UNIVERSITY OF DORTMUND

REIHE COMPUTATIONAL INTELLIGENCE

COLLABORATIVE RESEARCH CENTER 531

Design and Management of Complex Technical Processes and Systems by means of Computational Intelligence Methods

Theoretical Aspects of Evolutionary Algorithms

Ingo Wegener

No. CI-111/01

Technical Report ISSN 1433-3325 May 2001

Secretary of the SFB 531 · University of Dortmund · Dept. of Computer Science/XI
44221 Dortmund · Germany

This work is a product of the Collaborative Research Center 531, "Computational Intelligence", at the University of Dortmund and was printed with financial support of the Deutsche Forschungsgemeinschaft.

THEORETICAL ASPECTS OF EVOLUTIONARY ALGORITHMS

Ingo Wegener*

FB Informatik, LS2, Univ. Dortmund, 44221 Dortmund, Germany
wegener@ls2.cs.uni-dortmund.de

Abstract. Randomized search heuristics like simulated annealing and evolutionary algorithms are applied successfully in many different situations. However, the theory on these algorithms is still in its infancy. Here it is discussed how and why such a theory should be developed. Afterwards, some fundamental results on evolutionary algorithms are presented in order to show how theoretical results on randomized search heuristics can be proved and how they contribute to the understanding of evolutionary algorithms.

1 Introduction

Research on the design and analysis of efficient algorithms was quite successful during the last decades. The very first successful algorithms (Dantzig's simplex algorithm for linear programming and Ford and Fulkerson's network flow algorithm) have no good performance guarantee. Later, research was focused on polynomial-time algorithms (see Cormen, Leiserson, and Rivest (1990)) and this type of research has been extended to approximation algorithms (see Hochbaum (1997)) and randomized algorithms (see Motwani and Raghavan (1995)). Indeed, designing and implementing an efficient algorithm with a proven performance guarantee is the best we can hope for when considering an algorithmic problem. This research has led to a long list of efficient problem-specific algorithms. Moreover, several paradigms of algorithms have been developed, among them divide-and-conquer, dynamic programming, and branch-and-bound. There are general techniques to design and analyze algorithms. However, these paradigms are successful only if they are realized with problem-specific modules.

Besides these algorithms also paradigms for the design of heuristic algorithms have been developed like randomized local search, simulated annealing, and all types of evolutionary algorithms, among them genetic algorithms and evolution strategies. These are general classes of search heuristics with many free modules and parameters. We should distinguish problem-specific applications where we are able to choose the modules and parameters knowing properties of the considered problem and problem-independent realizations where we design a search heuristic to solve all problems of a large class of problems. We have to argue why

* This work was supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the Collaborative Research Center "Computational Intelligence" (531).

one should investigate such a general scenario. One main point is that we obtain the frame of a general search heuristic where some details may be changed in problem-specific applications. Moreover, there are at least two situations where problem-independent algorithms are of particular interest. First, in many applications, one has not enough resources (time, money, specialists,...) to design a problem-specific algorithm or problem-specific modules. Second, often we have to deal with "unknown" functions which have to be maximized. This scenario is called black box optimization. It is appropriate for technical systems with free parameters where the behavior of the system cannot be described analytically. Then we obtain knowledge about the unknown function only by "sampling". The t-th search point can be chosen according to some probability distribution which may depend on the first $t-1$ search points $x_1, \dots, x_{t-1}$ and their function values $f(x_1), \dots, f(x_{t-1})$. One main idea of all randomized search heuristics is to "forget" much of the known information and to make the choice of the probability distribution only dependent on the "non-forgotten" search points and their f-values.

Our focus is the maximization of pseudo-boolean functions $f : \{0,1\}^n \to \mathbb{R}$ which covers the problems from combinatorial optimization. We investigate and analyze randomized search heuristics which are designed to behave well on "many" of the "important and interesting" pseudo-boolean functions. Obviously, they cannot beat problem-specific algorithms and, also obviously, each randomized search heuristic is inefficient for most of the functions. The problem is to identify for a given randomized search heuristic classes of functions which are optimized efficiently and to identify typical functions where the heuristic fails. Such theoretical results will support the selection of an appropriate search heuristic in applications. One may also assume (or hope) that the search heuristic behaves well on a function which is "similar" to a function from a class where it is proved that the heuristic is efficient. Moreover, the proposed results lead to a better understanding of search heuristics. This again leads to the design of improved search heuristics and gives hints for a better choice of the parameters of the search heuristic. Finally, analytical results support the teaching of randomized search heuristics.

In black box optimization the black box (or oracle) answers queries x with $f(x)$ where $f : \{0,1\}^n \to \mathbb{R}$ is the function to be maximized. Since queries are expensive, the search cost is defined as the number of queries. For a fixed search heuristic let X_f be the random number of queries until "some good event" happens. The good event in this paper is that a query point is f-maximal. Then we are interested in the expected optimization time $E(X_f)$ and the success probability function $s(t) := \text{Prob}(X_f \leq t)$. This is an abstraction from the real problem, since obtaining the f-value of some optimal x does not imply that we know that x is optimal. In applications, we additionally need good stopping rules.

Our focus is on evolutionary algorithms which have been developed in the sixties of the last century and which have found many applications during the last ten years. Evolutionary algorithms are described in many monographs (Fogel

(1995), Goldberg (1989), Holland (1975), Schwefel (1995)) and in a more recent handbook (Bäck, Fogel, and Michalewicz (1997)). The experimental knowledge is immense, but the theory on evolutionary algorithms is still in its infancy. One can find several results on the one-step behavior of evolutionary algorithms, but these results most often have no implications on the expected optimization time or the success probability function. The famous schema theorem belongs to this category. There are even more results using simplifying or even unrealistic assumptions. The building-block hypothesis is such an idealized hypothesis which has turned out to be wrong in many realistic scenarios. Another idealized analysis works with "infinite populations". This makes it possible to apply methods from statistical dynamics. We claim that it is necessary to develop results on the expected optimization time and the success probability function which are not based on any assumptions, in particular, for generic variants of evolutionary algorithms and for "interesting" subclasses of functions. This does not exclude the investigation of fundamental problems without direct implications for concrete algorithms. The paper of Rabani, Rabinovich, and Sinclair (1995) is exemplary for such an approach.

In the rest of the paper, we elucidate our approach with some results. In Section 2, we introduce the simplest variant of an evolutionary algorithm, the so-called $(1+1)$EA, and in the following three sections we present results on the behavior of the $(1+1)$EA. In Section 3, we investigate monotone polynomials of bounded degree and, in Section 4, the special classes of affine functions and royal road functions. Section 5 contains an overview of further results on the $(1+1)$EA and some of its generalizations. In Section 6, we introduce a generic genetic algorithm which applies a crossover operator and discuss why it is more difficult to analyze evolutionary algorithms with crossover than evolutionary algorithms based solely on mutation and selection. Section 7 contains the first proof that crossover reduces the expected optimization time for some specific function from exponential to polynomial. We finish with some conclusions.

2 A simple evolutionary algorithm

We describe the simplest variant of an evolutionary algorithm which works with population size 1 and is based solely on selection and mutation.

Algorithm 1 ($(1+1)$EA).

1.) Initialization: The current string $x \in \{0,1\}^n$ is chosen randomly using the uniform distribution.
2.) Selection for mutation: The current string x is chosen.
3.) Mutation: The offspring x' of x is created in the following way. The bits x'_i are independent and $\text{Prob}(x'_i = 1 - x_i) = p_m(n)$ (this parameter is called mutation probability).
4.) Selection of the next generation: The new current string equals x', if $f(x') \geq f(x)$, and x, otherwise.
5.) Continue at Step 2 (until some stopping criterion is fulfilled).

The generic value of $p_m(n)$ equals $1/n$ implying that, on average, one bit is flipped. Then the number of flipping bits is asymptotically Poisson distributed (with parameter 1). The algorithm can easily be generalized to larger population size μ. Then Step 2 is not trivial. The number of offsprings can be generalized to λ. There are many selection schemes for Step 4. The most prominent are $(\mu+\lambda)$-selection (the best μ of the μ parents and the λ offsprings are chosen) and (μ, λ)-selection (the best μ of the λ offsprings are chosen). These two selection schemes lead to the class of so-called evolution strategies (which have been developed for continuous search spaces $\mathbb{R}^n$). This explains the notion $(1+1)$EA for Algorithm 1 which can be interpreted as evolution strategy with population size 1. Another possibility is to interpret Algorithm 1 as a randomized hill climber, since it does not accept an offspring with a smaller f-value (fitness). A crucial point is that each $x' \in \{0,1\}^n$ has a positive probability of being created as an offspring of x. Hence, the $(1+1)$EA cannot get stuck forever in a non-optimal region. The analysis of the $(1+1)$EA is interesting, since

- the $(1+1)$EA is for many functions surprisingly efficient,
- the analysis of the $(1+1)$EA reveals many analytical tools for the analysis of more general evolutionary algorithms, and
- the $(1+1)$EA can be interpreted as evolutionary algorithm and as randomized hill climber.

The reason for larger populations is that a single search point may randomly choose "the wrong way" and may reach a region which makes it difficult to find the optimum. Working with a larger population one hopes that not all individuals of the current population go into a wrong direction and that some of them find the optimum efficiently. However, the individuals are not considered independently. If the individuals "on the wrong way" have during the following steps a larger fitness, they may drive out all individuals "on the right way" by selection. Hence, it is crucial to have a selection scheme which supports "enough" diversity in the population and which nevertheless eliminates bad individuals. Multi-start variants of the $(1+1)$EA cope in many situations with these problems, since the different runs of the $(1+1)$EA are independent. Performing $m(n)$ runs, each for $t(n)$ steps, leads to a success probability of $1-(1-s(t(n)))^{m(n)}$, if $s(t(n))$ is the success probability of a single run of the $(1+1)$EA.

3 The $(1+1)$EA on monotone polynomials

Pseudo-boolean functions $f : \{0,1\}^n \to \mathbb{R}$ have a unique representation as polynomials, i.e.,

$$f(x) = \sum_{A \subseteq \{1,\ldots,n\}} w_A \cdot \prod_{i \in A} x_i.$$

The degree of f is the largest size of a set A where $w_A \neq 0$. It is well known that the maximization of pseudo-boolean polynomials of degree 2 is NP-hard and Wegener and Witt (2001) have explicitly defined a degree-2 polynomial where not only the expected optimization time of the $(1+1)$EA is exponential but also multi-start variants fail, since for some $c > 0$ the success probability after $2^{cn \log n}$ steps is $2^{-\Omega(n)}$. Such a function is almost a worst case function for the $(1+1)$EA, since the expected optimization time for each pseudo-boolean function is bounded above by $n^n = 2^{n \log n}$. This follows, since the probability to produce an optimal string within one step is always lower bounded by n^{-n}. We investigate monotone polynomials, i.e., polynomials where all weights w_A, $A \neq \emptyset$, are non-negative. The $(1+1)$EA treats zeros and ones in the same way. Therefore, our results also hold for polynomials which are obtained from monotone polynomials by replacing some variables x_i with $\overline{x_i} = 1 - x_i$. This includes all affine, i.e., degree-1 functions.

Knowing that a pseudo-boolean function is a monotone polynomial, the maximization is trivial. The all-one string always is optimal. However our motivation is black box optimization and we like to investigate the behavior of the $(1+1)$EA on monotone polynomials. This subclass of functions is interesting, since we can investigate the expected run time with respect to natural parameters, namely the input length n, the degree d, and the number N of terms with non-zero weight. Moreover, improvements are not always possible by the mutation of a small number of bits and strings with a large Hamming distance from the optimum may have much larger f-values than strings close to the optimum. It is easy to see that the degree is a crucial parameter. Garnier, Kallel, and Schoenauer (1999) have proved that the $(1+1)$EA has an expected optimization time of $\Theta(2^n)$ on the n-degree polynomial $x_1 x_2 \cdots x_n$. For this function we are searching for a needle, the all-one string 1^n, in a big haystack, namely $\{0,1\}^n$. It is obvious that such functions are difficult for black box optimization. The cited result can be extended to the general case of $N = 1$.

Lemma 1. *The expected optimization time of the* $(1+1)$*EA on a polynomial with* $N = 1$ *and degree* d *equals* $\Theta(n2^d/d)$*.*

Sketch of Proof. W.l.o.g. the polynomial equals $x_1 x_2 \cdots x_d$. The probability that at least one of the d essential bits flips in one step equals $1 - (1 - \frac{1}{n})^d = \Theta(d/n)$. Hence, the expected optimization time is by a factor of $\Theta(n/d)$ larger than the expected number of so-called active steps where one of the essential bits flips. As long as we have not found an optimal string, each new string is accepted and we have to analyze a simple Markoff chain. This can be done by standard arguments following Garnier, Kallel, and Schoenauer (1999). The expected number of active steps equals $\Theta(2^d)$ and the upper bound $O(2^d)$ holds for each initial string. □

This lemma proves that the $(1+1)$EA is efficient in the following black box scenario. We know that the function f is one of the functions which equals 1 if $x_1 = a_1, \ldots, x_d = a_d$ for some $(a_1, \ldots, a_d) \in \{0,1\}^d$ and 0 otherwise. No sampling algorithm can generate a smaller average optimization time than $(2^d + 1)/2$. We have an additional factor of $\Theta(n/d)$ for the so-called passive steps and only an additional factor of $\Theta(1)$ for active steps visiting some d-prefix which has been visited before. Moreover, for $d = \omega(\log n)$ we cannot hope for randomized search heuristics with an expected polynomial optimization time.

The following analysis of the $(1+1)$EA on low-degree monotone polynomials shows its efficiency on a large class of interesting functions. Moreover, the proof presents typical analytical tools.

Theorem 1. *The expected optimization time of the* $(1+1)EA$ *on a monotone polynomial with* N *non-vanishing terms and degree* $d \leq \log n$ *is bounded by* $O(Nn2^d/d)$*, i.e., by* $O(Nn)$ *for constant* d *and by* $O(Nn^2/\log n)$ *for all* $d \leq \log n$*.*

Sketch of Proof. Let $A(1), \ldots, A(N)$ be the N sets such that the weights $w_{A(i)}$ are non-vanishing, i.e., $w_{A(i)} > 0$, since the polynomial f is monotone. To simplify the notation we set $w_i = w_{A(i)}$ and assume w.l.o.g. that $w_1 \geq \cdots \geq w_N > 0$. A weight w_i is called active with respect to the string a, if $a_j = 1$ for all $j \in A(i)$, and w_i is called passive otherwise. The $(1+1)$EA can be described by a Markoff chain on $\{0,1\}^n$ and we have to estimate the expected time until we reach a string a such that all weights $w_1, \ldots, w_N$ are active with respect to a. The f-value of the current string is not decreasing during the search process.

A quite general technique is to partition $\{0,1\}^n$ into "fitness layers" and to estimate the expected time to leave a non-optimal layer. The choice of the layers is crucial. Here we choose $N+1$ layers $L_0, \ldots, L_N$ where

$$L_i := \{a | w_1 + \cdots + w_i \leq f(a) < w_1 + \cdots + w_{i+1}\},$$

if $i < N$, and $L_N := \{a | f(a) = w_1 + \cdots + w_N\}$ consists of all optimal strings. The search process leaves each layer at most once. If T_i is an upper bound for the expected time of leaving L_i from an arbitrary $a \in L_i$, then $T_0 + \cdots + T_{N-1}$ is an upper bound for the expected optimization time of the $(1+1)$EA on f. We prove the theorem by proving that $T_i = O(n2^d/d)$.

Let $a \in L_i$. Then, by definition, there exists some $j \leq i+1$ such that w_j is passive with respect to a. Moreover, if w_j gets active while no active w_m gets passive, we leave L_i. We assume w.l.o.g. that the monomial belonging to w_j equals $x_1 x_2 \cdots x_k$, $k \leq d$. The idea is to compare the "complicated" Markoff chain M_1 which describes the $(1+1)$EA on f starting with a and stopping when it leaves L_i with the "simple" Markoff chain M_2 which describes the $(1+1)$EA on $g(x) := x_1 x_2 \cdots x_k$ starting with a and stopping when it reaches a g-optimal string.

The analysis of M_2 (see Lemma 1) is simple, since each string is accepted until the process stops. M_1 is more complicated, since it is influenced by the other monomials. Some new strings are not accepted, since some of the active weights are deactivated. This can even happen for steps increasing the number of ones in the k-prefix of the string and in the $(n-k)$-suffix of the string. Nevertheless, since all weights are non-negative, we do not believe that this will be significant. In order to simplify the analysis we choose for each $m \in \{0, \ldots, k\}$ a string $a^m = (b^m, c^m)$ among the strings in L_i with m ones in the k-prefix b^m such that the expected time of M_1 to leave L_i when starting in a^m is maximal. Let

M_1' be the Markoff chain obtained from M_1 by replacing each string in L_i with m ones in the k-prefix with a^m. Let M_2' be the Markoff chain obtained from M_2 by replacing each string with m ones in the k-prefix with a^m. The expected stopping time of M_2' is by definition of g equal to the expected stopping time of M_2. The advantage is that M_1' and M_2' are Markoff chains on the small state space $\{0, \ldots, k\}$ representing the number of ones in the prefix.

It is sufficient to prove that for some constant $c' > 0$ the success probability of M_1' within $c'n2^d/d$ steps is bounded below by $\varepsilon > 0$, since the expected number of such phases then is bounded above by ε^{-1}. We analyze one phase of M_1' and estimate the failure probability, namely the probability of not leaving L_i.

If w_j gets active, it may happen that other weights get passive and we do not leave L_i. However, if w_j gets active, exactly all zeros in the k-prefix flip. The behavior of the bits in the $(n-k)$-suffix is independent of this event. Hence, with a probability of $(1-\frac{1}{n})^{n-k} \geq e^{-1}$ none of these bits flips implying that no active weight gets passive and we leave L_i. If one suffix bit flips in the step where w_j gets active, this is considered as a failure. If such a failure happens, the next phase can be handled in the same way, perhaps with another selected weight w_h instead of w_j. This failure event decreases the success probability of one phase at most by a factor of e^{-1}.

We want to compare M_1' with M_2'. In particular, we want to show that M_1' has a larger tendency to increase the number of ones in the k-prefix. However, this is not necessarily true if at least three bits of the k-prefix flip. Replacing 110 with 001 may increase the f-value while the inverse step decreases the f-value. Therefore, a step with at least three flipping prefix bits is considered as a failure. The failure probability for one step equals $\binom{k}{3}n^{-3} \leq d^3n^{-3}$ and the failure probability for one phase is bounded above by

$$c'n2^d d^{-1} d^3 n^{-3} \leq c' d^2 n^{-1} = o(1),$$

since $d \leq \log n$.

Let M_1'' and M_2'' be the Markoff chains M_1' and M_2', respectively, under the assumption that within one phase there is no step with at least three flipping prefix bits. The success probability of M_1'' and M_2'' compared with the success probability of M_1' and M_2', respectively, is decreased at most by a factor of $1-o(1)$. Let $p_1(m, m+d), d \in \{-2, -1, 0, +1, +2\}$, be the transition probabilities of M_1'' on the state space $\{0, \ldots, k\}$ and $p_2(m, m+d)$ the corresponding values of M_2''. Then

(1). $$p_1(m, m+d) \leq p_2(m, m+d), \text{ if } d \neq 0$$

The reason is that M_2'' accepts each new string. Moreover,

(2). $$p_2(m, m+d)e^{-1} \leq p_1(m, m+d), \text{ if } d > 0$$

This can be proved in the following way. Since at most two prefix bits are flipping, the number of ones in the prefix increases only if all flipping prefix bits flip from 0 to 1. If furthermore no suffix bit flips (probability at least e^{-1}), the new string is accepted by M_1''. Finally, we have to prove a tendency of M_1'' of increasing the ones in the prefix (in comparison to M_2''). We claim that

$$(3) \qquad \frac{p_1(m, m+d)}{p_2(m, m+d)} \geq \frac{p_1(m, m-d)}{p_2(m, m-d)}, \text{ if } 0 \leq m-d \leq m+d \leq k$$

Let us consider $a^m = (b^m, c^m), m \leq k$. Let $\overline{c}^m$ be any suffix. If M_1 accepts the mutated string $(b_-^m, \overline{c}^m)$ where b_-^m is obtained from b^m by flipping one (or two) ones into zeros, then M_1 accepts also $(b_+^m, \overline{c}^m)$ where b_+^m is obtained from b^m by flipping one (or two) zeros into ones. Inequality (3) follows, since M_2 accepts $(b_+^m, \overline{c}^m)$ and $(b_-^m, \overline{c}^m)$.

The Markoff chain M_2'' can be easily analyzed using the methods of Garnier, Kallel, and Schoenauer (1999) and its generalization in the proof of Lemma 1. The Markoff chain M_1'' has the same asymptotic behavior. Inequality (1) shows that M_1'' may stay longer in some state than M_2''. However, Inequality (2) shows that the probabilities of going to a larger state are for M_1'' at most by a constant factor smaller than for M_2''. Hence, the effect of staying longer in the same state has not a big influence. Inequality (3) is the most important one. It shows that the probability of increasing the state from m to $m+d$ within one step may be decreased for M_1'' compared to M_2''. However, then the probability of decreasing the state from m to $m-d$ within one step has been decreased at least by the same factor. This implies that the expected number of active steps (changing the state) is for M_1'' smaller than for M_2''. However, the proof of this claim needs a careful analysis of the Markoff chain M_1'' which is omitted here. By Markoff's inequality, we can choose a constant c' such that the success probability of M_1'' within one phase of length $c'n2^d/d$ is at least $1/2$. This implies by our considerations that the success probability of M_1 within such a phase is at least $1/(2e) - o(1)$ which proves the theorem. □

We emphasize one main difference between the analysis of general randomized search heuristics and problem-specific algorithms. Most of the problem-specific algorithms are designed with respect to efficiency and also with respect to the aim of analyzing the algorithm. For monotone polynomials the randomized hill-climber flipping in each step exactly one random bit is not less efficient but much easier to analyze than the $(1+1)$EA. However, this hillclimber has disadvantages for other functions. It gets stuck in each local maxima while the $(1+1)$EA can escape efficiently from a local maximum if a string with at least the same f-value and short Hamming distance to the local maximum exists.

4 The $(1+1)$EA on affine functions and royal road functions

Theorem 1 cannot be improved with respect to d, see Lemma 1 for the case $N = 1$. However, our analysis seems to be not optimal for large N. In order to leave the fitness layer L_i we have waited until one specific passive weight is turned into active. We also may leave L_i, since other weights get active. Moreover, monomials can be correlated positively, e.g., $f(x) = 2x_1x_2 \cdots x_d + x_2x_3 \cdots x_{d+1}$. It takes some time to leave L_0, since we have to activate the first monomial. Afterwards, no step flipping one of the first d bits which all are 1 is accepted. Hence, the expected time to activate the second monomial is only $O(n)$. Because of the monotonicity of the polynomials different monomials cannot be correlated negatively. One may think that the case of independent monomials is the worst case.

Let $n = dm$. We consider monotone polynomials with m monomials with non-vanishing weights. All monomials are of degree d and depend on disjoint sets of variables. The special case of weights 1 is known as royal road function (Mitchell, Forrest, and Holland (1992)), since it has been assumed that these functions are difficult for all evolutionary algorithms without crossover and easy for genetic algorithms (which are based on crossover).

Theorem 2. *The expected optimization time of the* $(1+1)$*EA on royal road functions of degree* d *is bounded by* $O(n(\log n)2^d/d)$.

Sketch of Proof. First, we consider m independent functions each consisting of one of the monomials of the royal road function with degree d. By Lemma 1 and Markoff's inequality, there is a constant c such that the success probability for a single monomial within $cn2^d/d$ steps is at least $1/2$. The success probability after $\lceil \log n \rceil + 1$ of such phases is at least $1 - 1/(2n)$ and, therefore, the probability that all monomials are optimized is at least $1/2$. This leads to the proposed upper bound in the scenario of m independent monomials. However, the $(1+1)$EA considers the m monomials in parallel. This causes only small differences. Steps where more active monomials are deactivated than passive monomials are activated are not accepted and monomials may be deactivated if enough passive monomials are activated. It is not difficult to prove that this increases the expected optimization time at most by a constant factor. □

A different proof method for Theorem 2 has been presented by Mitchell, Holland, and Forrest (1994). The result shows that for royal road functions there is not much room for improvements by crossover. We have seen in Section 3 that problem-independent search heuristics cannot be successful on the average with less than $(2^d + 1)/2$ steps. In particular, the improvement by any general search heuristic is bounded by a polynomial factor of $O(n(\log n)/d)$. The situation gets more difficult in the case of different weights. Then it is possible that more monomials get deactivated than activated. In a certain sense we may move far away from the optimum. This situation has been handled only in the case of affine functions, i.e., polynomials of degree 1. In this case it is not necessary to assume non-negative weights, since x_i can be replaced with $1 - x_i$.

Theorem 3. *The expected optimization time of the* $(1+1)$*EA on affine functions is bounded by* $O(n \log n)$. *It equals* $\Theta(n \log n)$ *if all* n *degree-1 weights are non-zero.*

Idea of Proof. The full proof by Droste, Jansen, and Wegener (2001) is involved and long. W.l.o.g. $w_\emptyset = 0$ and $w_1 \geq \cdots w_n \geq 0$ where $w_i := w_{\{i\}}$. The main idea is to measure the progress of the $(1+1)$EA with respect to the "generic" affine function

$$g(x_1, \ldots, x_n) := 2 \sum_{1 \le i \le n/2} x_i + \sum_{n/2 < i \le n} x_i.$$

This function plays the role of a potential function in the analysis of data structures and algorithms. Then successful steps ($x' \neq x$ and $f(x') \geq f(x)$ for the given affine function f) are distinguished from unsuccessful steps. The main step is to prove an upper bound of $O(1)$ on the expected number of successful steps to increase the g-value (not the f-value) of the current string. The bound on the number of unsuccessful steps follows then easily. Since the $(1+1)$EA accepts a string according to its f-value, it is possible that the g-value decreases. The idea is to design a slower Markoff chain where the g-value increases in one step by not more than 1 and where the expected gain of the g-value within one successful step is bounded below by a positive constant. Then a generalization of Wald's identity on stopping times can be proved and applied.

The lower bound is an easy application of the coupon collector's theorem. □

Up to now we were not successful to generalize this bound to monotone degree-2 polynomials. Nevertheless, we state the following conjecture.

Conjecture 1. The expected optimization time of the $(1+1)$EA on monotone polynomials of degree d is bounded by $O(n(\log n)2^d/d)$.

5 Further results on the $(1+1)$EA and its generalizations

In Sections 3 and 4, we have tried to present typical methods for the analysis of the $(1+1)$EA by investigating and analyzing monotone polynomials. Wegener (2000) presents an overview on more methods. Here we mention shortly further directions of the research on the $(1+1)$EA and its generalizations. Droste, Jansen, and Wegener (1998) have investigated the behavior of the $(1+1)$EA on so-called unimodal functions, where each non-optimal string has a better Hamming neighbor. In particular, they have disproved that the $(1+1)$EA has a polynomial expected optimization time on unimodal functions. Wegener and Witt (2001) have shown for some special degree-2 polynomials and all squares of affine functions that they are easy for the multi-start variant of the $(1+1)$EA, although some of them are difficult for the $(1+1)$EA. When optimizing a single monomial $x_1 x_2 \cdots x_d$ we are exploring for a long time the plateau of strings of fitness 0 and it would be less efficient to accept only strict improvements. Jansen and Wegener (2000b) investigate the problem of exploring plateaus more generally. They also show that it is sometimes much better to accept only strict improvements. It has been conjectured that the mutation probability $1/n$ is at least almost optimal for the $(1+1)$EA and each f. This has been disproved by Jansen and Wegener (2000a) who also have shown that it can be even better to work with a dynamic $(1+1)$EA which changes its mutation probability following a fixed schedule. This dynamic variant is analyzed for many functions by Jansen and Wegener (2001b). Further strategies to change the mutation probability are discussed by Bäck (1998).

6 A generic genetic algorithm

Evolutionary algorithms based on selection and mutation only are surprisingly successful. Genetic algorithms are based on selection, mutation, and crossover and there is a community believing that crossover is the essential operator. The main variants of crossover for $(a, b) \in \{0,1\}^n \times \{0,1\}^n$ are

- one-point crossover (choose $i \in \{1, \dots, n-1\}$ randomly and create the offspring $(a_1, \dots, a_i, b_{i+1}, \dots, b_n)$) and
- uniform crossover (choose $c \in \{0,1\}^n$ randomly and create the offspring $d = (d_1, \dots, d_n)$ where $d_i = a_i$, if $c_i = 0$ and $d_i = b_i$, if $c_i = 1$).

In order to apply crossover we need a population of size larger than 1. The main problem is to combine fitness-based selection with the preservation of enough diversity such that crossover has a chance to create strings different from those in the population. In the following it is sufficient to require that selection chooses x with at least the same probability as x' if $f(x) \geq f(x')$. This implies the same selection probabilities for x and x' if $f(x) = f(x')$. Many genetic algorithms replace a population within one step with a possibly totally different new population. It is easier to analyze so-called steady-state genetic algorithms where in each step only one offspring is created and perhaps exchanged with one member from the current population.

Algorithm 2 (Steady-state GA).

1) Initialization: The $s(n)$ members of the current population are chosen randomly and independently.
2) Branching: With probability $p_c(n)$, the new offspring is created with crossover (Steps 3.1, 3.2, 3.3) and with the remaining probability, the new offspring is created without crossover (Steps 4.1, 4.2).

3.1) Selection for crossover and mutation: A pair of strings (x, y) from the current population is chosen.
3.2) Crossover: z' is the result of crossover on (x, y).
3.3) Mutation: z is the result of mutation of z'. Go to Step 5.
4.1) Selection for mutation: A string x from the current population is chosen.
4.2) Mutation: z is the result of mutation of x.

5) Selection of the next generation: Add z to the current population and let W be the multi-set of strings in the enlarged population which have the minimal f-value and let W' be the set of strings in W which have the largest number of copies in W. Eliminate randomly one string from W' from the population to obtain the new population.
6) Continue at Step 2 (until some stopping criterion is fulfilled).

The analysis of genetic algorithms is even more difficult than the analysis of evolutionary algorithms without crossover. Although the crossover operator is in the focus of research since fourty years, there was no example known where crossover decreases the expected optimization time from exponential to polynomial. Experiments (Forrest and Mitchell (1993)) show that the $(1+1)$EA is for the royal road functions even faster than genetic algorithms. Watson (2000) presents a function where crossover probably helps. This is established by experiments and by a proof under some assumptions but not by a rigorous proof.

7 Real royal road functions and the crossover operator

Jansen and Wegener (2001a) present the first example where crossover provably decreases the expected optimization time from exponential to polynomial. Because of the history and the many discussions on the royal road functions they have called their functions real royal road functions. For $a \in \{0,1\}^n$ let $|a| = a_1 + \cdots + a_n$ and let $b(a)$ be the block size of a, i.e. the length of the longest block consisting of ones only (the largest l such that $a_i = a_{i+1} = \cdots = a_{i+l-1} = 1$ for some i). Then

$$R_{n,m}(a) := \begin{cases} 2n^2 & \text{if } a = (1,1,\ldots,1) \\ n|a| + b(a) & \text{if } |a| \leq n - m \\ 0 & \text{otherwise.} \end{cases}$$

For a proof of the following lemma see Jansen and Wegener (2001a).

Lemma 2. *Evolutionary algorithms without crossover need with a probability exponentially close to* 1 *exponentially many steps to optimize the real royal road function* $R_{n,\lceil n/3 \rceil}$ *and with a probability of* $1 - n^{-\Omega(\log n)}$ *superpolynomially many steps to optimize* $R_{n,\lceil \log n \rceil}$.

Theorem 4. *Let* $s(n) = n$, $m = \lceil n/3 \rceil$ *and* p_c *a positive constant less than 1. Then the expected optimization time of the steady-state GA with one-point crossover on* $R_{n,m}$ *is bounded by* $O(n^4)$.

Sketch of Proof. Here we use the proof technique to describe intermediate aims and to estimate the expected time until the aim is reached. The advantage is that we can use afterwards the assumption that the last aim has been reached.
Aim 1: All strings of the population have exactly $n - m$ ones or we have found the optimum.

This aim is reached in an expected number of $O(n^2)$ steps. It is very unlikely to start with strings with more than $n - m$ and less than n ones. The expected time to eliminate all these strings is $O(1)$. If we then do not find the optimum, we only have an expected waiting time of $O(n/m) = O(1)$ to increase the number of ones in the population. This is due to steps with mutation only. If the selected string has less than $n - m$ ones, there is a good chance to increase the number of ones by a 1-bit mutation. If the selected string has exactly $n - m$ ones, there is a good chance to produce a replica.
Aim 2: All strings of the population have exactly $n - m$ ones and a block size of $n - m$ or we have found the optimum.

This aim is reached in an expected number of $O(n^3 \log n)$ steps. If we do not find the optimum, we only have to increase the sum of the block lengths of the strings of the current population. If not all strings have the same block length, it is sufficient to produce a replica of a string with a non-minimal block length. Otherwise, certain 2-bit mutations increase the block length.

Aim 3: All strings of the population have exactly $n-m$ ones, a block size of $n-m$, and each of the $m+1$ different strings with this property is contained in the population or we have found the optimum.

This aim is reached in an expected number of $O(n^4)$ steps. If we do not find the optimum, there is always at least one string in the current population such that a 2-bit mutation creates a string with $n-m$ ones and block size $n-m$ which was not in the population before.

Aim 4: The optimum is found.

This aim is reached in an expected number of $O(n^2)$ steps. This is the only phase where crossover is essential. With a probability of at least $p_c(n)/n^2$ crossover is chosen as search operator and $1^{n-m}0^m$ and 0^m1^{n-m} are selected. Then, with a probability of at least $1/3$, one-point crossover creates 1^n and finally, with a probability of at least e^{-1}, mutation preserves 1^n and we have found the optimum. □

Uniform crossover is less efficient for these functions. The probability of creating 1^n from $1^{n-m}0^m$ and 0^m1^{n-m} is only 2^{-2m}. This leads to a polynomial expected optimization time only if $m = O(\log n)$. Hence, crossover reduces the expected optimization time for some functions only from superpolynomial to polynomial. Jansen and Wegener (2001a) have presented a more complicated function where uniform crossover decreases the expected optimization time from exponential to polynomial.

One may ask what happens if we replace in the definition of $R_{n,m}$ the value of $b(a)$ by 0. Then the size of the plateau of the second-best strings increases from $m+1$ to $\binom{n}{m}$ and it is much harder to generate enough diversity. Jansen and Wegener (1999) have investigated this function. With uniform crossover and the very small crossover probability $p_c(n) = 1/(n \log^3 n)$ they could prove a polynomial expected optimization time for $m = O(\log n)$. This proof is technically much more involved than the proof of Theorem 4 and its counterpart for uniform crossover. Altogether, we have only made the first steps of analyzing genetic algorithms with crossover.

Conclusions

We have argued why one should investigate and analyze different forms of randomized search heuristics, among them evolutionary algorithms. The differences in the analysis of problem-specific algorithms and general search heuristics for black box optimization have been discussed. Then our approach has been presented by analyzing some evolutionary algorithms on subclasses of the class of monotone polynomials and by proving for the first time that crossover can decrease the expected optimization time significantly.

References

1. Bäck, T. (1998). An overview of parameter control methods by self-adaptation in evolutionary algorithms. Fundamenta Informaticae 32, 51–66.
2. Bäck, T., Fogel, D. B., and Michalewicz, Z. (Eds.) (1997). *Handbook of Evolutionary Computation.* Oxford Univ. Press, Oxford.
3. Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1990). *Introduction to Algorithms.* MIT Press.
4. Droste, S., Jansen, T., and Wegener, I. (1998). On the optimization of unimodal functions with the (1+1) evolutionary algorithm. Proc. of PPSN V (Parallel Problem Solving from Nature), LNCS 1648, 13-22.
5. Droste, S., Jansen, T., and Wegener, I. (2001). On the analysis of the $(1+1)$ evolutionary algorithm. To appear: Theoretical Computer Science.
6. Fogel, D. B. (1995). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press.
7. Forrest, S., and Mitchell, M. (1993). Relative building-block fitness and the building-block hypothesis. Proc. of FOGA' 1993 (2nd Workshop Foundations of Genetic Algorithms), Morgan Kaufmann.
8. Garnier, J., Kallel, L., and Schoenauer, M. (1999). Rigorous hitting times for binary mutations. Evolutionary Computation 7, 173–203.
9. Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley.
10. Hochbaum, D. S. (Ed.) (1997). *Approxiamtion Algorithms for NP-Hard Problems.* PWS Publ. Co.
11. Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* The University of Michigan Press.
12. Jansen, T., and Wegener, I. (1999). On the analysis of evolutionary algorithms – a proof that crossover really can help. Proc. of ESA'99 (European Symp. on Algorithms), LNCS 1643, 184–193.
13. Jansen, T., and Wegener, I. (2000a). On the choice of the mutation probability for the $(1+1)$EA. Proc. of PPSN VI (Parallel Problem Solving from Nature), LNCS 1917, 89–98.
14. Jansen, T., and Wegener, I. (2000b). Evolutionary algorithms – how to cope with plateaus of constant fitness and when to reject strings of the same fitness. To appear: IEEE Trans. on Evolutionary Computation.
15. Jansen, T., and Wegener, I. (2001a). Real royal road functions – where crossover provably is essential. To appear: GECCO'2001.
16. Jansen, T., and Wegener, I. (2001b). On the analysis of a dynamic evolutionary algorithm. Submitted: ESA'2001.
17. Mitchell, M., Forrest, S., and Holland, J. H. (1992). The Royal Road function for genetic algorithms: Fitness landscapes and GA performance. Proc. of 1st European Conf. on Artificial Life, 245–254, MIT Press.
18. Mitchell, M., Holland, J. H., and Forrest, S. (1994). When will a genetic algorithm outperform hill climbing. In J. Cowan, G. Tesauro, and J. Alspector (Eds.): Advances in Neural Information Processing Systems. Morgan Kaufman.
19. Motwani, R., and Raghavan, P. (1995). *Randomized Algorithms.* Cambridge Univ. Press.
20. Rabani, Y., Rabinovich, Y., and Sinclair, A. (1998). A computational view of population genetics. Random Structures and Algorithms 12, 314–334.
21. Schwefel, H.-P. (1995). *Evolution and Optimum Seeking.* Wiley.

22. Watson, R. A. (2000). Analysis of recombinative algorithms on a non-separable building-block problem. Proc. of FOGA'2000 (6. Workshop Foundations of Genetic Algorithms), to appear.
23. Wegener, I. (2000). On the expected runtime and the success probability of evolutionary algorithms. Proc. of WG'2000 (26. Workshop on Graph-Theoretic Concepts in Computer Science), LNCS 1928, 1–10.
24. Wegener, I., and Witt, C. (2001). On the analysis of a simple evolutionary algorithm on quadratic pseudo-boolean functions. Submitted: Journal of Discrete Algorithms.