Runtime Analysis of (1+1) EA on Computing Unique Input Output Sequences

Per Kristian Lehre and Xin Yao

Abstract— **Computing unique input output (UIO) sequences is a fundamental and hard problem in conformance testing of finite state machines (FSM). Previous experimental research has shown that evolutionary algorithms (EAs) can be applied successfully to find UIOs on some instances. However, before EAs can be recommended as a practical technique for computing UIOs, it is necessary to better understand the potential and limitations of these algorithms on this problem. In particular, more research is needed in determining for what instances of the problem EAs are feasible.**

This paper presents a rigorous runtime analysis of the (1+1) EA on three classes of instances of this problem. First, it is shown that there are instances where the EA is efficient, while random testing fails completely. Secondly, an instance class that is difficult for both random testing and the EA is presented. Finally, a parametrised instance class with tunable difficulty is presented. Together, these results provide a first theoretical characterisation of the potential and limitations of the (1+1) EA on the problem of computing UIOs.

I. Introduction

As modern software systems grow larger and more complex, there is an increasing need to support the software engineer with better techniques. The field of *search based software engineering* (SBSE) approaches this challenge in a novel way by reformulating software engineering problems into optimisation problems. Such a reformulation allows the problems to be tackled with evolutionary algorithms and other randomised search heuristics [1].

One domain in which this approach has been taken is in conformance testing of finite state machines. This problem consists of checking whether an implementation machine is equivalent with a specification machine. While one has full information about the specification machine, the implementation machine is given as a black box. To check the implementation machine for faults, one is restricted to input a sequence of symbols and observe the outputs the machine produces. A fundamental problem which one will be faced with when trying to come up with such checking sequences is the *state verification problem*, which is to assess whether the implementation machine starts in a given state [10]. One way of solving the state verification problem is by finding a unique input output sequence (UIO) for that state. A UIO for a state is an input sequence which, when started in this state, causes the FSM to produce an output sequence which is unique for that state.

Computing UIOs is hard. All known algorithms for this problem have exponential runtime with respect to the number of states. Lee and Yannakakis proved that the decision problem of determining whether a given state has a UIO or not is PSPACE-complete, and hence also NP-hard [9]. In the general case, it is therefore unlikely that there will ever be an efficient method for constructing UIOs and one cannot hope to do much better than random search or exhaustive enumeration. The application of evolutionary algorithms or any other randomised search heuristic cannot change this situation. However, the existence of hard instances does not rule out the possibility that there are many interesting instances that can be solved efficiently with the right choice of algorithm. On such "easy" instances, EAs can potentially be more efficient than exhaustive enumeration and random search.

Guo *et al.* reformulated the problem of computing UIOs into an optimisation problem to which he applied an EA [7]. When comparing this approach with random search for UIOs, it was found that the two approaches have similar performance on a small FSM, while the evolutionary approach outperforms random search on a larger FSM. Derderian *et al.* presented an alternative evolutionary approach which also allows the specification machine to be partially specified [3]. Their approach was compared with random search on a set of real-world FSMs and on a set of randomly generated FSMs. Again, it was found that the evolutionary approach outperformed random search on large FSMs. Furthermore, the difference in performance increased with the size of the FSM.

Although previous experimental research have show that there are instances of the problem where the evolutionary approach is preferable over a simple random search strategy, more research is needed to get a deeper understanding of the potential of EAs for computing UIOs. Such a deeper insight can only be obtained if the experimental research is complemented with theoretical investigations. However, rigorous theoretical investigations of search heuristics have been lacking, not only on the problem of computing UIOs, but in the field of search based software engineering in general. An example of another area within SBSE in which theoretical characterisation of problem difficulty is needed is in test data generation. This paper responds to the need for theoretical research in SBSE. Using rigorous runtime analysis, the paper gives a first characterisation of some types of FSMs where computing UIOs is provably easy for an EA, and some types of FSMs where computing UIOs is hard for an EA.

Runtime analysis of EAs is difficult. When initiating the analysis in a new problem domain, it is an important first step to analyse a simple algorithm like the (1+1) EA. Without understanding the behaviour of such a simple algorithm in the new domain, it is difficult to understand the behaviour of more complex EAs. Furthermore, it is necessary to understand the behaviour of a single individual-based EA to understand

Per Kristian Lehre and Xin Yao are with 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 Email: {pkl,xin}@cs.bham.ac.uk.

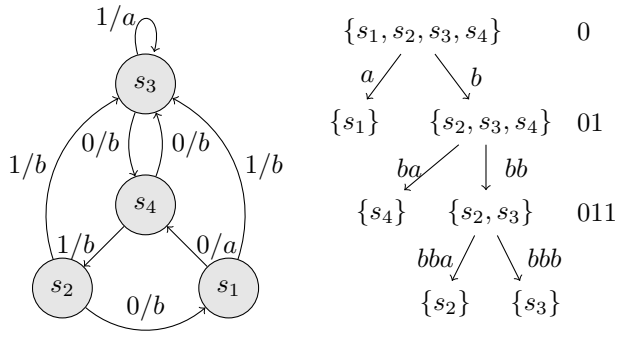

Fig. 1. FSM with corresponding state partition tree for input sequence 011.

the role of populations. We therefore focus on the single-individual based (1+1) EA in this paper.

II. Preliminaries

A. Notation

Symbol ϵ denotes the empty string. The length of a string x is denoted $\ell(x)$. Concatenation of strings x and y is denoted $x \cdot y$, and x^i denotes i concatenations of x. Standard notation (e. g., O, Ω and Θ) for asymptotic growth of functions (see, e. g., [2]) is used in the analysis.

B. Finite State Machines

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.
At any point in time, an FSM M is in exactly one state s in S. When receiving an input a from I, the machine outputs symbol $\lambda(s, a)$ and goes to state $\delta(s, a)$. The domain of the state transition function δ and the output function λ is generalised to non-empty strings over the input alphabet, i. e. $\delta(s, a_1 a_2 \cdots a_n) := \delta(\delta(s, a_1 a_2 \cdots a_{n-1}), a_n)$ and $\lambda(s, a_1 a_2 \cdots a_n) := \lambda(s, a_1) \cdot \lambda(\delta(s, a_1), a_2 \cdots a_n)$.

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 of M such that $\lambda(s, x) \neq \lambda(t, x)$ for all states t, $t \neq s$.

Def. 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 sequence 011 is a UIO for state s_3, because only from state s_3 will the FSM output the sequence bbb. The single input symbol 1 is also a UIO for state s_3, because only state s_3 has output a on input 1.

III. UIO generation as an optimisation problem

A. Representation and Fitness Function

Following [7], candidate solutions are represented as strings over the input alphabet of the FSM. To allow a straightforward application of the (1+1) EA, we consider only instances with the binary input alphabet $I = \{0, 1\}$. Although the shortest UIOs in the general case can be exponentially long with respect to the number of states n [9], all the instances presented here have UIOs of length n. In the interest of simplifying the theoretical analysis, we therefore restrict the search to input sequences of a fixed length n. As the length of the UIO is not an objective, we do not consider "no care symbols" which have been applied in previous experimental work [7]. Because search points represent input sequences, these two terms will be used interchangeably in the rest of the paper.

The objective in this paper is to search for an UIO for a specified state. Minimising the length of the UIO is not considered an objective in this work. As in [7], the fitness of an input sequence is defined as a function of the *state partition tree* induced by the input sequence. Intuitively, the state partition tree of an input sequence represents how increasingly long prefixes of the input sequence partitions the set of states according to the output they produce.

Fig. 1 (right) gives an example of a state partition tree for input sequence 011 on the FSM in Fig. 1 (left). The root node is the set of all nodes. On the first input symbol 0, state s_1 outputs symbol a, while states s_2, s_3 and s_4 output symbol b. The two partitions $\{s_1\}$ and $\{s_2, s_3, s_4\}$ are divided into three partitions on the following input symbol 1. On input 01, state s_1 outputs ab, state s_4 outputs ba, while states s_2 and s_3 output bb. Each singleton $\{s_i\}$ in a state partition tree indicates that the corresponding input sequence is a UIO for that state s_i. Because we are only looking for UIOs for one particular state s_1, we will use the cardinality of the leaf node containing state s_1 when defining the fitness of an input sequence. This approach is a variant of the approach taken in [7] where one searches for UIOs for several states in single runs.

Definition *3 (Fitness function):* For a finite state machine M with n states, a fitness function $f_{M,s} : I^n \to \mathbb{R}$ can now be defined as follows :

$$f_{M,s}(x) := n - \gamma_M(s, x), \quad \text{where}$$
$$\gamma_M(s, x) := |\{t \in S \mid \lambda(s, x) = \lambda(t, x)\}| .$$

The *instance size* of a fitness function $f_{M,s}$ is defined as the number of states n in FSM M. 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 n. If the shortest UIO for state s in FSM M has length no more than n, then $f_{M,s}$ has an optimum of $n-1$. As an example of Def. 3, consider the FSM in Fig. 1, for which the fitness function takes the values $f_{M,s_1}(0111) = 3$ and $f_{M,s_1}(1111) = 2$. In all the instances presented here, the objective is to find a UIO for state s_1. To simplify notation, the notation f_M will therefore be used instead of $f_{M,s}$, and the notation $\gamma(x)$ will be used instead of $\gamma_M(s, x)$, where the FSM M is given by the context.

B. Evolutionary Algorithms

This paper analyses a simple EA called (1+1) EA.

Definition *4 ((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** $x := x'$.

We say that one *step* of the (1+1) EA is one iteration of the *Repeat*-loop in the algorithm. In each step of (1+1) EA, the fitness value $f(x')$ must be evaluated. We can assume that the fitness value $f(x)$ of the current search point x is stored in a local variable. Hence, after step t of the algorithm, the fitness function has been evaluated t times. In the *black box scenario*, the runtime complexity of a randomised search heuristic is measured in terms of the number of evaluations of the fitness function, and not in terms of the number of internal operations in the algorithm [5].

Definition *5 (Runtime [4], [8]):* Given a function $f_n : \{0,1\}^n \to \mathbb{R}$, the runtime T_n of the (1+1) EA on f_n is defined as the number of evaluations of function f_n until the search point x attains the maximal value of f_n for the first time.

A function is considered *easy* for the (1+1) EA if the expected value of runtime T_n is bounded from above by a polynomial in n. Conversely, a function is considered *hard* for the (1+1) EA if the expected value of runtime T_n is bounded from below by an exponential function in n.

IV. Runtime analysis

This section presents three classes of finite state machines. The objective in all classes is to find a UIO for state s_1. The runtime analyses are carried out in two steps. In the first step, the values of the fitness function f_M are derived from the finite state machine M according to Def. 3. In the second step, the runtime of the algorithms are analysed on function f_M.

A. Easy instances

Our aim is to construct a class of instances which is hard for random search, while being easy for the (1+1) EA. In order to be hard for random search, the length of the shortest UIO for state s_1 must be at least linear in n, and there must be few UIOs of this length. To keep the instance class easy for the (1+1) EA, the idea is to ensure that the resulting fitness function has few interactions among the variables. It is well known that the (1+1) EA optimises all linear functions efficiently [8], [4].

Definition *6 (Easy instance class):* For instance sizes n, $n \geq 2$, define an FSM E with input and output symbols $I := \{0,1\}$ and $O := \{a,b\}$ respectively, and n states $S := \{s_1, s_2, ..., s_n\}$. For all states s_i, define the output function λ as

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

and for all states s_i, define the state transition function δ as

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

The objective is to find an UIO of length n for state s_1.
The instances in Def. 6 are illustrated in Fig. 2. This FSM resembles a classical *counter*, which counts the number of 1-symbols, and outputs the special symbol b after n inputs of symbol 1. Note that $(s_n, s_1, 1/b)$ is the only state transition with a distinguishing input/output behaviour. Furthermore, the states will never collapse, i.e. $\delta(s_i,x) \neq \delta(s_j,x)$ for any input sequence x and any pair of different states s_i and s_j. It is easy to see that any sequence of length n containing at most one 0 is a UIO for state s_1. We show that the easy instance class leads to a fitness function which is very similar to the well known fitness function OneMax [12].

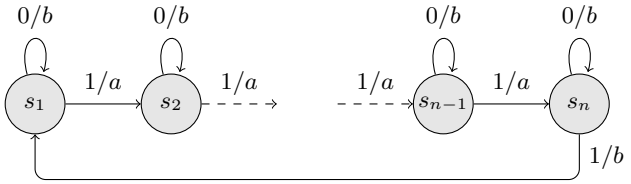

Fig. 2. Finding UIOs for state s_1 is easy with the (1+1) EA.

Proposition *1:* The fitness function f_E corresponding to the instance class in Def. 6 takes the following values

$$f_E(x) = \begin{cases} n-1 & \text{if } x = 1^n, \text{ and} \\ \sum_{i=1}^n x_i & \text{otherwise.} \end{cases}$$

Proof: The case where $\sum_{i=1}^n x_i \geq n-1$ is easy. State s_1 is the only state which outputs a on each of the first $n-1$ inputs of symbol 1. Hence, for such input sequences, $\gamma(x) = 1$.

Before showing that the proposition also holds for the remaining input sequences, we first show that for any input sequence x with $\gamma(x) > 1$, and any single input symbol p,

$$\gamma(x) = \gamma(x \cdot p) + p. \tag{1}$$

Eq. (1) obviously holds when symbol p is 0 because all states output symbol b on input symbol 0, so it remains to show that the equation also holds when symbol p is 1.

By the definition of the transition function, there must be a state t such that $\delta(t,x) = s_n$. Furthermore, we can show that state s_1 and state t produce the same output on input sequence x. Suppose not, that $\lambda(s_1,x) \neq \lambda(t,x)$. This would imply that on input x, state t must have reached the only distinguishing transition from state s_n to state s_1, i.e. sequence x can be expressed on the form $x = y1z$ with $\delta(t,y) = s_n$. Since both $\delta(t,y)$ and $\delta(t,y1z)$ equal state s_n, we must have $\sum_{i=1}^{\ell(z)} z_i \geq n-1$. However, this is a contradiction, because the assumption $\gamma(x) > 1$ implies that $\sum_{i=1}^n x_i < n-1$. It is thus clear that $\lambda(s_1,x) = \lambda(t,x)$, and furthermore $\lambda(s_1, x\cdot 1) \neq \lambda(t, x\cdot 1)$. For all other states s_i different than state t, $\lambda(\delta(s_i,x),1) = \lambda(\delta(s_1,x),1) = a$. So to conclude, if $\gamma(x) > 1$ then $\gamma(x) = \gamma(x\cdot 1) + 1$.

We can now show that the proposition also holds for input sequences where $\sum_{i=1}^n x_i < n-1$. On such input sequences, state s_2 cannot reach the distinguishing state transition from s_n to s_1. So state s_1 and s_2 are indistinguishable and $\gamma(x) > 1$. Obviously, the same also holds for all prefixes of input sequence x. Eq. (1) can now be applied recursively, and by noting the special case of $\gamma(\epsilon) = n$ on the empty string, we obtain the desired result. $\gamma(x_1 \cdots x_n) = \gamma(x_1 \cdots x_{n-2} x_{n-1}) - x_n = \gamma(x_1 \cdots x_{n-2}) - x_{n-1} - x_n = n - \sum_{i=1}^n x_i$. ∎

Theorem *1:* Using fitness function f_E, the probability that random search will find a UIO for state s_1 in less than $e^{c\cdot n}$ iterations is exponentially small $e^{-\Omega(n)}$, where c is a small constant.

Proof: An optimal solution has at most one 0-bit. Hence, the probability that a uniformly sampled sequence is optimal is $(n+1)\cdot 2^{-n}$, which is less than $e^{-n/4}$ for n larger than 4.

The probability that random search finds an optimal solution within $e^{c\cdot n}$, $n \geq 4$, steps is thus no more than

$$\sum_{i=1}^{\exp(c\cdot n)} \left(1-(n+1)\cdot 2^{-n}\right)^i \cdot (n+1)\cdot 2^{-n} \leq \sum_{i=1}^{\exp(c\cdot n)} (n+1)\cdot 2^{-n} \leq e^{c\cdot n}\cdot e^{-n/4} = e^{-\Omega(n)},$$

when $c < 1/4$. ∎

The runtime analysis of (1+1) EA on the problem of computing a UIO for state s_1 follows the well-known analysis of the (1+1) EA on the ONEMAX problem [12]. It is included here for completeness.

Theorem *2:* Using fitness function f_E, (1+1) EA will find a UIO for state s_1 in expected time $O(n \ln n)$.

Proof: By the values of fitness function f_E, in non-optimal search points x and y, $f_E(x) \geq f_E(y)$ if and only if search point x has at least as many 1-bits as search point y. So in a given step of (1+1) EA, the mutated search point x' will only be accepted if it has at least as many 1-bits as search point x. If x' has more 1-bits than x, we say that the step is successful. When x has i 0-bits, the probability of a successful step is at least $i/n \cdot (1-1/n)^{n-1} \geq i/en$.

Search points with at least $n-1$ 1-bits are optimal, hence it suffices to wait for $n-1$ successful steps to find the optimum. The expected runtime of (1+1) EA is therefore bounded above by $\sum_{i=2}^{n} en/i = O(n \cdot \ln n)$

∎

B. Hard instances

The idea behind the hard instance class is to make the resulting fitness function a large plateau. This can be achieved by constructing a finite state machine where the state partition tree gives little information about the UIO.

Definition *7 (Hard instance class):* For instance sizes $n \geq 2$, define an FSM H with input and output symbols $I := \{0,1\}$ and $O := \{a,b\}$, and states $S := \{s_1, s_2, ..., s_n\}$. Furthermore, for all states s_i, define the output function λ as

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

For all states s_i, define the state transition function δ as

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

The objective is to find an UIO of length n for state s_1.

The instances in Def. 7 are illustrated in Fig. 3, and resemble an implementation of a *combination lock*, outputting the special symbol b only when given the "secret" code consisting of n 1-symbols. Note that $(s_n, s_1, 0/b)$ is the only state transition with distinguishing input/output behaviour. It is easy to see that the only UIO of length n for state s_1 is the input sequence $1^{n-1}0$.

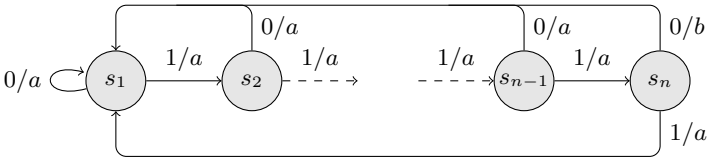

Fig. 3. Finding a UIO for state s_1 is hard for (1+1) EA.

Proposition 2 shows that this instance class leads to a fitness function that takes the same low value on all, except two input sequences. Hence, the fitness landscape is essentially a "needle in the haystack" which is hard for all EAs [5].

Proposition *2:* The fitness function f_H corresponding to the instance class in Def. 7 takes the value $f_H(x) = 1$ for all input sequences x, except on input sequences 1^n and $1^{n-1}0$ on which it takes the values $f_H(1^n) = 0$ and $f_H(1^{n-1}0) = n-1$.

Proof: The two special cases 1^n and $1^{n-1}0$ are simple. By the definition of the output function, $\lambda(s_i, 1^n) = a^n$ for any state s_i. Hence, $\gamma(1^n) = n$ so the value of the fitness function on the first special case is $f_H(1^n) = 0$.

On input sequence $1^{n-1}0$, the output function gives $\lambda(s_1, 1^{n-1}0) = a^{n-1}b$, and for states s_i different than s_1, the output function gives $\lambda(s_i, 1^{n-1}0) = a^n$. Hence, the value of the fitness function on the second special case is $f_H(1^{n-1}0) = 1$.

The remaining input sequences to consider are those that contain at least one 0-bit, but which are different from sequence $1^{n-1}0$. Such strings are of the form 1^k0z where k is an integer, $0 \leq k < n-1$, and z can be any sequence of length $\ell(z) = n-k-1$.

We claim that for any state s_i and such sequences, if $\lambda(s_1, 1^k0) = \lambda(s_i, 1^k0)$, then $\lambda(s_1, 1^k0z) = \lambda(s_i, 1^k0z)$. Suppose otherwise, that $\lambda(s_1, 1^k0) = \lambda(s_i, 1^k0)$ but $\lambda(s_1, 1^k0z) \neq \lambda(s_i, 1^k0z)$. But then we must have $\lambda(\delta(s_1, 1^k0), z) \neq \lambda(\delta(s_i, 1^k0), z)$, which implies the contradiction that $\lambda(s_1, z) \neq \lambda(s_1, z)$.

We now show that for the sequences on the form 1^k0z, there is exactly one state s_i for which $\lambda(s_1, 1^k0z) \neq \lambda(s_i, 1^k0z)$. We have just proved that this inequality requires that $\lambda(s_1, 1^k0) \neq \lambda(s_i, 1^k0)$. Because all states have the same output on input 1, it is necessary that $\lambda(\delta(s_1, 1^k), 0) \neq \lambda(\delta(s_i, 1^k), 0)$, which implies that $\lambda(s_{1+k}, 0) \neq \lambda(s_{i+k}, 0)$. The only way to satisfy this inequality is to let $i + k = n$. Hence, state s_{n-k} is the only state that produces different output than state s_1 on input sequences containing at least one 0-bit, and that are different from $1^{n-1}0$. ∎

By noting that the shortest UIO for state s_1 has length n, the following theorem can be proved similarly to Theorem 1.

Theorem *3:* The probability that random search will find a UIO for state s_1 in less than $e^{c\cdot n}$ iterations is exponentially small $e^{-\Omega(n)}$, where c is a small constant.

The drift theorem is a general technique for proving exponential lower bounds on first hitting-time in Markov processes and is an important technique in the theory of evolutionary computation [8]. The following variant of the drift theorem is taken from [6].

Lemma *1 (Drift Theorem):* Let $X_0, X_1, X_2, ...$ be a Markov process over a set of states S, and $g : S \to \mathbb{R}_0^+$ a function that assigns to every state a non-negative real number. Pick two real numbers $a(n)$ and $b(n)$ which depend on a parameter $n \in \mathbb{R}^+$ such that $0 < a(n) < b(n)$ holds and let random variable T denote the earliest point in time $t \geq 0$ where $g(X_t) \leq a(n)$ holds.

If there are constants $\lambda > 0$ and $D \geq 1$ and a polynomial $p(n)$ taking only positive values, for which the following four conditions hold

1) $g(X_0) \geq b(n)$
2) $b(n) - a(n) = \Omega(n)$
3) $E\left[e^{-\lambda(g(X_{t+1})-g(X_t))} \mid X_t, a(n) < g(X_t) < b(n)\right] \leq 1 - 1/p(n)$, for all $t \geq 0$, and
4) $E\left[e^{-\lambda(g(X_{t+1})-b(n))} \mid X_t, b(n) \leq g(X_t)\right] \leq D$, for all $t \geq 0$,

then for all time bounds $B \geq 0$, the following upper bound on probability holds for random variable T

$$\Pr[T \leq B] \leq e^{\lambda(a(n)-b(n))} \cdot B \cdot D \cdot p(n).$$

Theorem *4:* The probability that (1+1) EA will find the optimal solution on f_H within $e^{c \cdot n}$ steps is exponentially small $e^{-\Omega(n)}$, where c is a small constant.

Proof: We lower bound the time it takes until the current search point of (1+1) EA has at least $n-1$ 1-bits for the first time. This time is clearly shorter than the time the algorithm needs to find the optimal search point $1^{n-1}0$.

Let random variables $Y_0, Y_1, Y_2, ...$ represent the stochastic behaviour of (1+1) EA on fitness function f_H, where each variable Y_t denotes the number of zeros in the search point in step t. Then $Y_0, Y_1, Y_2, ...$ is a Markov process.

To simplify this Markov process, we introduce another Markov process $X_0, X_1, X_2, ...$, defined for all $t \geq 0$ as $X_0 := Y_0$, and

$$X_{t+1} := \begin{cases} X_t + 1 & \text{when } Y_{t+1} \geq Y_t + 2, \text{ and} \\ X_t + Y_{t+1} - Y_t & \text{otherwise .} \end{cases}$$

Let random variable T denote the first point in time t where $X_t \leq 1$. Intuitively, the simplified process corresponds to an "improved" algorithm which never looses more than one 0-bit in each step, but otherwise behaves as the (1+1) EA. Clearly, the expected optimisation time $E[T]$ of the modified process is no more than the expected optimisation time of the original process.

The drift theorem is now applied to derive an exponential lower bound on random variable T. Define $g(x) := x$ and parameters $a(n) := 1$ and $b(n) := cn$, where c is a constant that will be determined later. With this setting of $a(n)$ and $b(n)$, the *second condition* of the drift theorem is satisfied.

The following notation will be used

$$p_j := \Pr[g(X_{t+1}) - g(X_t) = j \mid X_t, 1 < g(X_t) < cn]$$
$$r_j := \Pr[g(X_{t+1}) - g(X_t) = j \mid X_t, cn \leq g(X_t)]$$

The terms in the equation

$$E\left[e^{-\lambda(g(X_{t+1})-g(X_t))} \mid X_t, 1 < g(X_t) < cn\right] = \sum_{j=-cn}^{n-cn} p_j \cdot e^{-\lambda j} \tag{2}$$

can be divided into four parts according to the value of the index variable j. The term where $j = 1$ simplifies to $p_1 \cdot e^{-\lambda} \leq e^{-\lambda}$, the term where $j = 0$ simplifies to $p_0 \cdot e^0 = (1 - 1/n)^n \leq 1/e$, the term where $j = -1$ simplifies to

$$p_{-1} \cdot e^{\lambda} \leq e^{\lambda} \left(1 - \frac{1}{n}\right)^{n-1} \frac{1}{n} \cdot X_t \leq e^{\lambda} c,$$

and the remaining terms where $j \leq -2$ can be simplified as follows:

$$\begin{aligned} \sum_{j=2}^{cn} e^{j\lambda} \cdot p_{-j} &= \sum_{j=2}^{cn} e^{j\lambda} \frac{1}{n^j} \left(1 - \frac{1}{n}\right)^{n-j} \binom{X_t}{j} \\ &\leq \sum_{j=2}^{cn} e^{j\lambda} \frac{1}{n^j} \frac{(cn)^j}{j!} = \sum_{j=2}^{cn} \frac{(e^{\lambda}c)^j}{j!} \\ &\leq -1 - e^{\lambda}c + \sum_{j=0}^{\infty} \frac{(e^{\lambda}c)^j}{j!} \\ &= -1 - e^{\lambda}c + \exp\left(e^{\lambda}c\right). \end{aligned}$$

The sum in Eq. (2) can now be bounded from above as

$$\begin{aligned} & E\left[e^{-\lambda(g(X_{t+1})-g(X_t))} \mid X_t, 1 < g(X_t) < cn\right] \\ &\leq \quad e^{-\lambda} + 1/e + e^{\lambda}c - 1 - e^{\lambda}c - \exp\left(e^{\lambda}c\right) \\ &= \quad e^{-\lambda} + 1/e - 1 + \exp\left(e^{\lambda}c\right). \end{aligned}$$

For appropriate values of λ and c (eg. $\lambda = \ln 2$ and $c = 1/32$), the value of this expression is less than $1-\delta$ for a constant $\delta > 0$. Hence, the *third condition* in the drift theorem is satisfied. It is straightforward to see that the *fourth condition* holds now that condition three holds.

$$\begin{aligned} & E\left[e^{-\lambda(g(X_{t+1})-cn)} \mid X_t, cn \leq g(X_t)\right] \\ &\leq \quad E\left[e^{-\lambda(g(X_{t+1})-g(X_t))} \mid X_t, cn \leq g(X_t)\right] \\ &= \quad r_1 \cdot e^{-\lambda} + \sum_{j=0}^{n} r_{-j} \cdot e^{j\lambda}. \end{aligned}$$

Using the same ideas as above, the expectation can be bounded from above by

$$E\left[e^{-\lambda(g(X_{t+1})-cn)} \mid X_t, cn \leq g(X_t)\right] \leq e^{-\lambda} + \exp(e^{\lambda}).$$

When parameter $c = 1/32$, using Chernoff bounds [11], the probability that first search point has less than cn zeros is $e^{-\Omega(n)}$. Hence we can assume with high probability that the *first condition* is satisfied as well.

All four conditions of the Drift theorem now hold. By setting $B = e^{c' \cdot n}$ for some small constant c', one obtains the exponential lower bound $\Pr\left[T \leq e^{c' \cdot n}\right] = e^{-\Omega(n)}$. ■

C. Instances with tunable difficulty

The previous two subsections presented classes of FSMs for which it is either easy or hard to compute a UIO with the (1+1) EA. To complement these results, this subsection presents an instance class with tunable difficulty. The instance class is defined with respect to a parameter k. It will be shown that the instance class is easy when the value of parameter k is low, and the problem becomes harder when parameter k is increased.

Definition *8 (k-gap instances):* For instance sizes $n \geq 7$, let k be a constant integer, $2 \leq k \leq (n-3)/2$ and define $m := n-k-1$. Define an FSM $G(k)$ with input and output symbols $I := \{0,1\}$ and $O := \{a,b\}$ respectively, and n states $S := \{s_1\} \cup \{r_1, r_2, ..., r_k\} \cup \{q_1, q_2, ..., q_m\}$. For all states t in S, define the output function λ as

$$\lambda(t,0) := b \text{ and } \lambda(t,1) := \begin{cases} b & \text{if } t = q_m, \text{ and} \\ a & \text{otherwise.} \end{cases}$$

For state s_1, define the state transition function δ as

$$\delta(s_1,0) := s_1 \text{ and } \delta(s_1,1) := r_1.$$

For states r_i, $1 \leq i \leq k$, define the state transition function δ as

$$\delta(r_i,1) := s_1 \text{ and } \delta(r_i,0) := \begin{cases} q_{k+2} & \text{if } i = k, \text{ and} \\ r_{i+1} & \text{otherwise.} \end{cases}$$

And finally, for states q_i, $1 \leq i \leq m$, define the state transition function δ as

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

The objective is to find an UIO of length n for state s_1.

One way of creating a problem with tunable difficulty is to make sure that the fitness function contains a "trap" which easily leads the EA into a local optimum at distance k from the global optimum. By increasing the distance k between the local and global optimum, the problem gets harder [4]. The "trap" in the FSM defined in Def. 8 are the m states $q_1, ..., q_m$. By producing an input sequence with many leading 1-bits, the (1+1) EA easily makes the output from these states different from state s_1. However, as can be seen from Fig. 4, the UIO for state s_1 must contain k 0-bits somewhere in the beginning of the input sequence.

Proposition *3:* Let z be any string with length $\ell(z) = k+1$.

$$f_{G(k)}(10^k 1^{n-2k-2} z) = n-1. \tag{3}$$

In other words, any search point on the form $10^k 1^{n-2k-2} z$ is a UIO for state s_1. Let i be any integer $0 \leq i < n$, and z any string of length $n-i-1$. If string z does not contain the substring 1^{n-2k-2}, then

$$f_{G(k)}(1^i 0 z) = \min(i, n-k-1), \quad \text{and} \tag{4}$$

$$\gamma(1^i) = \gamma(1^i 0 z). \tag{5}$$

Proof: We first prove Eq. (3). On input sequence 10^k, only $\delta(s_1, 10^k) = q_{k+2}$ and for all other states t, $\delta(t, 10^k) = s_1$. Hence, state s_1 goes through the distinguishing transition on input $10^k 1^{n-2k-2}$ while all other states are in transition between states s_1 and r_1, showing that state s_1 has a unique output. Therefore, search points on the form $10^k 1^{n-2k-2} z$ are optimal. (There are other optimal search points, but knowing the structure of a few optimal search points will be sufficient in the analysis.)

We now show that $\gamma(1^i 0) = n - \min(i,m)$. Note that $(q_m, q_1, 1/b)$ is the only distinguishing state transition. For i no larger than m, the i states $q_{m-i+1}, ..., q_m$ reach this transition on input sequence 1^i and therefore produce different outputs than state s_1. For i at least m, all m states $q_1, ..., q_m$ reach the distinguishing transition. State s_1 and the k states $r_1, ..., r_k$ do not reach the distinguishing transition on input sequence $1^i 0$. Therefore, the number of states that produce different outputs than state s_1 on input sequence $1^i 0$ is $\min(i,m)$.

Finally, we prove Eq. (4) and Eq. (5) under the assumption that z does not contain the substring 1^{n-2k-2}. For all states s, either $\delta(s, 1^i 0) = s_1$ or $\delta(s, 1^i 0) = r_2$. All state transition paths from either state s_1 or state r_2 to the distinguishing state transition from state q_m must go through the $n-2k-2$ state transitions between q_{k+2} and q_m. Transitions along this path require an input sequence with $n-2k-2$ consecutive 1-bits, which is not possible with sequence z. Therefore, we have $\gamma(1^i 0 z) = \gamma(1^i 0) = n - \min(i,m)$. This also proves Eq. (5) because $\gamma(1^i) = \gamma(1^i 0) = n - \min(i,m)$. ■

Proposition *4:* Let i be any integer $0 \leq i \leq 2k+2$, and z any sequence of length $\ell(z) = n-i-1$ containing the sequence 1^{n-2k-2}. If the sequence $1^i 0 z$ is not optimal, then $f_{G(k)}(1^i 0 z) \leq 2k+2$.

Proof: Assume first that $i = 0$, i.e. the search point begins with a 0-bit. In this case, all states $q_1, ..., q_m$ collapse with state s_1, and the suffix z can at most distinguish s_1 from the k states $r_1, ..., r_k$. Hence, in this case $\gamma(0z) \geq n-k$.

Assume now that $1 \leq i \leq k+2$. After input $1^i 0$, all states have moved to either state s_1 or state r_2. If i is even, then state s_1 has collapsed with states $q_1, ..., q_m$. Hence, the suffix z can at most distinguish the k states $r_1, ..., r_k$ from state s_1, i.e. $\gamma(1^i 0 z) \geq n-i-k \geq n-(k+2)-k$. If i is odd, then states $r_1, ..., r_k$ have collapsed with states $q_1, ..., q_m$. So if x is not optimal, then $\gamma(1^i 0 z) = n-i \geq n-k-2$.

Finally, assume that $k+2 < i \leq 2k+2$. After input $1^i 0$, no more states can reach the distinguishing transition because moving from state s_1 or state r_2 to the distinguishing transition requires at least the subsequence $0^{k-1} 1^{n-2k-2}$ which is longer than subsequence z. So in this case, we have $\gamma(1^i 0 z) = n-i \geq n-(2k-2)$. ■

Analysing the (1+1) EA on the problem is easy if we can assume that the sequence 1^{n-2k-2} never occurs in the suffix. Proposition 5 shows that this assumption holds in most cases.

Definition *9 (Typical run):* A *typical run* of (1+1) EA on $f_{G(k)}$ is a run where the current search point x is never on the form $1^i 0 z$, $0 \leq i < 2k+2$, where z is a sequence of length $\ell(z) = n-i-1$ containing the sequence 1^{n-2k-2}. A run of (1+1) EA on $f_{G(k)}$ is divided into the following three phases:

Phase 1 is defined as the time interval in which the search point has less than $2k+2$ leading 1-bits. If the current search point during this phase has a suffix containing sequence 1^{n-2k-2}, then we say that we have a *failure*. The event of failure will be denoted $\mathcal{F}$. *Phase 2* is defined as the time

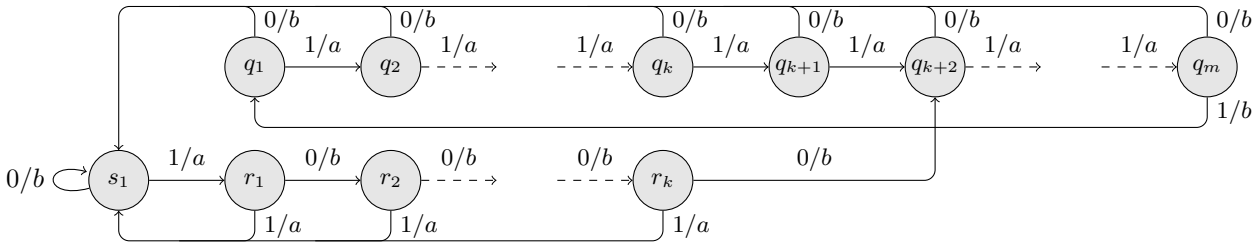

Fig. 4. Finding a UIO for state s_1 with (1+1) EA becomes harder when increasing parameter k.

interval when the search point has between $2k+2$ and $n-k-1$ leading 1-bits. *Phase 3* is defined as the time interval when the search point has at least $n-k-1$ leading 1-bits, and this phase lasts until the search point is optimal for the first time.

Proposition *5:* The probability of a failure during Phase 1 is bounded from above by $e^{-\Omega(n)}$.

Proof: The current search point x of (1+1) EA in Phase 1 is on the form $x = 1^i 0 z$ for some i, $0 \leq i < 2k+2$ and z a string of length $\ell(z) = n-i-1$. We call this substring z occurring after the first 0-bit the *suffix* of the current search point.

We first show that as long as the run for the first t steps has been typical, then the suffix z in step $t+1$ is a random string. The initial search point is a random string, so the suffix is also a random string. Assume that the run has been typical until step t and the suffix z is a random string. By Eq. (5) in Proposition 3, any bitflip of the suffix will be accepted. Randomly mutating a random string, will clearly produce a new random string. The suffix in step $t+1$ will therefore be a random string. The suffix z of the new search point in step $t+1$ can contain 1^{n-2k-2}, i.e. we may have a failure in step $t+1$. However, we show that this is unlikely. The probability that the string 1^{n-2k-2} occurs in a random string shorter than n is no more than $(2k+2)\cdot 2^{-n+2k+2}$, which for large n is less than $e^{-n/16}$. One way of increasing the number of leading 1-bits without having a failure is by flipping the first 0-bit and flip no other bits. So the probability of increasing the number of leading 1-bits without having a failure in the following step is at least $(1/n)\cdot(1-1/n)^{n-1} \geq 1/en$.

Hence, for large n, the probability that the number of leading 1-bits increases before we have a failure is at least

$$\frac{1/en}{1/en + 1/e^{n/16}} \geq 1 - ne\cdot e^{-n/16} \geq 1 - e^{-n/32}.$$

A failure must occur before the number of leading 1-bits has been increased more than $2k+2$ times. So the failure probability $\Pr[\mathcal{F}]$ is no more than

$$\sum_{i=0}^{2k+2} \left(1-e^{-n/32}\right)^i \cdot e^{-n/32} \leq (2k+2)\cdot e^{-n/32} = e^{-\Omega(n)}.$$

■

Theorem *5:* Let k be any constant integer $k \geq 2$. The expected runtime of (1+1) EA on $f_{G(k)}$ is asymptotically $\Theta(n^k)$.

Proof: Given the probability of the failure event $\mathcal{F}$, the expected runtime of (1+1) EA can be calculated as

$$E[T] = (1-\Pr[\mathcal{F}])\cdot E\left[T \mid \overline{\mathcal{F}}\right] + \Pr[\mathcal{F}]\cdot E[T \mid \mathcal{F}]. \quad (6)$$

To estimate an *upper bound* on the the expected runtime, we use that $E[T] \leq E\left[T \mid \overline{\mathcal{F}}\right] + \Pr[\mathcal{F}]\cdot E[T \mid \mathcal{F}]$. We will first find an upper bound on the runtime conditional on a typical run $E\left[T \mid \overline{\mathcal{F}}\right]$ and pessimistically assume that the optimal search point will not be found during Phase 1 or 2 of the run. We first upper bound the duration of Phase 1 and 2. Let $i, 0 \leq i \leq n-k-1$, be the number of leading 1-bits in the current search point. A step of the algorithm is called successful if the mutated search point x' has more leading 1-bits than the current search point x. In typical runs, Proposition 3 guarantees that x' will be accepted in a successful step. To reach the end of Phase 2, we have to wait at most for $n-k-1$ successful steps. The probability of a successful step is at least $1/n\cdot(1-1/n)^{n-1} \geq 1/en$, so the expected duration of Phase 1 and Phase 2 is $O(n^2)$. By Proposition 3, for Phase 3 to end, it is sufficient to flip k consecutive 1-bits starting at position 2. The probability that this will happen in any step of Phase 3 is at least $(1/n)^k\cdot(1-1/n)^{n-k} \geq 1/(n^k e)$. Hence, the expected duration of Phase 3 is bounded from above by $O(n^k)$. An upper bound on the expected runtime conditional on the event that the run is typical is therefore $E\left[T \mid \overline{\mathcal{F}}\right] = O(n^k)$.

We now give an upper bound on the expected time $E[T \mid \mathcal{F}]$ conditional on a failure. To keep the analysis simple, we give a pessimistic upper bound. At some time in such a run, the current search point has a suffix containing the sequence 1^{n-2k-2}. We assume that this search point is not the optimal search point, and furthermore, we assume that in this situation, we will never accept an optimal search point during Phase 1. Clearly, this will only slow down the optimisation process. By Proposition 4, this search point has fitness at most $2k+2$. To end Phase 1, Proposition 3 shows that it is sufficient to wait for a step in which all the 0-bits in the $2k+3$ long prefix of the search point is flipped into 1-bits. The probability of such a mutation is at least $(1/n)^{2k+3}(1-1/n)^{n-2k-3} \geq 1/en^{2k+3}$. So if a failure occurs, the duration of Phase 1 will be no longer than $O(n^{2k+3})$. Failures do not occur in Phase 2 or Phase 3, we therefore reuse the upper bounds of $O(n^2)$ and $O(n^k)$ that were calculated for the typical runs, yielding an upper bound of $O(n^{2k+3})$ for the duration of runs with failures. Due to the exponentially small failure probability, the unconditional expected runtime of (1+1) EA is therefore $E[T] = O(n^k)$.

A *lower bound* on the expected runtime is estimated using the inequality $E[T] \geq (1 - \Pr[\mathcal{F}]) \cdot E\left[T \mid \overline{\mathcal{F}}\right]$. We need to estimate the expected runtime conditional on a typical run. Optimal search points contain the suffix 1^{n-2k-2}, hence the optimal search point will not be found during Phase 1 of typical runs. By Proposition 3 and Proposition 4, only search points with at least $2k+2$ leading 1-bits or an optimal search point will be accepted during Phase 2. Optimal search points must contain $10^k 1^{n-2k-2}$. Hence, in order to find the optimum in the second phase, it is necessary to flip k consecutive 1-bits into 0-bits, starting somewhere in the interval between position 2 and $k+2$. The probability of this event in any given step is no more than k/n^k. Hence, the expected duration of Phase 2 and Phase 3 is at least n^k/k steps. The unconditional expected runtime can now be bounded from below by $E[T] \geq (1 - e^{-\Omega(n)}) \cdot n^k/k = \Omega(n^k)$. ■

V. Discussion

Three classes of finite state machines have been constructed and studied in this paper. These classes of FSMs are not based on any real world system, but have been constructed in a way which allows us to make use of the current knowledge about the runtime behaviour of the (1+1) EA. Most of the existing results on the runtime behaviour of the (1+1) EA are on artificial problems, like the OneMax, Needle, LeadingOnes and Jump [4]. This paper shows that the results and techniques obtained from theoretical investigations of such constructed "toy" problems can be useful for understanding the behaviour of EAs on real-world problems, like the problem of computing UIOs.

Although the instance classes studied in this paper are not real-world systems in themselves, they may very well occur as sub-modules within more complex systems. For example, the easy instance class in Def. 6 resembles a classical counter, and the hard instance class in Def. 7 resembles an FSM implementation of a digital combination lock. Alternatively, one can envisage the FSM in Def. 7 as a part of a larger FSM used for parsing. The FSM could be used to accept strings which contain exactly n repetitions of a particular symbol.

The notions of easy and hard instances depend on both the EA used and on the way the problem of finding UIOs has been defined. This paper uses the terms hard and easy relative to the (1+1) EA, as described in Section III-B. These terms should not be confused with the terms EA-hard and EA-easy which are sometimes used in evolutionary computation to mean problems that are thought to be generally hard, respectively easy for *all* EAs. There are certainly functions that are hard in the sense of Section III-B for (1+1) EA, but which are easy for other EAs. Furthermore, the hardness of finding UIOs is relative to the way the fitness function for this problem has been defined. We believe the formulation in Def. 3 is quite natural, however one could envisage other fitness function definitions which could potentially lead to different runtimes for the (1+1) EA.

VI. Conclusion

This paper has analysed the runtime of (1+1) EA on the problem of computing unique input output sequences in finite state machines. It is shown that there are instances of the problem where the (1+1) EA is highly efficient, whereas random search fails completely. This result indicates that a (1+1) EA can be preferable over the sometimes proposed strategy of randomly searching for UIOs. Furthermore, it is shown that there are instances of this problem where the (1+1) EA is unsuccessful. On this particular instance class, the state partition tree gives little information about the UIO. The existence of such hard instances for the (1+1) EA is to be expected since the general problem of finding UIOs is NP-hard. Although such hard instances can be constructed, it is hoped that the (1+1) EA will be successful on wide range of natural instances of the problem. Finally, an instance class with tunable difficulty for the (1+1) EA is presented. This instance class highlights how specific, small changes to the FSM can make the problem of computing UIOs increasingly hard.

Runtime analysis of EAs is still in an early stage. Most previous research has been concerned with relatively simple toy problems and other artificially constructed classes of fitness functions. This paper shows that it is possible to analyse the runtime of EAs on real world problems.

Acknowledgements

The authors would like to thank Mark Harman, Rob Hierons, Pietro Oliveto, Ramón Sagarna and Andrea Arcuri for their useful comments. This work was supported by EPSRC under grant no. EP/D052785/1.

References

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

[2] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw Hill, New York, NY, 2nd edition, 2001.

[3] Karnig Derderian, Robert M. Hierons, Mark Harman, and Qiang Guo. Automated unique input output sequence generation for conformance testing of fsms. *The Computer Journal*, 49(3):331–344, 2006.

[4] Stefan Droste, Thomas Jansen, and Ingo Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276:51–81, 2002.

[5] Stefan Droste, Thomas Jansen, and Ingo Wegener. Upper and lower bounds for randomized search heuristics in black-box optimization. *Theory of Computing Systems*, 39(4):525–544, 2006.

[6] Oliver Giel. *Zur Analyse von randomisierten Suchheuristiken und Online-Heuristiken*. PhD thesis, Universität Dortmund, 2005.

[7] Q. Guo, R. M. Hierons, M. Harman, and K. Derderian. 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*, pages 164–177, 2004.

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

[9] D. Lee and M. Yannakakis. Testing finite-state machines: state identification and verification. *IEEE Transactions on Computers*, 43(3):306–320, 1994.

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

[11] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[12] Heinz Mühlenbein. How genetic algorithms really work I. Mutation and Hillclimbing. In *Proceedings of the Parallel Problem Solving from Nature 2, (PPSN-II)*, pages 15–26. Elsevier, 1992.