Crossover can be constructive when computing unique input output sequences*

Per Kristian Lehre and Xin 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, United Kingdom
{P.K.Lehre,X.Yao}@cs.bham.ac.uk

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 (μ+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.

1 Introduction

Evolutionary Algorithms (EAs) are general purpose optimisation algorithms. In principle, they can 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.

*This is a preprint of a paper that will appear in the Proceedings of the 7th International Conference on Simulated Evolution and Learning (SEAL'08).

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 better equipped in making an informed decision 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 finite state machines (FSMs), a computationally hard problem from the software engineering domain [1]. UIOs have important applications in conformance testing of FSMs [2]. Similarly to other approaches in *search based software engineering* [3], the UIO problem has been reformulated as an optimisation problem and tackled with EAs [4, 5]. Experimental results show that EAs can construct UIOs efficiently on some instances. Guo *et al.* compared an evolutionary approach with a random search strategy, and found that the two approaches have similar performance on a small FSM, while the evolutionary approach outperforms random search on a larger FSM [5]. Derderian *et al.* presented an alternative evolutionary approach [4], confirming Guo *et al's* results.

Theoretical results confirm that EAs can outperform random search on the UIO problem [1]. 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 [1]. The UIO problem is NP-hard [2], 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 [1]. To reliably apply EAs to the UIO problem, it is necessary to distinguish easy from hard instances. 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 $\Theta(n^k)$ [1].

Do EA settings have any significant impact on the chance of finding UIOs efficiently? Guo *et al* hypothesise that crossover is helpful [6], without giving further evidence than two example sequences that recombine into a UIO. The theoretical results in this paper confirms this hypothesis, proving that crossover can be essential for finding the 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 Sections 2 and 3.

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 an input symbol a, the machine makes the transition from its current state s to a next state $\delta(s, a)$ and outputs symbol $\lambda(s, a)$. 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, $t \neq s$.*

To compute UIOs with EAs, candidate solutions are represented as strings over the input alphabet of the FSM, which is here restricted to $I = \{0,1\}$ [5]. Although the shortest UIOs in the general case can be exponentially long with respect to the number of states [2], all the instances 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 [5, 1].

Definition 3 (UIO fitness function)**.** *For a finite state machine M with m states, the 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* $\gamma_M(s,x) := |\{t \in S \mid \lambda(s,x) = \lambda(t,x)\}|$.

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 $\mathrm{UIO}_{M,s}$ has an optimum of $m-1$. 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. Lemma 1 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 [7, 8])**.** *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 $T_{A,\mathcal{F}}(n)$, including the *expected runtime* $\mathbf{E}[T_{A,\mathcal{F}}(n)]$ and the *success probability* $\mathbf{Pr}[T_{A,\mathcal{F}}(n) \leq t(n)]$ within time bound $t(n)$.

2 Impact of Acceptance Criterion

The (1+1) EA is a simple single-individual algorithm, which in each iteration replaces the current search point x by a new search point x' if and only if $f(x') \geq$

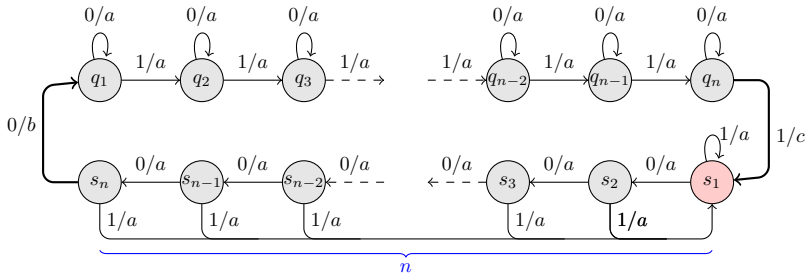

Figure 1: RIDGE FSM instance class.

$f(x)$. 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 [9]. Here, it is shown that the same result holds on the UIO problem for the RIDGEFSM 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 *$f(x') \geq f(x)$,* **then** *$x := x'$.*

Definition 6 (RIDGE FSM)**.** *For instance sizes n, $n \geq 2$, 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 $Q := \{q_1, q_2, ..., q_n\}$ and $R := \{s_1, s_2, ..., s_n\}$. For all states q_i and s_i, $1 \leq i \leq n$, define the transition and output functions as $\delta(q_i, 0) := q_i$, $\delta(s_i, 1) := s_1$, $\lambda(q_i, 0) := a$, $\lambda(s_i, 1) := a$, and*

$$\delta(q_i,1) := \begin{cases} s_1 & \text{if } i = n, \\ q_{i+1} & \text{otherwise .} \end{cases} \qquad \delta(s_i,0) := \begin{cases} q_1 & \text{if } i = n, \\ s_{i+1} & \text{otherwise.} \end{cases}$$

$$\lambda(q_i,1) := \begin{cases} b & \text{if } i = n, \\ a & \text{otherwise.} \end{cases} \qquad \lambda(s_i,0) := \begin{cases} b & \text{if } i = n, \\ a & \text{otherwise.} \end{cases}$$

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*

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

Proof. We claim that on inputs x of length n and different from 0^n, the number of states, among the states q_i, $1 \leq i \leq n$, with different output than state s_1 equals $\textsc{OneMax}(x) := \sum_{i=1}^{n} x_i$ and the number of states, among the states s_i, $2 \leq i \leq n$, with different output than state s_1 equals $\textsc{Lz}\ (x) := \sum_{i=1}^{n} \prod_{j=1}^{i}(1 - x_i)$. The first claim follows from the characterisation of the easy FSM instance class in [1] (see Proposition 1). All states s_i, $1 \leq i \leq n$, collapse to state s_1 on input 1. Hence, for a state s_i, $2 \leq i \leq n$, if $\lambda(s_1, 0^j 1z) \neq \lambda(s_i, 0^j 1z)$, then $\lambda(s_1, 0^j) \neq \lambda(s_i, 0^j)$. To reach transition (s_n, q_1) 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 j. □

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 . 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 [9]. When reaching the path, (1+1) EA will make a random walk until it hits the global optimum. (1+1) 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) 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) EA* is still likely to reach the path far from the optimum because (1+1)* EA optimises Onemax quicker than Lz . The formal proof of this, and some of the following theorems have been omitted due to space limitations. (A complete version of this paper containing all the proofs is available as a technical report [10].)

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. Theorem 1 Non-optimal search points are on the form $0^i 1z$, where i is an integer $0 \leq i < n$ and z is some bitstring of length $n - 1 - i$. We divide the search process into two phases. The process is in Phase 1 if the suffix z contains at least one 0-bit, and the process is in Phase 2 when z does not contain any 0-bit. By Proposition 1, the process will never return to Phase 1 once Phase 2 has been entered.

Let j denote the number of 0-bits in the tail. By Lemma 4, the value of j will never increase. The probability of decreasing the value of j in an iteration is at least j/en. So the expected time to remove the at most $n - 1$ 0-bits and end Phase 1 is $O(n \ln n)$.

In Phase 2, only search points on the form $0^i 1^{n-i}$ 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 [9], showing an upper bound of $O(n^3)$ iterations. □

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* $\mathrm{UIO}_{\mathrm{RIDGE},s_1}$ *in less than $n^{n/24}$ steps, is bounded from above by $e^{-\Omega(n)}$.*

Proof. Theorem 2 The search process is divided into two phases in the same way as in the proof of Theorem 1. We claim that with probability $1 - e^{-\Omega(n)}$, when the process enters Phase 2, the first 1-bit will occur before position $11n/12$. This can be proved by adapting an argument in Droste et al [7] for lower bounding the runtime of (1+1) EA on LO (leading ones) . In order to reduce the number 1-bits, it is necessary, but in our case not always sufficient, to flip the leftmost 1-bit. Following the terminology in [7], a step in which the left-most 1-bit flips is called *essential*. An essential step can decrease the number of 1-bits by more than one if the 1-bit is followed by one or more 0-bits. Such consecutive blocks of 0-bits are called *free riders*.

We consider a time interval of $n^2/12$ iterations and define four types of failures. *Failure 1* occurs when the process is still in Phase 1 after $n^2/12$ iterations. *Failure 2* occurs when there have been more than $n/6$ essential steps in Phase 1. *Failure 3* occurs when there are more than $2n/3$ free-riders during $n/6$ essential steps. *Failure 4* occurs when more than $n/4$ leftmost 1-bits are flipped during $n/6$ essential steps.

Failure 1: By Lemma 4, the number of 0-bits in the tail will not increase. The probability of flipping a given 0-bit is at least $1/en$, so the probability that a given 0-bit has not been flipped after $n^2/12$ iterations is no more than $(1 - 1/en)^{n^2/12} \leq e^{-n/12e}$. Hence, the probability that at least one of the at most $n - 1$ 0-bits in the tail has not been flipped after $n^2/12$ iterations is no more than $(n-1) \cdot e^{-n/12e} = e^{-\Omega(n)}$.

Failure 2: The probability of having an essential step in a given iteration is less than $1/n$, so the expected number of essentials steps in $n^2/12$ iterations is less than $n/12$. So the probability of having at least $n/6$ essential steps is bounded (using a Chernoff bound) from above by $e^{-n/36} = e^{-\Omega(n)}$.

Failure 3: The total number of free-riders in Phase 1 cannot be larger than the number of 0-bits in the suffix. The initial search search point is a uniformly sampled bitstring, and Lemma 4 guarantees that the number of 0-bits in the tail will not increase. Hence, the probability that the tail contains more than $2n/3$ free-riders is bounded from above by the probability that the initial bitstring contains more than $2n/3$ 0-bits, which by a Chernoff bound is $e^{-\Omega(n)}$.

Failure 4: An essential step can reduce the number of left-most 1-bits by flipping more than one 1-bit simultaneously. Let random variables $X_t \in \{0, .., n\}, t \geq 0$ denote the number of initial 1-bits which are flipped in the t'th essential step. The probability of flipping the $1 + i$ leftmost 1-bits in any essential step is less than n^{-i}, hence we can bound the expectation of each variable X_t with

$$\mathbf{E}[X_t] \leq 1 + \sum_{i=0}^{n} i \cdot n^{-i} \leq 1 + \sum_{i=0}^{\infty} i \cdot n^{-i} = 1 + \frac{1}{n - 2 + 1/n} \leq 1 + \frac{1}{n-2}.$$

In $m := n/6$ essential steps, we now have less than $X := \sum_{t=1}^{n/6} X_t$ leftmost 1-bit flips. For instance sizes $n > 7$, the expectation of X is $\mathbf{E}[X] =$

$\sum_{t=1}^{n/6} \mathbf{E}[X_t] \leq n/6 \cdot (1 + 1/(n-2)) \leq n/5$. Hence, by Theorem 5, the probability that more than $n/5 + n/20 = n/4$ 1-bits are flipped in $n/6$ essential steps is less than $\mathbf{Pr}[X \geq \mathbf{E}[X] + n/20] \leq \exp\left(-2mn^2/400n^2\right) = e^{-\Omega(n)}$. The total failure probability is bounded from above by $e^{-\Omega(n)}$.

If there are no failures in Phase 1, the number of leading 0-bits when Phase 1 ends is less than $n/4 + 2n/3 < 11n/12$, and Phase 2 starts with a search point on the form $0^i 1^{n-i}$, with $i < 11n/12$. From this point, the selection operator will only accept the optimum, which is the only search point with higher fitness. To reach optimum 0^n from search point $0^i 1^{n-i}$, it is necessary to flip at least $n/12$ 1-bits in an iteration, an event which occurs with probability less than $1/n^{n/12}$. So in runs without failures, the probability that the optimum is found within $n^{n/24}$ iterations is less than $1/n^{n/24} = e^{-\Omega(n)}$.

Hence, the probability that a failure has not occurred and the optimum has not been found after $n^{n/24}$ steps is at least $(1 - e^{-\Omega(n)}) \cdot (1 - e^{-\Omega(n)}) = 1 - e^{-\Omega(n)}$. □

3 Impact of Crossover

Although (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 [11, 12]. For the Ising model, a small runtime gap was proven for rings [13], and an exponential runtime gap was proven for trees [14]. Recently, crossover was proven essential on a vertex cover instance [15], but this result depends on an artificially low crossover probability. We present an instance class of the UIO problem and a steady state genetic algorithm where crossover is provably essential. 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.

Definition 7. *For instance sizes n, $n \geq 2$ 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 $R := \{s_1, s_2, ..., s_{n+1}\}$ and*

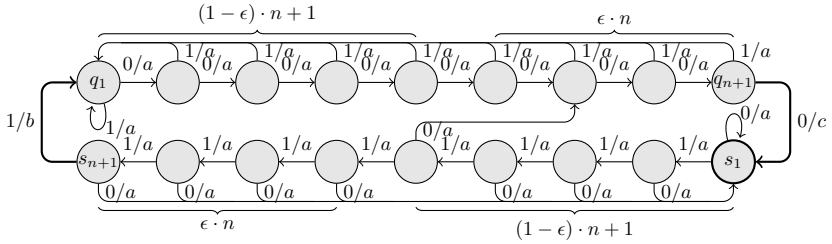

Figure 2: TwoPaths FSM instance class.

$Q := \{q_1, q_2, ..., q_{n+1}\}$. *The output and transition functions are defined as*

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

$$\delta(s_i, 0) := \begin{cases} q_{(1-\epsilon)\cdot n+3} & \text{if } i = \epsilon \cdot n + 1, \\ s_1 & \text{otherwise.} \end{cases} \qquad \delta(s_i, 1) := \begin{cases} q_1 & \text{if } i = n+1, \\ s_{i+1} & \text{otherwise.} \end{cases}$$

$$\delta(q_i, 1) := q_1, \quad \text{and,} \qquad \delta(q_i, 0) := \begin{cases} s_1 & \text{if } i = n+1, \text{ and} \\ q_{i+1} & \text{otherwise.} \end{cases}$$

Proposition 2. *Let ϵ be any constant $0 < \epsilon < 1$. On input sequences of length n, the fitness function $\mathrm{UIO}_{\textsc{TwoPaths},s_1}$ takes the following values, where $A = \{1^i 0^{\epsilon n} \alpha \mid \alpha \in \{0,1\}^{(1-\epsilon)n-i}\}$,*

$$\textsc{TwoPaths}\ (x) = \begin{cases} 2n+1 & \text{if } x = x^* \\ \textsc{Lo}\ (x) + 1 & \text{if } x \in A \setminus \{x^*\} \\ \textsc{Lo}\ (x) & \text{if } x_1 = 1 \text{ and } x \notin A \\ \textsc{Lz}\ (x) + 1 & \text{if } x_1 = 0 \text{ and } \textsc{Lz}\ (x) \geq \epsilon n, \\ \textsc{Lz}\ (x) & \text{if } x_1 = 0 \text{ and } \textsc{Lz}\ (x) < \epsilon n, \end{cases} \qquad \begin{aligned} \textsc{Lo}(x) &:= \sum_{i=1}^{n} \prod_{j=1}^{i} x_j, \\ \textsc{Lz}(x) &:= \sum_{i=1}^{n} \prod_{j=1}^{i} (1 - x_j) \\ x^* &:= 1^{(1-\epsilon)\cdot n} 0^{\epsilon \cdot n}. \end{aligned}$$

Proof. The states s_{n+1} and q_{n+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 reach any distinguishing state on input sequence x, then $\lambda(s, x) = \lambda(t, x) = a^{\ell(x)}$.

On input sequences x of length n, we first 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 $x = 1^{(1-\epsilon)\cdot n+1-i} 0^{\epsilon \cdot n} \alpha$. 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 on the form $x = 1^{(1-\epsilon)\cdot n+1-i}0^j1\alpha$ with $0 \leq j < \epsilon \cdot n$, then $\delta(s_i, 1^{(1-\epsilon)\cdot n-i}0^j1) = q_1$, and it is impossible to reach state q_{n+1} from state q_1 with the remaining substring α which is shorter than n. Our first claim 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 and TwoPaths $(1^{(1-\epsilon)\cdot n}0^{\epsilon\cdot n}) = 2n + 1$.

We secondly claim that $\lambda(s_1, 0^j1z) = \lambda(q_i, 0^j1z)$ if and only if $\lambda(s_1, 0^j) = \lambda(q_i, 0^j)$ for any state $q_i \in Q$, and $\ell(z) = n-j-1$ and $1 \leq j \leq n-1$. ($\Longrightarrow$) The assumption $\lambda(s_1, 0^j1z) = \lambda(q_i, 0^j1z)$ implies $\lambda(s_1, 0^j) = \lambda(q_i, 0^j)$ by Lemma 1. ($\Longleftarrow$) The assumption $\lambda(q_i, 0^j) = \lambda(s_1, 0^j) = a^j$ implies that $\delta(q_i, 0^j1) = q_1$. Neither state q_1 nor state $\delta(s_1, 0^j1) = s_2$ reach any of the distinguishing states on input z, hence $\lambda(s_2, z) = \lambda(q_1, z)$, and $\lambda(s_1, 0^j1z) = \lambda(q_i, 0^j1z)$.

On input 0^j, a state $q_i \in Q$ has different output from state s_1 if and only if $j > n + 1 - i$. Hence, on input sequences 0^j1z, the number of states in Q with different output than state s_1 equals $j =$ Lz (0^j1z). Furthermore, by the first claim, the number of states in R with different output than state s_1 on input 0^j1z is at most 1. Therefore Lz $(0z) \leq$ TwoPaths $(0z) \leq$ Lz $(0z) + 1$. On input symbol 1, all states $q \in Q$ collapse into state q_1, therefore none of these states will reach a distinguishing state on any input sequence $1z \neq x^*$ of length n. Hence, using a similar argument as for input sequences $0z$ above, we have Lo $(1z) \leq$ TwoPaths $(1z) \leq$ Lo $(1z) + 1$, which completes the proof. □

If all individuals reach the same local optimum, then the crossover operator will not be helpful. An essential challenge with the idea behind TwoPaths is to ensure that both local optima are reached. In addition to a large population size, some sort of diversity mechanism might be helpful. Here, we will consider a steady state GA where population diversity is ensured through the acceptance criteria.

Definition 8 ((μ+1) SSGA)**.**

Sample a population P of μ points u.a.r. from $\{0,1\}^n$.
repeat
  with probability $p_c(n)$,
    Sample x and y u.a.r. from P.
    $(x', y') :=$ *one point crossover*(x, y).
    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.
    $x' :=$ *Mutate*(x).
    if $f(x') \geq f(x)$
      then $x := x'$.

(μ+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 [15], 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)**.** *If x was added to the population by mutating y, then y is the* parent *of x. If $z = \alpha_1 \cdot \beta_2$ was added to the population via crossover between $x = \alpha_1 \cdot \alpha_2$ and $y = \beta_1 \cdot \beta_2$, then y is the parent of z if $\alpha_1 = \beta_1$, otherwise x is the parent of z. The* lineage *of an individual in the population, is the sequence of search point associated with the parent relations.*

Definition 10 (TwoPaths suffix)**.** *If a search point $x = x_1 \cdots x_i x_{i+1} \cdots x_n$ satisfies $x_1 = x_2 = \cdots = x_i$ and $x_i \neq x_{i+1}$, then the substring $x_{i+1} \cdots x_n$ is called the* suffix *of search point x.*

Proposition 3. *The probability that any of the initial e^{cn} generations of $(\mu+1)$ SSGA on* TwoPaths *contain a non-optimal individual with the bitstring $0^{\epsilon n}$ in its suffix is exponentially small $e^{-\Omega(n)}$.*

Proof. Proposition 3 Denote by P_t the probability that there exists an individual in generation t with $0^{\epsilon n}$in its suffix, conditional on the event that none of the previous $t-1$ generations contained such an individual. Then by Lemma 6, 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 2. *As long as no individual in the population has a suffix containing substring $0^{\epsilon n}$, the fitness along any lineage of SSGA on* TwoPaths *is monotonically increasing.*

Proof. Lemma 2 Suppose otherwise, that there are parents x and y, and offspring x' and y' on the form

$$\begin{aligned} x &= \alpha_1 \cdot \alpha_2, & x' &= \alpha_1 \cdot \beta_2 \\ y &= \beta_1 \cdot \beta_2, & y' &= \beta_1 \cdot \alpha_2, \end{aligned}$$

such that the offspring are accepted, i.e. one of the offspring x' has fitness

$$\text{TwoPaths}\ (x') \geq \text{TwoPaths}\ (x), \text{ and} \tag{1}$$
$$\text{TwoPaths}\ (x') \geq \text{TwoPaths}\ (y). \tag{2}$$

Without loss of generality, we assume that prefix β_1 begins with a 1-bit. If the fitness along the lineage decreases, we must have

$$\textsc{TwoPaths}\ (y') < \textsc{TwoPaths}\ (y) \tag{3}$$

$$\textsc{Lo}\ (\beta_1 \cdot \alpha_2) < \textsc{Lo}\ (\beta_1 \cdot \beta_2), \tag{4}$$

which is only possible if prefix β_1 contains 1-bits only, and

$$\textsc{Lo}\ (\alpha_2) < \textsc{Lo}\ (\beta_2). \tag{5}$$

Hence, prefix α_1 contains only 0-bits, otherwise $\textsc{TwoPaths}\ (x') < \textsc{TwoPaths}\ (y')$. But, by Ineq. (5), suffix β_2 must have at least one leading 1-bit, which implies the following contradiction with Ineq. (2)

$$\begin{aligned}\textsc{TwoPaths}\ (x') &= \textsc{Lz}\ (\alpha_1 \cdot \beta_2) = \textsc{Lz}\ (\alpha_1) = \textsc{Lo}\ (\beta_1) \\ &\leq \textsc{TwoPaths}\ (y') < \textsc{TwoPaths}\ (y).\end{aligned}$$

□

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 one the same path as it started.

Lemma 3. *For $n \geq 4$, and any lineage x, let t be the generation at which x reaches a local optimum. If no individual until generation t has $0^{\epsilon n}$ in its suffix, then 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. Lemma 3 Denote by $b \in \{0, 1\}$ 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 b-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 suffices for proving the lemma because $i \leq n$.

Inequality (6) clearly holds for $i = 3$, because the probability of getting three identical leading bits in the initial generation is $1/4$. Suppose 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 b-bits without flipping the first bit, equals the probability of the event that the lineage acquires k leading b-bits, and then the number of leading b-bits is increased before k leading b-bits are flipped simultaneously.

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

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

Theorem 3. *The expected runtime of (μ+1) SSGA with a constant crossover probability $p_c > 0$ on* TwoPaths *is $O(n^2\mu \log \mu + \exp(n \ln n - \mu/96))$.*

Proof. The process is divided into two phases. Phase 1 begins with the initial population and ends when all individuals have reached either 0^n or 1^n. Phase 2 lasts until the optimum is found. *Phase 1*: We consider a *failure* in phase 1 to occur if at any time during phase 1, there exists an individual with a suffix containing the string $0^{\epsilon n}$. Assume that a failure does not occur. Let ℓ be the lowest fitness in the population, and i the number of individuals with fitness ℓ. In order to decrease the number of individuals with fitness ℓ, it suffices to make a mutation step, select one among i individuals with fitness ℓ, and flip none but the left-most 1-bit (or 0-bit), an event which happens with probability at least $(1 - p_c)\cdot(i/\mu)\cdot(1/n)\cdot(1-1/n)^{n-1} \geq (1-p_c)i/e\mu n$. By Lemma 2, the fitness does not decrease along any lineage. Hence, the expected time until the entire population has reached either 0^n or 1^n is bounded from above by $\sum_{\ell=1}^{n-1}\sum_{i=1}^{\mu} e\mu n/i(1 - p_c) = O(n^2\mu \log \mu/(1 - p_c))$. By Proposition 3, the failure probability during phase 1 is $e^{-\Omega(n)}$. If a failure occurs, then the number of leading 1-bits can potentially be reduced. Assume pessimistically that Lo (x) + Lo $(x) = 1$ in any lineage x, i.e. the phase restarts. The expected duration of phase 1, then becomes $O(n^2\mu \log \mu/(1 - p_c))/(1 - e^{-\Omega(n)}) = O(n^2\mu \log \mu/(1 - p_c))$. *Phase 2:* We consider a *failure* to occur in phase 2, if the phase starts with less than $\mu/64$ individuals on the local optimum with fewest individuals. By Lemma 3, the probability that any lineage has a leading 1-bit (or 0-bit) and never changes path before reaching a local optimum is at least 1/12. Hence, by Chernoff bounds, the probability that the population contains less than $\mu/24$ individuals which starts with a 1-bit (or 0-bit) and does not change path is bounded from above by $e^{-\mu/96}$. Assuming no failure, the probability of making a crossover step, select two parent individuals 1^n and 0^n, and then making a crossover at point ϵn in any generation in Phase 2 is at least $p_c(1/24)(23/24)/n$. Hence, the expected duration of Phase 2, assuming no failure is $O(n/p_c)$. If a failure occurs in Phase 2, the optimum can be generated from any search point by mutating at most n bits in any individual, an event which happens in less than n^n expected time.

The unconditional expected duration of Phase 1 and Phase 2 is therefore bounded by $O(n^2\mu \log \mu/(1-p_c)+n/p_c+e^{-\mu/96}\cdot n^n/(1-p_c)) = O(n^2\mu \log \mu/(1-p_c) + n/p_c + \exp(n \ln(n/(1 - p_c)) - \mu/96))$. □

Finally, we state the runtime 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 ϵn leading 0-bits, then all these must be flipped into 1-bits. If there is at least one 1-bit among the first ϵn bits, then with high probability, a large number of 1-bits must be flipped in the tail of the search point.

Theorem 4. *The probability that (μ+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. Theorem 4 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 ϵn bits long *prefix*, and an $(1-\epsilon)n$ bits long *suffix* v, such that $x = u \cdot v$. The global optimum has prefix $1^{\epsilon n}$ and suffix $0^{(1-\epsilon)n}$.

If the run at some point reaches a search point with prefix $u = 0^{\epsilon n}$, then subsequent search points are only accepted if they have prefix $0^{\epsilon n}$ or $1^{\epsilon n}$. The probability of reaching the optimum in any such iteration is therefore bounded above by $n^{-\epsilon n}$, and the success probability within e^{cn} iterations is bounded above by $n^{-\epsilon n} \cdot e^{cn} = e^{-\Omega(n)}$.

For runs where the current search point has at least one 1-bit in the prefix, we will use the simplified drift theorem (Theorem 2 in [16]) to bound the time until the suffix contains only 0-bits. Let the state $i \in \{0, ..., N\}$ be the number of 1-bits in the suffix, with $N := (1-\epsilon)n$. Furthermore, define $a := 0$ and $b := (1-\epsilon)n/10$. 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 [16]. Assuming $a < i < b$, the expected drift in the process is $\mathbf{E}[\Delta(i)] = ((1-\epsilon)n - i)/n - i/n \geq (4/5)(1-\epsilon)$, and condition 1 of the drift theorem holds with $\beta = (4/5)(1-\epsilon)$. 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 $\binom{(1-\epsilon)n}{j} n^{-j} \leq 1/j! \leq 2^{-j+1}$, 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 $\mu \cdot e^{-\Omega(n)}$. □

4 Conclusion

This paper has investigated the impact of the acceptance criterion in (1+1) EA and the crossover operator in (μ+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 *first part* 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 that apparently minor modification of an algorithm can have an exponentially large runtime impact when computing UIOs. The *second part* considers the impact of crossover when computing UIOs with the (μ+1) SSGA. The result shows that on the TWOPATHS FSM instance class, the SSGA finds the UIO in polynomial time as long as the crossover probability is a non-zero constant and the population is sufficiently large. However, with crossover probability 0, the runtime of (μ+1) SSGA increases exponentially. This result means that when comput-

ing UIOs, the crossover operator can be essential, and simple EAs including the (1+1) EA can be inefficient. This result is important because although the crossover operator is often thought to be important for GAs, there exist very few theoretical results in non-artificial problem domains confirming that this is the case.

Acknowledgements

The authors would like to thank Pietro Oliveto for useful comments. This work was supported by EPSRC under grant no. EP/C520696/1.

References

[1] Lehre, P.K., Yao, X.: Runtime analysis of (1+1) EA on computing unique input output sequences. In: Proceedings of 2007 IEEE Congress on Evolutionary Computation (CEC'07). (2007) 1882–1889

[2] Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines-a survey. Proceedings of the IEEE **84**(8) (1996) 1090–1123

[3] Clark, J.A., Dolado, J.J., Harman, M., Hierons, R.M., Jones, B., Lumkin, M., Mitchell, B., Mancoridis, S., Rees, K., Roper, M., Shepperd, M.: Reformulating software engineering as a search problem. IEE Proceedings-Software **150**(3) (2003) 161–175

[4] Derderian, K.A., Hierons, R.M., Harman, M., Guo, Q.: Automated unique input output sequence generation for conformance testing of fsms. The Computer Journal **49**(3) (2006) 331–344

[5] Guo, Q., Hierons, R.M., Harman, M., Derderian, K.A.: Computing unique input/output sequences using genetic algorithms. In: Proceedings of the 3rd International Workshop on Formal Approaches to Testing of Software (FATES'2003). Volume 2931 of LNCS. (2004) 164–177

[6] Guo, Q., Hierons, R.M., Harman, M., Derderian, K.A.: Constructing multiple unique input/output sequences using metaheuristic optimisation techniques. IEE Proceedings Software **152**(3) (2005) 127–140

[7] Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. Theoretical Computer Science **276** (2002) 51–81

[8] He, J., Yao, X.: A study of drift analysis for estimating computation time of evolutionary algorithms. Natural Computing **3**(1) (2004) 21–35

[9] Jansen, T., Wegener, I.: Evolutionary algorithms - how to cope with plateaus of constant fitness and when to reject strings of the same fitness. IEEE Transactions on Evolutionary Computation **5**(6) (2001) 589–599

[10] Lehre, P.K., Yao, X.: Crossover can be constructive when computing unique input output sequences. Technical Report (forthcoming), University of Birmingham, School of Computer Science (2008)

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

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

[13] Fischer, S., Wegener, I.: The one-dimensional ising model: Mutation versus recombination. Theoretical Computer Science **344**(2-3) (2005) 208–225

[14] Sudholt, D.: Crossover is provably essential for the ising model on trees. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2005). (2005) 1161–1167

[15] Oliveto, P., He, J., Yao, X.: Analysis of population-based evolutionary algorithms for the vertex cover problem. In: Proceedings of the IEEE World Congress on Computational Intelligence (WCCI'08), Hong Kong, June 1-6, 2008. (2008)

[16] Oliveto, P., Witt, C.: Simplified drift analysis for proving lower bounds in evolutionary computation. Technical Report Reihe CI, No. CI-247/08, SFB 531, Technische Universität Dortmund, Germany (2008)

[17] Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association **58**(301) (1963) 13–30

Lemma 4. *Let $x = 0^i1\alpha$ and $y = 0^j1\beta$ 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* $\textsc{Ridge}(x) > \textsc{Ridge}(y)$.

Proof. Let bitstrings x and y be on the form $x = 0^i1\alpha$, and $y = 0^j1\beta$, where $i, j \geq 0$. Assume substring α contains less 0-bits than substring β. Then

$$\begin{aligned} n - i - 1 - \textsc{OneMax}(\alpha) &< n - j - 1 - \textsc{OneMax}(\beta) \\ \textsc{OneMax}(\alpha) &> \textsc{OneMax}(\beta) + j - i. \end{aligned}$$

So the fitness of search point x is

$$\begin{aligned} \textsc{Ridge}(x) &= i + 1 + \textsc{OneMax}(\alpha) \\ &> i + 1 + \textsc{OneMax}(\beta) + j - i = \textsc{Ridge}(y). \end{aligned}$$

□

Lemma 5. *Let x and y be two search points that do not contain $0^{\epsilon n}$ in their suffices, and that bit x_i is in the suffix of x, and bit y_i is in the prefix y. If a crossover product $\{x', y'\}$ between x and y is accepted on* TwoPaths *, then $x_i = x'_i$.*

Proof. The lemma trivially holds if the crossover point was higher than i. Let $x = \alpha_1\alpha_2$ and $y = \beta_1\beta_2$ and $x' = \alpha_1\beta_2$ and $y' = \beta_1\alpha_2$ and. The lemma obviously holds for $\alpha_1 = \beta_1$, so assume that $\alpha_1 \neq beta_1$. Since bit y_i is in the prefix of y, we have $y_1 = y_2 = \cdots = y_i$, and $f(y) \geq i > f(x)$. Assume by contradiction that $x_i \neq y_i$, then $f(y') < i$ and if the crossover product was accepted, it is necessary that $f(x') \geq i$, which implies that $x'_1 = x'_2 = \cdots = y_i$. However, this contradicts with the assumption that $\alpha_1 \neq \beta_1$. □

Lemma 6. *For any $t \geq 0$, if bit $x_i(t)$ belongs to the suffix of individual $x(t)$ in generation t of (μ+1) SSGA on* TwoPaths *, and the suffix of individual $x(t)$ does not contain $0^{\epsilon n}$, 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

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

Assume 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 by the induction hypothesis $\mathbf{Pr}[x_i(k+1) = 1] = \mathbf{Pr}[y_i(k)] = 1/2$. Finally, if the bit $y_i(k)$ occurs in the

prefix of bitstring $y(k)$, then by Lemma 5, the crossover occurs only if $x_i(k) = y_i(k)$. Hence,

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

The lemma now holds for all generations t by induction. □

5 Additional results

This appendix describes results obtained elsewhere which were used and cited in runtime analysis of EAs in this paper.

Theorem 5 (Hoeffding [17])**.** *If $X = \sum_{i=1}^{n} X_i$ where $X_1, ..., X_n$ are independent random variables with $a_i \leq X_i \leq b_i$ for $1 \leq i \leq n$, then for $t > 0$*

$$\mathbf{Pr}\left[X \geq \mathbf{E}\left[X\right] + t\right] \leq \exp\left(-\frac{2n^2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right).$$

Theorem 6 (Simplified Drift Theorem [16])**.** *Let X_t, $t \geq 0$, be the random variables describing a Markov process over the state space $S := \{0, 1, ..., N\}$, and denote $\Delta(i) := (X_{t+1} - X_t \mid X_t = i)$ 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. *$\mathbf{E}\left[\Delta(i)\right] \geq \beta$ for $a < i < b$, and*
2. *$\mathbf{Pr}\left[\Delta(i) = -j\right] \leq 1/(1+\delta)^{j-r}$ 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 $\mathbf{Pr}\left[T^* \leq 2^{c^*(b-a)}\right] = 2^{-\Omega(b-a)}$.