitness. IEEE Trans Evol Comput 5(6):589–599

Jansen T, Wegener I (2002) The analysis of evolutionary algorithms—a proof that crossover really can help. Algorithmica 34(1):47–66

Lee D, Yannakakis M (1996) Principles and methods of testing finite state machines-a survey. Proc IEEE 84(8):1090–1123

Lehre PK, Yao X (2007) Runtime analysis of (1 + 1) EA on computing unique input output sequences. In: Proceedings of 2007 IEEE congress on evolutionary computation (CEC'2007). IEEE Press, pp 1882–1889

Lehre PK, Yao X (2008) Crossover can be constructive when computing unique input output sequences. In: Proceedings of the 7th international conference on simulated evolution and learning (SEAL'2008). Springer, Heidelberg, pp 595–604

Motwani R, Raghavan P (1995) Randomized algorithms. Cambridge University Press, Cambridge

Oliveto PS, He J, Yao X (2008) Analysis of population-based evolutionary algorithms for the vertex cover problem. In: Proceedings of IEEE world congress on computational intelligence (WCCI'2008), Hong Kong, June 1–6, 2008, pp 1563–1570

Oliveto PS, Witt C (2008) Simplified drift analysis for proving lower bounds in evolutionary computation. In: Proceedings of parallel problem solving from nature (PPSN'X), number 5199 in LNCS, pp 82–91

Storch T, Wegener I (2004) Real royal road functions for constant population size. Theor Comput Sci 320(1):123–134

Sudholt D (2005) Crossover is provably essential for the Ising model on trees. In: Proceedings of the genetic and evolutionary computation conference (GECCO'2005), pp 1161