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

A Natural and Simple Function Which is Hard For All Evolutionary Algorithms

Stefan Droste Thomas Jansen Ingo Wegener

No. CI-93/00

Technical Report ISSN 1433-3325 August 2000

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.

A Natural and Simple Function Which is Hard for All Evolutionary Algorithms*

Stefan Droste **Thomas Jansen** **Ingo Wegener**
LS Informatik 2
Univ. Dortmund
44221 Dortmund Germany
{droste, jansen, wegener}@ls2.cs.uni-dortmund.de

Abstract

Evolutionary algorithms (EAs) are randomized search strategies which have turned out to be efficient for optimization problems of quite different kind. In order to understand the behavior of EAs, one also is interested in examples where EAs need exponential time to find an optimal solution. Until now only artificial examples of this kind were known. Here an example with a clear and simple structure is presented. It can be described by a short formula, it is a polynomial of degree 3, and it is an instance of a well-known problem, the theoretically and practically important MAXSAT problem.

1 Introduction

Evolutionary algorithms (EAs) are randomized search heuristics often used for optimization. There is not a single EA but a class of search strategies, see [3], [5], and [12]. A lot of successful experiments with EAs have been reported and theoretical studies are gaining more and more attention. Their aim is to determine the behavior of different variants of EAs on different classes of functions. The behavior of an algorithm on a function is described by the expected time until an (almost) optimal solution is found and by the probability of doing so within a given time.

All functions which have been proved to be difficult for EAs are somehow "artificial". Hence, it is an interesting problem to present a "non-artificial" or natural and simple function, which is difficult for EAs. In Section 2, we discuss properties of functions which make them natural and simple. In Section 3, we present such a function which is claimed to be difficult for EAs. In order to prove such a claim we have to describe the class of considered EAs (Section 4). In Section 5, we prove that mutation-based EAs fail on our example with overwhelming probability. In Section 6, EAs based on mutation and crossover are investigated. We finish with some conclusions.

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

2 Natural and Simple Functions

We restrict ourselves to functions $f_n\colon \{0,1\}^n \to \mathbb{Z}$ which have to be maximized in a black-box scenario. It is known [2] that functions f_n suitable for black-box optimization have to fulfill the following conditions: The evaluation of f_n has to be possible in polynomial time (with respect to n). The function f_n has to be representable by polynomial-size hardware. The function f_n has to have a compact representation, more precisely small Kolmogoroff complexity (for the theory of Kolmogoroff complexity see [8]). These all are necessary conditions. Moreover, we look for functions which are instances of an important, well-known, and fundamental problem with the additional property that the chosen values for the free parameters of the problem are not artificial. Finally, each function $f_n\colon \{0,1\}^n \to \mathbb{Z}$ can be written as a unique polynomial with respect to the variables $x_1, \dots, x_n$, whose degree is bounded by n. Polynomials of small degree may be called simple. After all, it should be clear that there is no precise definition of natural and simple functions but we now have a lot of reasonable criteria to argue why some functions are natural and simple and others are not. Obviously, all linear functions, i. e., polynomials of degree 1, are simple and they are natural if the coefficients are not artificial.

The most famous functions which are hard for EAs are the functions called needle-in-the-haystack. The function $\mathrm{HAY}_{a,n}$, $a \in \{0,1\}^n$, equals 1, if its argument is a, and 0 otherwise. These functions can be evaluated in linear-time and have a compact representation but they are not instances of an important or fundamental problem. Their degree is n. The functions $\mathrm{HAY}_{a,n}$ can be optimized easily if a is known. Otherwise, all search strategies have to search more or less blindly. Either one has found the optimal input a or one has seen some points $a_1, \ldots, a_m$ and has only learned that the optimal point a is not among $a_1, \ldots, a_m$. Hence, all search strategies have to evaluate on average at least $2^{n-1} + 1/2$ points to find the optimum. This implies that the consideration of the needle-in-the-haystack functions does not have implications on the quality of search strategies.

Another well-known example of a function being hard for EAs is the function TRAP_n, which equals $\mathrm{ONEMAX}_n(x) = x_1 + \cdots + x_n$ for all inputs $x \in \{0,1\}^n$ except for $x = (0, \ldots, 0)$, where TRAP_n equals $n+1$. The function TRAP_n is of degree n, although it differs only at one point from the trivial linear function $\mathrm{ONEMAX}_n(x)$. As long as a search strategy does not produce the all zero string, by chance, it cannot distinguish TRAP_n from ONEMAX_n. We expect from a good search strategy that it quickly finds the all one string, which is optimal for ONEMAX_n but suboptimal for TRAP_n. To replace the all one string by any $a \in \{0,1\}^n$, we replace x_i by $\overline{x}_i$ for all i where $a_i = 0$. This variant $\mathrm{TRAP}_{a,n}$ has similar properties and its global optimum at a. Each EA has only a tiny chance to optimize $\mathrm{TRAP}_{a,n}$ in subexponential time. Again, we conclude that these functions cannot be called natural.

It would be easy to go on in this way and to present artificial functions where EAs are not efficient. One such example is a function called $\mathrm{LONGPATH}_n$ [1] which is unimodal and nevertheless hard for EAs. It is more interesting to look for a function which fulfills all our criteria for natural and simple functions and which nevertheless is hard for EAs.

3 A Special Natural and Simple Function

The function which we shall investigate intensively is an instance of one of the best known optimization problems namely the MAXSAT problem. We define all necessary notions. A literal is a Boolean variable x_i or a negated Boolean variable $\overline{x}_i$. A literal x_i resp. $\overline{x_i}$ is satisfied by an input $a = (a_1, \ldots, a_n)$ iff $a_i = 1$ resp. $a_i = 0$. A clause is a disjunction (Boolean OR) of some literals, i.e., a clause is satisfied by an input a iff at least one of its literals is satisfied. An instance of the MAXSAT problem is a sequence of clauses $c_1, \ldots, c_m$ over the variables $x_1, \ldots, x_n$ and the task is to find an input satisfying as many clauses as possible. Hence, we implicitly also have to decide whether all clauses can be satisfied simultaneously which is the SAT problem, the first problem which ever has been proved to be NP-complete. The SAT problem has a lot of applications, e.g., the verification problem. A lot of other problems can be described easily as MAXSAT problems. It is well-known [4] that MAXSAT is NP-hard even if we only allow clauses with at most two literals.

Nevertheless, it is still possible that instances of MAXSAT with a simple structure also can be solved easily by EAs. The following instance has been presented by [9]. It consists of the clauses

- x_i, $1 \le i \le n$, and
- $x_i \vee \overline{x}_j \vee \overline{x}_k$, $(i,j,k) \in \{1, \ldots, n\}^3$, $i \ne j \ne k \ne i$.

This instance has a lot of interesting features. All clauses are so-called Horn clauses, i.e., they have at most one positive literal. Such clauses correspond to typical database queries. Because of its symmetric description the instance has a simple and clear structure and, moreover, the reader "sees" the solution within a second: the all one string satisfies all clauses and it is the only input with this property. Nevertheless, the following well-known search strategy which has been designed especially for the MAXSAT problem takes expected exponential time on this instance [9].

The search strategy starts with a random input. If not all clauses are satisfied it chooses randomly an unsatisfied clause and one of its literals and flips the value of the corresponding variable. This can be seen as a specialized strategy based on mutations which takes care of the problem type. The (with respect to expected worst case time) best known algorithm for MAXSAT problems with clauses of at most three literals is due to [11]. It is a multistart variant of the above strategy and also takes exponential time for the above example. The reason is the following. If we choose the clause $x_i \vee \overline{x}_j \vee \overline{x}_k$, it is more likely to flip an input bit from 1 to 0 as vice versa. We may expect that EAs have the same tendency.

For our later discussions of EAs we translate our MAXSAT instance into a polynomial $\mathrm{COUNTSAT}_n\colon \{0,1\}^n \to \mathbb{R}$ where $\mathrm{COUNTSAT}_n(a)$ is the number of clauses satisfied by a. Then $\mathrm{COUNTSAT}_n(x)$ is

$$\sum_{1 \le i \le n} x_i + \sum_{1 \le i \le n} \sum_{\substack{1 \le j \le n \\ j \ne i}} \sum_{\substack{1 \le k \le n \\ k \ne i, k \ne j}} (1 - (1 - x_i) x_j x_k).$$

Because of the symmetry we obtain a simpler description $\mathrm{COUNTSAT}^*_n\colon \{0, \ldots, n\} \to \mathbb{R}$ defined on $s =$

$x_1 + \cdots + x_n$. Since $(1-(1-x_i)x_jx_k) = 1 - x_jx_k + x_ix_jx_k$, $\mathrm{COUNTSAT}_n(x)$ can be written as

$$\begin{aligned} & s + n(n-1)(n-2) - 2(n-2) \sum_{1\le j<k\le n} x_jx_k \\ & +6 \sum_{1\le i<j<k\le n} x_ix_jx_k \\ = \quad & s + n(n-1)(n-2) - 2(n-2)\binom{s}{2} + 6\binom{s}{3} \\ = \quad & \mathrm{COUNTSAT}^*_n(s). \end{aligned}$$

$\mathrm{COUNTSAT}_n$ is a fitness function fulfilling all properties of a natural and simple function. The evaluation of $\mathrm{COUNTSAT}_n$ is possible in polynomial time $O\left(n^3\right)$ (with the description as $\mathrm{COUNTSAT}^*_n$ even in linear time $O\left(n\right)$) and the function has a very compact representation. The function is an instance of the fundamental, important, and well-known MAXSAT problem, the parameters of the function are simple, small and symmetric with respect to all variables. Finally, the fitness function is a polynomial of degree 3 only.

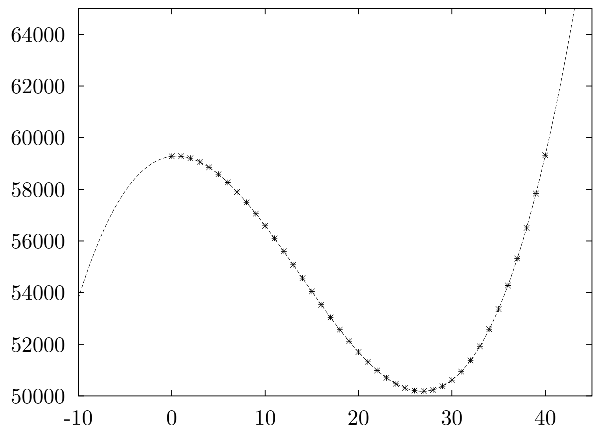

Fig. 1: The function $\mathbf{COUNTSAT}^*_{40}$.

The function $\mathrm{COUNTSAT}^*_{40}$ is shown in Figure 1, where the range of the function is extended to $[-10, 45]$. As $\mathrm{COUNTSAT}^*_n(0) = n(n-1)(n-2)$ the all zero string is a local optimum, while $\mathrm{COUNTSAT}^*_n(n) = n(n-1)(n-2)+n$ and the all one string is the only global optimum for $\mathrm{COUNTSAT}_n$. It is easy to see that $\mathrm{COUNTSAT}^*_n$ has its local minimum for $s \approx \frac{2}{3}(n+1)$. Random inputs have with overwhelming probability approximately $n/2$ ones. Then local changes decreasing the number of ones lead to better inputs while local changes increasing the number of ones decrease the fitness. Hence, we might expect that EAs quickly find the all zero string and have difficulties to find the all one string.

The function $\mathrm{COUNTSAT}_n$ can be seen as a non-artificial variant of $\mathrm{TRAP}_{1^n,n}$. Both only depend on s. If s is not too large, inputs with less ones are better but the global optimum is the all one string. In both cases EAs should interpret this as a hint to decrease the number of ones and to find the all zero string.

One may argue that exact optimization is not the right aim for black box optimization. The local optimum with value $n(n-1)(n-2)$ is almost as good as the global one with value $n(n-1)(n-2)+n$. Hence, we discuss the variants $\mathrm{COUNTSAT}_{n,r}$ and $\mathrm{COUNTSAT}^*_{n,r}$, $r \in \mathbb{N}$. They are based on the same instance of MAXSAT, but every clause x_i, $1 \le i \le n$, is included r-times. Then $\mathrm{COUNTSAT}^*_{n,r}(s) = s^3 - (n+1)s^2 + (n+r)s + n(n-1)(n-2)$. The increase of the parameter r has many effects. The function $\mathrm{COUNTSAT}^*_{n,r}$ has a local maximum and a local minimum at the positions

$$\left.\begin{array}{l} s_{max}(r) \\ s_{min}(r) \end{array}\right\} = \frac{1}{3}(n+1) \genfrac{}{}{0pt}{}{-}{+} \sqrt{\left(\frac{n+1}{3}\right)^2 - \frac{n+r}{3}}.$$

EAs get good hints if $s_{min}(r) \le n/2$ and even if $s_{min}(r)$ is not sufficiently larger than $n/2$. Hence, we only consider values of r such that $s_{min}(r) \ge (1/2+\varepsilon)n$ which means values of r such that $r \le \lceil (1/4-\varepsilon)(n^2-n)\rceil$ for some constant $0 < \varepsilon < 1/4$. Under this restriction we like to maximize the quotient of the fitnesses of the all one string and of a locally optimal string (which no longer is the all zero string). Let $r = \alpha n^2$. The fitness of the all one string and a locally optimal string equals $\Theta(n^3)$. Hence, we only compare the terms of order n^3. The fitness of the all one string then is $(1+\alpha)n^3$ and the fitness of the locally optimal string is $(1+\beta-\beta^2+\beta^3)n^3$ for $\beta = 1/3 - \sqrt{1/9 - \alpha/3}$. The quotient $(1+\alpha)/(1+\beta-\beta^2+\beta^3)$ is optimal for $\alpha = 0.25$. Then the global optimum is approximately by $9,31\%$ "better" than the local one.

4 Evolutionary Algorithms

It would not be convincing to prove that some special EA fails on $\mathrm{COUNTSAT}_n$. We try to prove this property for a class of EAs. An EA is a combination the modules initialization, selection, mutation, and crossover. We make no assumption how these modules are combined. This leads to statements for a variety of EAs. We assume that EAs are working in rounds called generations where sets of individuals called populations are considered.

The population size is assumed to be at most polynomial with respect to the instance size n. During the initialization the members of the first generation are chosen uniformly at random. Selection is the possibly randomized process to determine the members of the next generation. The selection process is allowed to depend on the individuals only via their fitness values and the property whether the individual is a child or a parent. The chance of individuals to be chosen is positively correlated with the fitness. More precisely, if $f(x) \geq f(x')$ and either x and x' are children or x and x' are parents, the individual x has at least the same chance as x' to be chosen. Usually, the same is true if x is a child and x' is a parent. There may be rules to prevent duplicates. Selection is also the process to choose individuals for mutation and/or crossover.

Mutation is driven by a probability p. If the individual x is chosen for mutation, each bit is flipped independently with probability p. The idea of mutation is to produce randomly small changes. Hence, we assume that $p \leq 1/2$. Of several existing crossover operators we only investigate uniform crossover, since for MAXSAT there are no variables which are more "neighbored" than others. Let x and y be chosen for uniform crossover and let d be the number of positions i where $x_i \neq y_i$. Each z, where $z_j = x_j$ for those j, where $x_j = y_j$, has a probability of 2^{-d} to be the result of a uniform crossover between x and y.

5 Mutation-Based EAs

In this section we analyze EAs based on initialization, selection, and mutation only. The aim is to prove that with overwhelming probability the EA has not produced an optimal individual within $t(n)$ steps where $t(n)$ is growing exponentially but not too quickly. We fix an EA by choosing the population size $S = S(n)$, the mutation probability $p = p(n)$, and the selection scheme. For a point of time $t = t(n)$ we ask for the success probability $p^* = p^*(n)$, that the globally optimal individual has been produced. Since we are interested in (small) upper bounds on p^*, we may change the Markoff process describing the EA in such a way that the success probability increases. The idea is to obtain a Markoff process which is easier to handle. Because the fitness function is symmetric, we can replace each individual with s ones by the string $0^{n-s}1^s$ without influencing the success probability.

By Chernoff's bound [6] the probability that an individual after the random initialization has at least $(1/2 + \varepsilon/2)n$ ones is bounded above by $\exp(-\varepsilon^2 n/6)$. Since $S(n)$ is polynomially bounded, the probability that at least one individual of the first generation has at least $(1/2 + \varepsilon/2)n$ ones is bounded above by $\exp(-\Omega(\varepsilon^2 n))$. If some individual with at least $(1/2 + \varepsilon/2)n$ ones is produced in the initialization, we consider this as success of the EA. Hence, we assume in the following that no individual of the first generation has at least $(1/2 + \varepsilon/2)n$ ones.

The fitness function $\mathrm{COUNTSAT}^*_{n,r(\varepsilon)}$ increases for $s \geq \lceil s_{min}(r(\varepsilon)) \rceil \geq (1/2 + \varepsilon)n$. E. g., the usual (1+1)-EA with $p(n) = 1/n$ finds the global optimum quickly if it has found an individual with enough ones. Therefore, we enlarge the event describing a success. The EA is called successful if it produces at least one individual with at least $(1/2 + \varepsilon)n$ ones. Informally, we believe that the individual $I = 0^{n-s}1^s$ is better for our optimization task than $I' = 0^{n-s'}1^{s'}$, if $s > s'$. Formally, we prove that for I it is at least as likely to obtain by mutation a string with at least s'' ones as for I'. For this reason we compare I and I':

$$\begin{array}{rcl} I & = & \boxed{0 \ldots 0}\boxed{1 \ldots 1}\boxed{1 \ldots 1} \\ I' & = & \underbrace{\boxed{0 \ldots 0}}_{n-s}\underbrace{\boxed{0 \ldots 0}}_{s-s'}\underbrace{\boxed{1 \ldots 1}}_{s'} \end{array}$$

Mutation works in the same way on the first $n - s$ bits and the last s' bits. Independently from this, mutation flips each of the $s - s' > 0$ bits in the middle part independently with probability p. Since $p \leq 1/2$, the probability of flipping at most d bits is at least as large as flipping at least $(s - s') - d$ bits.

For $r = 1$, the fitness function is decreasing with the number of ones (as long as this number is smaller than the local minimum). Then fitness-based selection only can prefer individuals with less ones. Hence, we can assume w. l. o. g. that selection does not depend on the fitness of the individuals. This is the best selection scheme (fulfilling the assumptions from Section 4) if the fitness gives wrong hints. For larger r, the fitness function is increasing for small values of s. We only consider parameters r where $r \leq n^2/4$ which implies that the fitness function is decreasing, if $n/6 \leq s \leq (1/2 + \varepsilon)n$. Fitness-based selection can prevent that individuals with less than $n/6$ ones survive.

We only increase the success probability by replacing each individual with less than $(1/2+\varepsilon/2)n$ ones by an individual with exactly $(1/2+\varepsilon/2)n$ ones. Then we either have a success or we consider only individuals such that the fitness function is decreasing with the number of ones. Then the arguments for $r = 1$ work and we can consider an EA where selection does not depend on the fitness of the individuals.

In the last step, we consider the situation that the EA produces an individual I^* with at least $(1/2 + \varepsilon)n$ ones. This individual has a history (such an approach has been used for the first time in [10]), i. e., there is a sequence $I_0, I_1, \ldots, I^*$ of individuals such that I_0 belongs to the initial population and I_{i+1} is produced from I_i by mutation. Hence, $ones(I_0) = (1/2 + \varepsilon/2)n$, $ones(I_i) \geq (1/2 + \varepsilon/2)n$, and $ones(I^*) \geq (1/2 + \varepsilon)n$. We consider the subsequence starting with the last individual with exactly $(1/2+\varepsilon/2)n$ ones. This sequence is denoted (after renumbering) by $I_0, I_1, \ldots, I_{t^*} = I^*$ where $ones(I_0) = (1/2+\varepsilon/2)n$, $ones(I_i) > (1/2+\varepsilon/2)n$ for $i > 0$, and $ones(I_{t^*}) \geq (1/2 + \varepsilon)n$. Because of the second property individual I_i is produced by mutation from I_{i-1} and not by mutation followed by a replacement as described above.

The strings $I_0, I_1, \ldots, I_{t^*-1}$ altogether contain at least $(1/2 + \varepsilon/2)nt^*$ ones and at most $(1/2 - \varepsilon/2)nt^*$ zeros. We like to estimate the probability that starting with I_0 we get an individual I_{t^*} which is a success. All single bits of all I_i, $i < t^*$, have a chance to be mutated. The mutation probability is p. It is a necessary condition that altogether at least $n\varepsilon/2$ more bits are flipping from 0 to 1 than bits are flipping from 1 to 0. For such a success, it is necessary that at most $nt^*p/2$ ones flip or that at least $nt^*p/2$ zeros flip. Otherwise, the number of bits flipping from one to zero is larger than the number of bits flipping from zero to one. Note, that since ε is a positive constant we have that

$$\frac{nt^*p}{2} \leq (1 - \delta_1) \cdot (1/2 + \varepsilon/2)\, nt^*p$$

for a constant $\delta_1 > 0$, where $(1/2+\varepsilon/2)nt^*p$ is a lower bound on the expected number of bits flipping from one to zero, and

$$\frac{nt^*p}{2} \geq (1 + \delta_2) \cdot (1/2 - \varepsilon/2)\, nt^*p$$

for a constant $\delta_2 > 0$, where $(1/2 - \varepsilon/2)nt^*p$ is an upper bound on the expected number of bits flipping from zero to one. Thus, we can estimate the probability of both events by $\exp(-\Omega(nt^*p))$ (by Chernoff's bounds). If $p = \Omega((t^*n^{1/2})^{-1})$, this probability is exponentially small. If $p = O\left((t^*n^{1/2})^{-1}\right)$, $nt^*p/2 = O\left(n^{1/2}\right)$. In this case, we use the fact that at least $\varepsilon n/2$ zeros have to flip. Again, by Chernoff's bound, this probability is bounded by $\exp(-\Omega(n^{1/2}))$. If the EA produces $\exp(o(n^{1/2}))$ individuals, the success probability still is bounded by $\exp(-\Omega(n^{1/2}))$. Hence, we have:

Theorem 1 *Each mutation-based EA (as defined in Section 4) which produces* $\exp(o(n^{1/2}))$ *individuals has a success probability for* $\mathrm{COUNTSAT}_n$ *(and even for* $\mathrm{COUNTSAT}_{n,r(\varepsilon)}$*, if* $0 < \varepsilon < 1/4$ *is a constant) which is bounded by* $\exp(-\Omega(n^{1/2}))$.

6 On the Effect of Uniform Crossover

The usefulness of crossover has been shown by many experiments. There is also a proof that, for some simple but not natural function, all EAs without crossover take superpolynomial time with overwhelming probability while an EA with crossover only needs polynomial time with overwhelming probability [7]. This effect is not surprising but it is hard to deal with crossover which is an operator creating inter-dependent individuals [10]. Therefore, it is not possible to consider simultaneously a class of different types of EAs with crossover.

Let $x, y \in \{0,1\}^n$ and let z be the random outcome of a uniform crossover between x and y. The strings x and y have a common part and for this part z equals x and y. For the other bit positions, z is a random string. This implies that z is a random string if x and y are random and independent strings. Here we are interested in the number of ones k, l, and m of x, y, and z, resp. W. l. o. g. $k \leq l$. Let c be the number of common ones in x and y. Then there are exactly $k + l - 2c$ positions where x and y differ. The random number m of ones of z given c equals $c + M$ where M is binomially distributed with respect to $k + l - 2c$ and $1/2$. The expected number of ones equals $(k+l)/2$ independently from c but the variance decreases linearly as c increases. If $c = k$, $m \leq l$ and there is no chance to obtain a string with more ones than y. If x and y have no bit in common, we obtain a random string. If k and l are given and x and y are random under this assumption, the expected value of c equals lk/n. Hence, uniform crossover has not the tendency to increase the average number of ones in the population. If the strings x and y are positively (negatively) correlated, the variance of the random number of ones in z is smaller (larger) than for two independent strings x and y.

After these basic considerations we investigate a string x which is the outcome of t combined steps of uniform crossover and mutation. Using a history-based approach there are 2^t ancestors from generation 0 which lead to x. If all these 2^t ancestors are random and independent strings, also x is a random string. Since we altogether do not create more than polynomially many (or $\exp(o(n))$) strings, it is very unlikely to create a string with at least $(1/2+\varepsilon)n$ ones. But an EA uses fitness based selection and works with populations of polynomial size. Hence, among the 2^t ancestors there are a lot which represent the same individual. This implies that mutation influences this individual at each stage in the same way. Uniform crossover for the same pair of individuals leads to the same outcome. Hence, the variety of produced individuals is much smaller and the probability of producing an individual with at least $(1/2+\varepsilon)n$ ones seems to decrease. As we all know, EAs need some fitness based selection to improve variants of blind random search. As long as no individual with at least $(1/2+\varepsilon)n$ ones is produced, fitness based selection can only prefer individuals with less ones (for our $\mathrm{COUNTSAT}_n$ functions) and this decreases the probability of creating many ones in the next step. Both effects, the positive correlation between the individuals because of the limited population size and also because of fitness based selection and the preference of individuals with less ones because of fitness based selection, have the local property to decrease the probability of creating individuals with many ones. These arguments are the basis for a rigorous proof of the following claim.

Claim 2 *Let A be an EA (as defined in Section 4) working on the fitness function* $\mathrm{COUNTSAT}_n$ *(or* $\mathrm{COUNTSAT}_{n,r(\varepsilon)}$*, if $0 < \varepsilon < 1/4$ is a constant). The probability that A produces within t steps an individual with at least $(1/2+\varepsilon)n$ ones is not larger than the probability that a random search algorithm producing the same number of individuals leads to an individual with at least $(1/2+\varepsilon)n$ ones.*

This probability is bounded above by $s \cdot \exp(-\Omega(n))$ if s is the number of produced individuals. Hence, the claim implies that no EA (as defined in Section 4) has more than a tiny chance to optimize $\mathrm{COUNTSAT}_n$ or $\mathrm{COUNTSAT}_{n,r(\varepsilon)}$ in a reasonable time.

7 Conclusions

EAs are randomized search strategies which create new individuals by mutation and/or crossover. The search is essentially guided by fitness based selection. Hence, EAs can be successful only for functions where the fitness values often give good hints where to search for the optimum. Functions like needle-in-the-haystack give no hints at all and functions like the trap function only give wrong hints. These are artificial functions. Here a natural and simple function describing a symmetric instance of the famous and important MAXSAT problem is presented which also has the property that almost all fitness values give wrong hints. Hence, it is the first non-artificial example which is proven to be hard for EAs. The reader should admit that our results are asymptotic ones. Hence, we do not claim that EAs fail on $\mathrm{COUNTSAT}_n$, e. g., for $n = 40$.

References

[1] S. Droste, T. Jansen, and I. Wegener. On the optimization of unimodal functions with the (1+1) ea. In *Parallel Problem Solving from Nature* (*PPSN V*), pages 13–22. Springer, 1998.

[2] S. Droste, T. Jansen, and I. Wegener. Perhaps not a free lunch but at least a free appetizer. In *Proc. Genetic and Evolutionary Computation Conference* (*GECCO 99*), pages 833–839. Morgan Kaufmann, 1999.

[3] D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press, 1995.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W. H. Freeman Company, 1979.

[5] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addision-Wesley, 1989.

[6] T. Hagerup and C. R. Rüb. A guided tour of Chernoff bounds. *Information Processing Letters*, 33:305–308, 1989.

[7] Thomas Jansen and Ingo Wegener. On the analysis of evolutionary algorithms — a proof that crossover really can help. In *Proc. of the 7th Ann. European Symposium on Algorithms* (*ESA '99*), pages 184–193. Springer, 1999.

[8] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications.* Springer, 1993.

[9] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[10] Y. Rabani, Y. Rabinovich, and A. Sinclair. A computational view of population genetics. *Random Structures and Algorithms*, 12(4):314–334, 1998.

[11] Uwe Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *Proc. of the 40th Ann. IEEE Symposium on Foundations of Computer Science* (*FOCS '99*), pages 410–414. IEEE Press, 1999.

[12] Hans-Paul Schwefel. *Evolution and Optimum Seeking.* Wiley, 1995.