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

On the Design and Analysis of Evolutionary Algorithms

Ingo Wegener

No. CI-100/00

Technical Report ISSN 1433-3325 September 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.

On the design and analysis of evolutionary algorithms*

Ingo Wegener

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

Abstract: Evolutionary algorithms are problem-independent randomized search heuristics. It is discussed when it is useful to work with such algorithms and it is argued why these search heuristics should be analyzed just as all other deterministic and randomized algorithms. Such an approach is started by analyzing a simple evolutionary algorithm on linear functions, quadratic functions, unimodal functions, and its behavior on plateaus of constant fitness. Furthermore, it is investigated what can be gained and lost by a dynamic variant of this algorithm. Finally, it is proved that crossover can decrease the run time of evolutionary algorithms significantly.

Keywords: evolutionary algorithms, analysis of algorithms, crossover.

1 Introduction

The history of evolutionary algorithms dates back to the sixties (see Fogel (1985), Goldberg (1989), Holland (1975), or Schwefel (1995), the handbook edited by Bäck, Fogel, and Michalewicz (1997) presents all developments). It is a class of problem-independent randomized search heuristics used for optimization, adaptation, learning, and some other aims. Here we focus on optimization. Randomization is also a major tool for problem-independent algorithms (see Motwani and Raghavan (1995)). Nevertheless, there is almost no contact between people working on evolutionary algorithms and people working on classical randomized optimization algorithms. The reasons are certain misunderstandings. Many people doubt the need for problem-independent search heuristics and argue that problem-dependent algorithms have to better, since they are based on problem-specific knowledge. We discuss these aspects in Section 2. There we present some realistic scenarios where one has to work with problem-independent search heuristics.

The classical research on efficient algorithms combines the design and the analysis of algorithms. There are only few analytical results on evolutionary algorithms comparable to the analytical results on the classical algorithms. The analysis of a problem-independent algorithm working on a specific problem is typically harder than the analysis of problem-specific algorithms which often are designed to support the analysis. We discuss in Section 3 the limited value of often discussed statements on evolutionary algorithms and argue why the analysis of evolutionary algorithms should be performed in the same way as the analysis of all other types of algorithms. In the rest of the paper we present shortly results which have been obtained this way.

In Section 4, we introduce the (1+1)EA, the perhaps simplest evolutionary algorithm which is in many situations surprisingly successful. The next three sections consider the analysis of the (1+1)EA on pseudo-boolean functions $f : \{0,1\}^n \to \mathbb{R}$ in order to prove or to disprove well accepted conjectures. In Section 5, it is shown that the (1+1)EA optimizes linear functions efficiently. However, statements on the efficiency on classes of functions are difficult. In Section 6, two conjectures are disproved namely the conjectures that the (1+1)EA is efficient for all unimodal functions and for all functions of small degree. In Section 7, the behavior of the (1+1)EA and one of its variant on plateaus of constant fitness is investigated. The (1+1)EA works with the

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

mutation probability $1/n$. In Section 8, we analyze a dynamic variant of the (1+1)EA working with alternating mutation probabilities.

The (1+1)EA uses only mutation and selection. Many people believe that crossover is a fundamental operation. Nevertheless, this was proved by experiments only. In Section 9, we present the first proof that crossover can decrease the expected run time for a specific function from superpolynomial to polynomial.

The main purpose of this paper is to convince researchers that evolutionary algorithms are a generic part of the research area of efficient algorithms.

2 Scenarios for general randomized search heuristics

First, we have to define the notion of randomized search heuristics. For some state space S, we look for heuristics to find efficiently good or even optimal points for functions $f : S \to \mathbb{R}$. The main handicap of the search is that the algorithm does not "know" f in advance. It can sample f, i.e., it can choose search points x from S and then evaluate $f(x)$. The further search can be directed by the partial knowledge about f obtained from previous sampling.

Definition 1 *Randomized search heuristic for real-valued functions on S.*
The first search point x_1 is chosen according to a probability distribution on S specified by the heuristic. Then $f(x_1)$ for the actual f is evaluated. The search point x_t is chosen according to a probability distribution which is specified by the heuristic and which may depend on $(x_1, f(x_1), \ldots, x_{t-1}, f(x_{t-1}))$. Then $f(x_t)$ is evaluated. The process is iterated until a stopping criterion is fulfilled.

This definition captures all randomized search heuristics. If an algorithm starts with n randomly chosen search points, this is captured by Definition 1, since the search points x_i, $2 \le i \le n$, can be chosen independently from $(x_1, f(x_1), \ldots, x_{i-1}, f(x_{i-1}))$. The consideration of so-called populations is space efficient and prescribes that the further search process is independent from all sampled search points which are not in the actual population. Figure 1 illustrates the main differences between the classical optimization scenario and the so-called black-box optimization scenario which assumes that the function f to be optimized is hidden in some black-box.

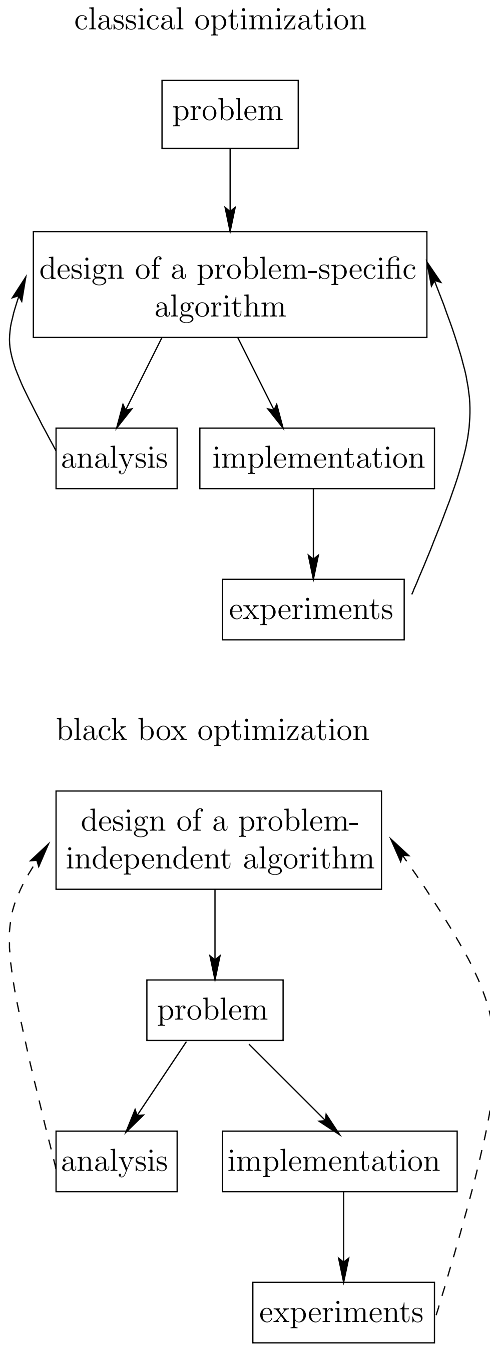


Figure 1: Optimization scenarios.

The design of problem-specific algorithms is influenced by the results of the analysis of the algorithm and by the experimental experience. The analytical results and the experimental experience about a problem-independent algorithm on a specific problem should have not too much influence on the design of this algorithm as a problem-independent one. Only results and experiments on many different problems should have such an influence. Otherwise, the algorithm may turn into a problem-specific one. The question is why we need problem-independent algorithms. If we know a problem, analyze it and use all our knowledge to design a problem-specific algorithm, this should be better than a problem-independent algorithm. Nobody should doubt this. However, there are at least two situations or scenarios where one needs problem-independent algorithms.

Scenario 1 *A problem has to be solved quickly on not too many instances and there are not enough resources (time, money, and/or knowledge) to design a problem-specific algorithm.*

This scenario is realistic in applications. There are not so many experts in designing algorithms that all optimization problems in applications can be attacked by them.

Scenario 2 *The function* $f : S \rightarrow \mathbb{R}$ *is not "known" and one can only "sample"* f.

If one likes to optimize a complicated technical system by fixing some free parameters, it happens that the technical system is so complicated that its input-output behavior cannot be computed and only experiments improve the knowledge about the system.

In black-box optimization the cost is measured by the number of search points which are sampled.

3 How to analyze evolutionary algorithms

First, we introduce the essential modules of EAs (evolutionary algorithms):

- *Selection* is used to decide which search points will be forgotten (the others are the members of the actual population called generation) and it is also used to decide which "individuals" of the "current population" are chosen as "parents".
- *Mutation* and *crossover* produce new individuals called "children" from the parents. Mutation works on one parent and produces usually one child while crossover works on at least two parents (most often exactly two parents) and produces a certain number of children.

Evolutionary algorithms are designed as robust search heuristics with the hope that they are quite efficient on many problems of different type. However, often exaggerated statements like "an evolutionary algorithm is better than all other algorithms on the average of all problems" have been used. The NFL theorem of Wolpert and Macready (1997) shows that such statements are wrong and it is necessary to analyze evolutionary algorithms more carefully. Too general statements tend to be wrong. This is also the case with the *building block hypothesis*. Mitchell, Forrest, and Holland (1992) have presented the so-called *royal-road* functions where evolutionary algorithms with crossover should have big advantages if the building block hypothesis holds. Later they have admitted that a simple evolutionary algorithm outperforms genetic algorithms which rely on crossover (Mitchell, Holland, and Forrest (1994)). The often cited *schema theorem* is a correct but simple result on the behavior within one step. Other one step measures like the *quality gain* and the *progress rate* are well analyzed. The results describe the microscopic behavior, but in many situations this implies almost nothing for the macroscopic behavior. Finally, evolutionary algorithms are analyzed as dynamical systems and therefore implicitly under the assumption of an infinite population. Then the difference between finite and infinite populations has to be investigated carefully. Rabani, Rabinovich, and Sinclair (1995) have started such an approach.

We have chosen another point of view. A randomized search heuristic is nothing but a randomized algorithm and should be analyzed like a randomized algorithm with one distinction. Since f is assumed to be unknown, one never knows when the optimum is reached. Hence, we consider evolutionary algorithms without stopping criterion and investigate random variables X_f measuring the first point of time where on f something nice happens. Here the nice event is the evaluation of an optimal search point and we consider the maximization of f. Our interest (motivated by the requirements of applications) is the behavior within reasonable time bounds and not the limit behavior. Moreover, we are interested in the behavior for typical functions (typical instances for classical algorithms). This leads to bounds on the best and worst case behavior on classes of functions (problems for classical algorithms). Such results can be used to compare different randomized search heuristics.

The obvious problem is to estimate the expected search time $E(X_f)$. This expected value can be exponentially large, although the event "$X_f \leq t(n)$" for small $t(n)$ has a probability which is not too small. Such a result is of special interest, since we have the multi-start option, i.e., many independent runs of the algorithm can be performed in parallel.

4 The simple (1+1)EA

Algorithm 1 *(1+1)EA*

- *choose* $x \in \{0,1\}^n$ *randomly,*
- *let* x' *be the result of a mutation of* x*, i.e., the bits* x'_i *are generated independently and* $x'_i = \overline{x}_i$ *with the mutation probability* $1/n$ *and* $x'_i = x_i$ *otherwise,*
- x *is replaced with* x' *iff* $f(x') \geq f(x)$*,*
- *the last two steps are iterated.*

The notation (1+1)EA means that the population size is 1, the current search point creates one child by mutation and the child replaces its parent iff it is not worse than its parent. One can easily design algorithms with larger populations. However, selection has the tendency to create populations with a small diversity. Hence, one can prove in many cases that p independent runs of the (1+1)EA outperform an evolutionary algorithm with population size p. This will not be discussed here in detail, but we sometimes mention the behavior of multi-start variants.

5 The (1+1)EA on linear functions

Definition 2 *A function* $f : \{0,1\}^n \to \mathbb{R}$ *is called* linear *if* $f(x_1, \ldots, x_n) = w_0 + w_1x_1 + \cdots + w_nx_n$.

For the analysis of the (1+1)EA on linear functions we can assume without loss of generality that $w_0 = 0$ and $w_1 \geq \cdots \geq w_n > 0$. Droste, Jansen, and Wegener (1998a) have proved the following result which has been an open conjecture for several years.

Theorem 1 *The expected run time of the (1+1)EA with non-zero linear weights equals* $\Theta(n \log n)$*.*

In this survey article we only list the main ideas of the proofs. The lower bound follows easily with the Chernoff's bound, showing that the initial point has a large Hamming distance to the optimum, and the coupon collector's theorem, proving a bound on the time until the bits different to the bits of the optimum have tried to flip at least once.

The simpliest case where $w_1 = \cdots = w_n = 1$ has been solved by Mühlenbein (1992). A general upper bound of $O(n^2)$ can be obtained easily. Let

$$A_i = \{x | w_1 + \cdots + w_i \leq f(x) < w_1 + \cdots + w_{i+1}\}.$$

Then it is easy to see that for the Hamming distance H and $i < n$

$$\forall x \in A_i \ \exists y \in A_{i+1} \cup \cdots \cup A_n : H(x,y) = 1.$$

Hence, the expected time to leave A_i is bounded above by $e \cdot n$ leading to the upper bound $e \cdot n^2$.

The proof method for the upper bound of Theorem 1 is best explained for the special case of the binary value function BV defined by

$$\mathrm{BV}_n(x) = x_1 2^{n-1} + x_2 2^{n-2} + \cdots + x_{n-1} 2 + x_n.$$

The crucial fact is that steps increasing the Hamming distance to the optimal point 1^n can be accepted. The leftmost flipping bit decides whether the child is accepted ($0 \to 1$ is accepted, $1 \to 0$ is rejected). There may be many 1-bits to the right which flip to 0. The analysis distinguishes successful steps (x' replaces x and $x' \neq x$) and unsuccessful steps.

We define a Markoff process which is provably slower than the (1+1)EA and we measure the progress with respect to a potential function. The situation is illustrated in Figure 2.

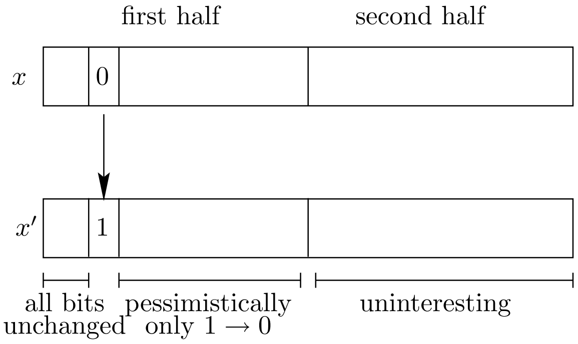


Figure 2: A successful mutation step for BV_n.

The potential function counts the number of ones in the first half. For BV_n, the last half has no influence on the decision whether a step is successful – as long as some bit in the first half flips. A step is successful iff the first flipping bit is a 0. With respect to the potential function we pessimistically assume that all other flipping bits in the first half flip from 0 to 1. We look for an upper bound on the number of successful steps until the number of ones in the first half has increased from i to a larger value. The number of ones is by our assumption a random walk on $\mathbb{Z}$ (in order to obtain a homogeneous random walk, we allow negative numbers). The difference between two consecutive points of time is described by a random variable Y where (by our assumption) $Y \leq 1$ and (as one can prove) $E(Y) \geq \frac{1}{2}$ (here it is essential to consider only one half of the string). By Wald's identity, the number of successful steps until the number of ones has increased is bounded by 2. If the first half consists of ones only, the same analysis works for the second half, if one takes into account that no bit from the first half flips in successful steps. The probability of a successful step is for strings with i ones in the corresponding half bounded below by $e^{-1} \cdot (n-i)/n$ leading to the proposed bound.

The situation of general linear functions is much more difficult. However, a very simple potential function can be used namely

$$v(x) := 2 \cdot (x_1 + \cdots + x_{n/2}) + (x_{n/2+1} + \cdots + x_n).$$

We need a tedious case inspection to control the progress with respect to this pontential function and we also have proved a generalization of Wald's identity.

6 The (1+1)EA on unimodal functions and quadratic functions

The following conjectures had a great influence in the discussion on evolutionary algorithms. First, we introduce the notion of unimodal and quadratic functions.

Definition 3 *i) A function $f : \{0,1\}^n \to \mathbb{R}$ is called* unimodal *if it has a unique global optimum and all other search points have a Hamming neighbor with a larger fitness.*

ii) A function $f : \{0,1\}^n \to \mathbb{R}$ is called quadratic *if*

$$f(x_1, \ldots, x_n) = w_0 \quad + \sum_{1 \leq i \leq n} w_i x_i + \sum_{1 \leq i < j \leq n} w_{ij} x_i x_j.$$

The conjectures claim that the (1+1)EA is efficient on all unimodal and on all quadratic functions. Both conjectures are wrong proving that statements that evolutionary algorithms are efficient on not very "simple" classes of functions are in general not true. The class of unimodal functions includes path functions.

Definition 4 *A path p starting at $a \in \{0,1\}^n$ is defined by a sequence of points $p = (p_0, \ldots, p_m)$ where $p_0 = a$ and $H(p_i, p_{i+1}) = 1$. A function $f : \{0,1\}^n \to \mathbb{R}$ is a path function with respect to the path p if $f(p_{i+1}) > f(p_i)$ for $0 \leq i \leq m-1$ and $f(b) < f(a)$ for all b outside the path.*

Path functions where the fitness outside the path is defined in such a way that the (1+1)EA reaches the path with high probability at the beginning of the path may lead to the conjecture that the (1+1)EA "has to follow" the path which would take an expected time of $\Theta(kn)$. Horn, Goldberg, and Deb (1994) have defined exponentially long paths leading to unimodal functions. However, Rudolph (1997) has shown that these paths allow "shortcuts", i.e., mutation steps where only a few bits flip (here three bits) and where the progress on the path is exponentially large. The so-called long-path functions are optimized by the (1+1)EA in expected time $\Theta(n^3)$. Rudolph (1997) has defined other long-path functions such that for all $i \le n^{1/2}$ each point p_j on the path has only one successor on the path with Hamming distance i, namely p_{i+j}. Hence, shortcuts are only possible by flipping $\Omega(n^{1/2})$ bits simultaneously, an event with an exponentially small probability. Droste, Jansen, and Wegener (1998b) have defined a unimodal function based on this long path and have proved that the (1+1)EA needs an expected time of $\Theta(n^{3/2}2^{n^{1/2}})$ for this function. Additionally, it can be shown that multi-start variants will not lead to subexponential run times.

Theorem 2 *There are unimodal functions where the expected run time of the (1+1)EA is of size* $\Theta(n^{3/2}2^{n^{1/2}})$.

Wegener and Witt (2000) have analyzed the behavior of the (1+1)EA on quadratic functions. In contrast to the class of linear functions, it is not possible to assume without loss of generality that all weights are non-negative. The trick to replace x_i by $\overline{x}_i = 1 - x_i$ does not work, since $w_{ij}x_ix_j$ leads by this replacement to a new term $w_{ij}x_j$ which can make the weight of x_j negative. In particular, there are non-unimodal quadratic functions essentially depending on all variables. With the methods of the $O(n^2)$ bound for linear functions one obtains the following result.

Theorem 3 *Let* $f : \{0,1\}^n \to \mathbb{R}$ *be a quadratic function without negative weights and with* N *non-zero weights. Then the expected run time of the (1+1)EA is bounded by* $O(Nn^2)$.

There is another interesting special case namely the case of squares of linear functions, i.e., quadratic functions which can be written as

$$f(x) = g(x)^2 \text{ where } g(x) = w_0 + w_1x_1 + \cdots + w_nx_n.$$

Again we can assume without loss of generality that $w_1 \ge \cdots \ge w_n > 0$. The case $w_0 \ge 0$ is not different from the case of linear functions. However, the case $w_0 < 0$ is interesting. If $w_1 = \cdots = w_n = 1$ and $w_0 = -(n/2 - 1/3)$, the (1+1)EA finds one of the local optima 0^n and 1^n quickly. Sitting at the suboptimal point 0^n it has to wait for a simultaneous flipping of all bits leading to an expected run time of $\Theta(n^n)$. However, the probability of reaching 1^n within $O(n \log n)$ steps is at least $1/2 - \varepsilon$ and multi-start variants are very efficient. This result can be generalized.

Theorem 4 *The expected run time of the (1+1)EA on some squares of linear functions equals* $\Theta(n^n)$. *The success probability for each square of a linear function within* $O(n^2)$ *steps is bounded below by* $1/8 - o(1)$.

Finally, Wegener and Witt (2000) present a very difficult quadratic function. The function

$$\begin{aligned}\mathrm{TRAP}_n(x_1,\ldots,x_n) = \quad & -(x_1 + \cdots + x_n)\\ & +(n+1)x_1 * \cdots * x_n\end{aligned}$$

has degree n and is very difficult. The fitness function gives hints to go to 0^n. However, 1^n is the global optimum. Using a reduction due to Rosenberg (1975) TRAP_n is reduced to

$$\begin{aligned}\mathrm{TRAP}^*_n(x_1,\ldots,x_n) \quad = \quad & -\sum_{1\le i\le n} x_i + (n+1)x_1x_{2n-2}\\ & -(n+1)\sum_{1\le i\le 2n-2}(x_{n-i}x_{n+i-1}\\ & +x_{n+i}(3-2x_{n-i}-2x_{n+i-1})),\end{aligned}$$

a quadratic function with almost the same properties as TRAP_n. Nevertheless, the analysis of the (1+1)EA on TRAP^*_n is more involved than on TRAP_n.

Theorem 5 *With probability* $1 - 2^{-\Omega(n)}$, *the (1+1)EA needs* $2^{\Omega(n \log n)}$ *steps on the quadratic function* $TRAP^*_n$.

7 Evolutionary algorithms on plateaus of constant fitness

Definition 5 *A plateau for* $f : \{0,1\}^n \to \mathbb{R}$ *is a set* $P \subseteq \{0,1\}^n$ *such that* $f(a) = f(b)$ *for all* $a, b \in P$ *and there is path from* a *to* b *inside* P.

As long as an evolutionary algorithm only finds search points from a plateau, it gets no hints how to direct the search. The $(1+1)$EA searches blindly on the plateau and accepts each string. We are discussing plateaus which are short paths. Is it possible for the (1+1)EA to reach the "end" of such a plateau efficiently? To be more precise, Jansen and Wegener (2000) have defined the function SPP_n (short path as plateau), see Figure 3. We define $||a|| = a_1 + \cdots + a_n$.

Definition 6

$$SPP_n(a) = \begin{cases} 2n & \textit{if } a = 1^n \\ n & \textit{if } a = 1^i 0^{n-i}, i \neq n \\ n - ||a|| & \textit{otherwise.} \end{cases}$$

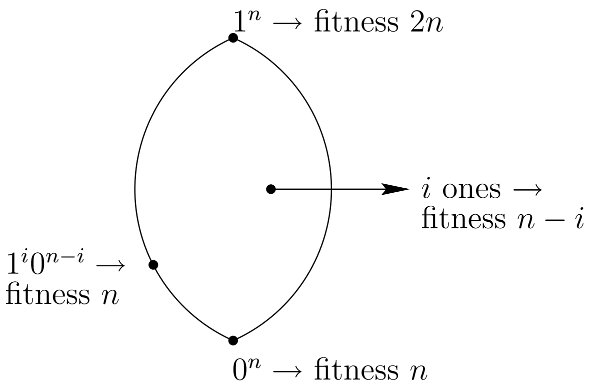


Figure 3: The function SPP_n.

Theorem 6 *The expected run time of the* $(1+1)$*EA on* SPP_n *is bounded by* $O(n^3)$ *and the success probability for* n^4 *steps is* $1 - 2^{-\Omega(n)}$.

It is easy to see that the $(1+1)$EA reaches the plateau with high probability close to 0^n and is far away from the "golden point" 1^n. Then only points from the plateau or 1^n are accepted. The $(1+1)$EA makes a random walk on the plateau like on a bounded interval of $\mathbb{Z}$. A fraction of $\Theta(1/n)$ of the steps is successful, since these steps find another point on the plateau. The random walk is almost symmetric (we cannot jump "behind" 0^n or 1^n). Now we can use results on the gambler's ruin problem to prove that the success probability within cn^2 steps, c large enough, is bounded below by a positive constant. This leads to the bounds of Theorem 6.

For SPP_n, it is essential to investigate the plateau. Let the $(1+1)^*$EA be the variant which accepts x' only if $f(x') > f(x)$. The $(1+1)^*$EA also finds the plateau close to 0^n and then only accepts 1^n. It is quite easy to prove that its expected run time is $2^{\Theta(n \log n)}$ and that the success probability for $n^{n/2}$ steps is only $2^{-\Omega(n)}$. One cannot expect that the $(1+1)^*$EA has any advantage. However, Jansen and Wegener (2000b) have also shown that the $(1+1)^*$EA can prevent that the search process runs into a trap. Their function is called MPT (multiple plateaus with traps).

We define MPT_n informally. Again, the path of all $1^i 0^{n-i}$ plays a central role and 1^n with fitness $3n$ is the unique optimum. Between $1^{n/4} 0^{3n/4}$ and $1^{n/2} 0^{n/2}$ there are $\lfloor n^{1/2}/5 \rfloor$ so-called holes with a distance of at least $\lfloor n^{1/2} \rfloor$. The holes have the fitness 0. All other points $1^i 0^{n-i}$ have the fitness $n+i$. Hence, we have a path with increasing fitness values besides some holes where a jump where two bits have to flip simultaneously has to be performed. The expected run time along this path is $\Theta(n^{5/2})$. Just before a hole, say at $1^i 0^{n-i}$, a plateau consisting of a short path starts. All the points $1^i 0^{n-i-j} 1^j$, $1 \leq j \leq \lfloor n^{1/4} \rfloor - 1$ have the same fitness as $1^i 0^{n-i}$. The final point on this path $1^i 0^{n-i-j} 1^j$, $j = \lfloor n^{1/4} \rfloor$ has the second best fitness value of $2n$. These points are traps, since these points are far away from 1^n and only jumps to 1^n or to other traps are accepted. All other points with i ones have the fitness $n-i$ (see Figure 4).

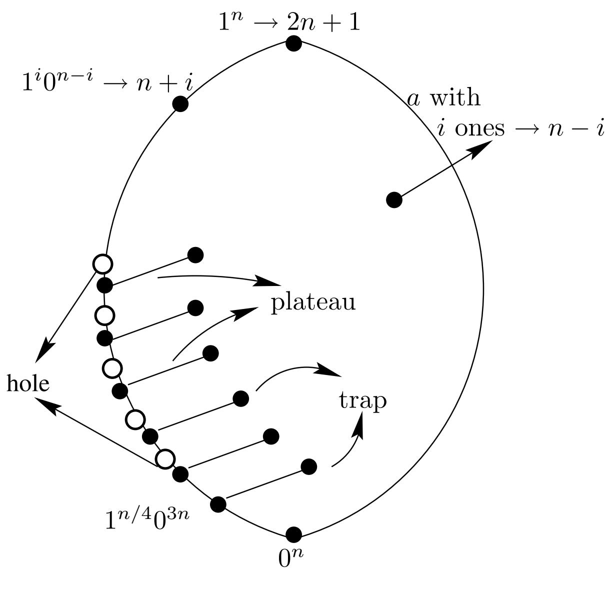


Figure 4: The function MPT_n.

A typical run of the $(1+1)$EA or the $(1+1)^*$EA with a reasonable run time has no step where at least $\Omega(n^{1/2})$ bits flip simultaneously. Therefore, the plateaus and traps can be treated independently. The path is again reached close to 0^n. Both algorithms have a good chance to reach points at the beginning of a plateau. The $(1+1)^*$EA waits there for a better search point. Better strings on the search path are much closer than the traps. Hence, it is very likely that the $(1+1)^*$EA omits all traps. Instead of waiting at the same place for a better string, the $(1+1)$EA explores the plateau. The expected time to reach the trap is only $\Theta(n^{3/2})$, since the plateau length is $\Theta(n^{1/2})$. Hence, there is already a good chance to reach a single trap, and the probability of omitting all traps is exponentially small.

Theorem 7 *The success probability of the $(1+1)$EA on MPT_n within $n^{n/2}$ steps is bounded above by $(1/n)^{\Omega(n^{1/2})}$. The success probability of the $(1+1)^*$EA on MPT_n within $O(n^3)$ steps is bounded below by $1-(1/2)^{\Omega(n^{1/4})}$.*

8 The dynamic $(1+1)$EA

The choice of the mutation probability $1/n$ seems to be a good choice. The average number of flipping bits equals 1 implying that much smaller values seem to be not useful. However, there is a reasonable possibility of flipping more bits. Indeed, the probability of k flipping bits is approximately $1/(ek!)$ (Poisson distribution). One may imagine situations where larger mutation probabilities can help.

If the search space is $\mathbb{R}^n$ one should start with larger steps and one needs smaller steps to approximate a unique optimum. In this scenario, several strategies to adapt the "mutation strength" have been discussed and analyzed (Bäck (1993, 1998), Beyer (1996)). The idea of self-adaptation seems to be less useful in search spaces like $\{0,1\}^n$. Jansen and Wegener (2000a) have proposed and analyzed a simple dynamic variant of the $(1+1)$EA. The first mutation step uses the mutation probability $1/n$. The mutation probability is doubled in each step until a value of at least $1/2$ is reached. A mutation probability of $1/2$ is equivalent to random search. Therefore, we do not allow such large mutation probabilities and start again with a mutation probability of $1/n$. Hence, we have phases of approximately $\log n$ steps where each mutation probability within the interval $[1/n, 1/2]$ is approximately used in one step. If one specified mutation probability is optimal, we may expect that the dynamic $(1+1)$EA is by a factor of $O(\log n)$ worse than the best static one. The reason is that all but one step during a phase are "wasted". Such a behavior can be proved for the function LO_n (leading ones) which measures the length of the longest prefix of the string consisting of ones only.

Theorem 8 *The expected run time of the static (1+1)EA with mutation probability $1/n$ on LO_n equals $\Theta(n^2)$ while the expected run time of the dynamic (1+1)EA equals $\Theta(n^2 \log n)$.*

It is possible to prove for all linear functions and the dynamic (1+1)EA an upper bound of $O(n^2 \log n)$ and for the special case where all weights equal 1 an upper bound of $O(n \log^2 n)$. The dynamic (1+1)EA wastes a factor of $\Theta(\log n)$ on all (short or long) path functions considered in this paper. Jansen and Wegener (2000a) have also presented a function where the dynamic $(1+1)$EA beats all static $(1+1)$EAs and where it would be difficult to find the best mutation probability. The function is called PJ_n (path and jump) and is defined in the following way (see Figure 5).

Definition 7

$$PJ_n(a) = \begin{cases} 3n & \text{if } ||a|| = k \text{ and} \\ & a_1 + \cdots + a_k = 0 \\ 2n - i & \text{if } a = 1^i 0^{n-i}, \\ & 0 \le i < \lfloor n/4 \rfloor \\ n + a_1 + \cdots + a_{\lfloor n/4 \rfloor} & \text{if } ||a|| = \lfloor n/4 \rfloor \\ ||a|| & \text{if } ||a|| < \lfloor n/4 \rfloor \text{ and} \\ & a \text{ does not belong} \\ & \text{to the previous cases} \\ n - ||a|| & \text{otherwise.} \end{cases}$$

where $k = \lceil \log n \rceil$.

Theorem 9 *The optimal static mutation probability equals $(\ln n)/(4 \cdot (\ln 2) \cdot n)$ leading to an expected run time of the $(1+1)$EA of $\Theta(n^{2.361\ldots})$. The expected run time of the dynamic $(1+1)$EA equals $\Theta(n^2 \log n)$.*

The search procedure works as follows. It is rather easy to find the level $\lfloor n/4 \rfloor$, a mutation probability of $1/n$ is good for this purpose. On this level we look for the best point $1^{\lfloor n/4 \rfloor} 0^{n-\lfloor n/4 \rfloor}$.

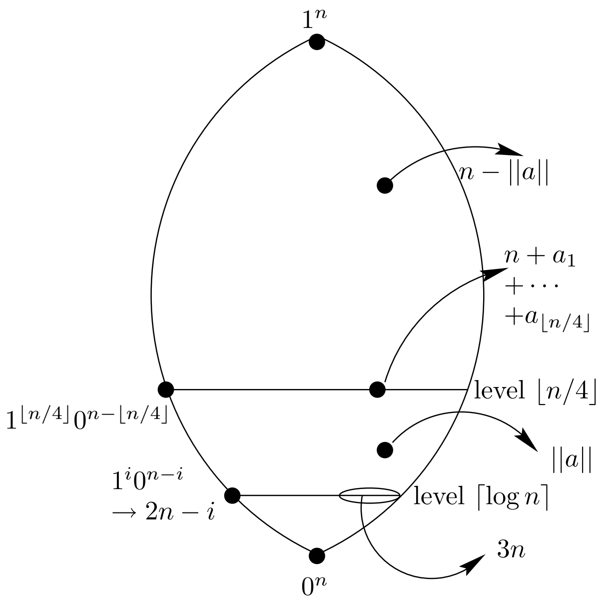


Figure 5: The function PJ_n.

We have to exchange zeros from the first quarter with ones from the rest. Again mutation probabilities of approximately $1/n$ are good for this purpose and the same holds for the short path with increasing fitness values from $1^{\lfloor n/4 \rfloor}0^{n-\lfloor n/4 \rfloor}$ to 0^n. Finally, we have to flip $\lceil \log n \rceil$ zeros (not from the first $\lceil \log n \rceil$ positions) simultaneously. Then a mutation probability of $\Theta((\log n)/n)$ is appropriate. The dynamic $(1+1)$EA has during each phase for each situation one step with a good mutation probability. A good static mutation probability has to be a compromise between a good choice for the early stages and a good choice for the last jump.

However, the dynamic $(1+1)$EA can be a disaster for functions which can be solved efficiently with a static $(1+1)$EA with mutation probability $1/n$. The corresponding function is called PT_n (path with trap). Again, we consider the path from 0^n to 1^n containing all points 1^i0^{n-i}. Except some holes with fitness 0 the fitness of the points equals $n+i$ and 1^n is the global optimum with fitness $3n$. The holes are chosen to guarantee that the algorithm cannot pass the path too quickly. The trap consists of all points which can be reached from some point 1^i0^{n-i}, $n/4 \le i \le 3n/4$, if approximately $n/8$ bits flip simultaneously and which have a Hamming distance of at least $n/16$ to the path. The trap points have a fitness of $2n$. The only points accepted while sitting in a trap are the other trap points and 1^n. Finally, a jump of length at least $n/8$ flipping exactly all zeros is necessary to reach 1^n. All not mentioned points a have a fitness of $n - ||a||$.

It is unlikely that the dynamic $(1+1)$EA reaches the trap before reaching the path. However, while passing the path it is very likely to reach the trap which leads to an exponential run time with very high probability. The $(1+1)$EA with mutation probability $1/n$ also reaches the path with high probability before reaching the trap. Sitting on the path it is very unlikely to reach the trap whose distance is at least $n/16$. Hence, with overwhelming probability the global optimum is reached in polynomial time.

9 Crossover really can help

Jansen and Wegener (1999) were the first to prove that uniform crossover can decrease the run time of evolutionary algorithms without crossover from superpolynomial to polynomial.

Definition 8 *Uniform crossover applied to $x, x' \in \{0,1\}^n$ produces the random point $x'' \in \{0,1\}^n$ where the bits x''_i are chosen independently and $x''_i = x_i$, if $x_i = x'_i$, and $Pr(x''_i = 0) = Pr(x''_i = 1) = 1/2$ otherwise.*

Definition 9 *The function $JUMP_{m,n}$ is defined by*

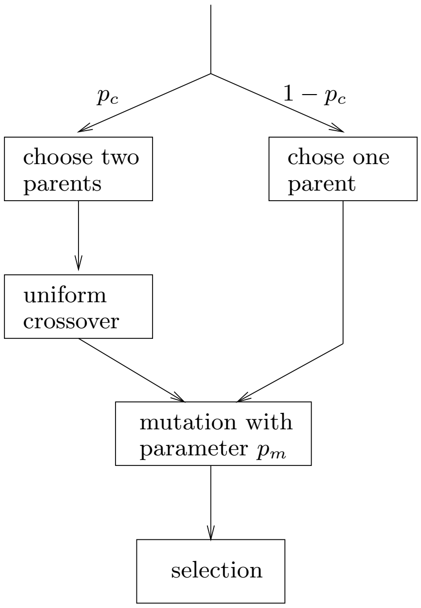
$$JUMP_{m,n}(a) = \begin{cases} ||a|| & \text{if } ||a|| \le n-m \text{ or} \\ & ||a|| = n \\ n - ||a|| & \text{otherwise.} \end{cases}$$

We only consider the case where $m = o(n)$. Then it is very likely that the initial population of polynomial size only contains individuals with at most $n-m$ ones. Individuals with more than $n-m$ ones and less than n ones will not be accepted. Hence, the global optimum has to be produced from individuals with at most $n-m$ ones. Mutations have difficulties with this task. The expected time for the mutation probability $1/n$ is bounded below by n^m and a lower bound of $\Omega(n^{m-\varepsilon})$, $\varepsilon > 0$, for the general case can be obtained easily.

What about uniform crossover? If we have two parents with m zeros and if these parents have no common zero, uniform crossover produces 1^n with probability 2^{-2m} and the probability that a mutation step with mutation probability $1/n$ does not destroy 1^n is approximately e^{-1}. Hence, uniform crossover works well, more precisely leads to expected polynomial time if the individuals are random strings with m zeros and $m = O(\log n)$. The difficulty is the so-called *hitchhiking* effect. Because of selection and crossover the individuals are not independent and have a larger chance of "overlapping zeros".

Therefore, Jansen and Wegener (1999) have investigated a genetic algorithm with the very small crossover probability $p_c = 1/(n \log n)$. The algorithm works as follows.

Algorithm 2 *Steady state genetic algorithm with crossover probability $p_c = 1/(n \log n)$ and mutation probability $p_m = 1/n$. The population size is n and the population is initialized randomly. The main loop of the algorithm looks as follows:*



The choice of the parents or the parent can be done randomly or fitness based. The selection procedure is the following one. If the new individual x equals one of its parents, it is rejected. Otherwise, it replaces one of the worst individuals y if $f(x) \geq f(y)$.

Theorem 10 *The steady state genetic algorithm has for $JUMP_{m,n}$ the following behavior.*

1. *If $m = O(1)$, the expected run time is bounded by $O(n^2 \log n)$.*
2. *If $m = O(\log n)$, the expected run time is bounded by $O(n \log^3 n(n \log^2 n + 2^{2m})) = poly(n)$.*

For the proof of this bound we describe a "typical run" of the genetic algorithm starting with an arbitrary initial population. A typical run should reach the optimum within a given time bound. We describe the events leading to a non-typical run and estimate the corresponding "failure probability". If the sum of failure probabilities is bounded above by $1 - \delta$, $\delta > 0$, the success probability for the considered number of steps is at least δ. If the search was not successful, we start the next phase. Since the initial population was assumed to be arbitrary, we can use the same estimates. Hence, the expected number of phases can be estimated above by δ^{-1}. In our case, $1 - \delta$ is even $o(1)$.

For $\text{Jump}_{m,n}$, a typical run consists of three phases where the length of all phases is bounded above by the stated asymptotic bounds. A typical run has the following properties:

- after phase 1, we either have found the optimum or the population only contains individuals with exactly $n - m$ ones (this implies for all further phases that we only accept individuals with $n - m$ ones or the optimum),

- after phase 2, we either have found the optimum or the population only contains individuals with exactly $n - m$ ones and for each bit position there are at most $n/(4m)$ individuals with a zero at this position (the zeros are sufficiently well spread),

- during phase 3, we either find the optimum or the population only contains individuals with exactly $n - m$ ones and for each bit position there are at most $n/(2m)$ individuals with a zero at this position and at the end of phase 3 the optimum is found.

The analysis of phase 1 is easy using the coupon collector's theorem. If the condition of phase 3 is fulfilled, the probability that two randomly chosen parents (or fitness based chosen parents which is the same, since the fitness of all individuals is the same) have no common zero is at least

1/2 and uniform crossover can work. The analysis of phase 2 is the difficult task (the difficult part of the analysis of phase 3 can be performed similarly). We consider only bit position 1, all other positions lead to the same failure probabilities. Hence, the total failure probability can be bounded by n times the failure probability for bit position 1.

We denote by $p^-(z)$ and $p^+(z)$ the probability that the number of individuals with a zero at bit position 1 decreases by 1 resp. increases by 1 during one step starting with z individuals with a zero at bit position 1. In order to estimate these probabilities we have to perform quite exact calculations. Since p_c is small enough, we can consider a crossover step as a "bad event" increasing the number of zeros at position 1. The result of the calculations is that

$$z \geq n/(8m) \Longrightarrow \quad \begin{array}{l} p^-(z) = \Theta\left(\frac{1}{n}\right) \text{ and} \\ p^-(z) - p^+(z) = \Theta\left(\frac{1}{n}\right). \end{array}$$

Hence, either $z < n/(8m)$ and we are far away from a failure or $z \geq n/(8m)$ and we have a large enough tendency to decrease the number of zeros. The precise calculations lead to the desired result.

Conclusions

We have argued why the analysis of evolutionary algorithms should be performed just as the analysis of other randomized algorithms and search heuristics. Several examples of new results, some of them proving or disproving well-known conjectures, others leading to results of a new type, prove that this approach is worth to be followed further.

References

[1] Bäck, T. (1993). Optimal mutation rates in genetic search. Proc. of 5th ICGA (Int. Conf. on Genetic Algorithms), 2–8.

[2] Bäck, T. (1998). An overview of parameter control methods by self-adaptation in evolutionary algorithms. Fundamenta Informaticae 35, 51–66.

[3] Bäck, T., Fogel, D. B., and Michalewicz, Z. (Eds). (1997). *Handbook of Evolutionary Computation.* Oxford Univ. Press.

[4] Beyer, H.-G. (1996). Toward a theory of evolution strategies: Self-adaptation. Evolutionary Computation 3, 311–347.

[5] Droste, S., Jansen, T., and Wegener, I. (1998a). A rigorous complexity analysis of the (1+1) evolutionary algorithm for separable functions with Boolean inputs. Evolutionary Computation 6, 185–196.

[6] Droste, S., Jansen, T., and Wegener, I. (1998b). On the optimization of unimodal functions with the (1+1) evolutionary algorithm. Proc. of PPSN V (Parallel Problem Solving from Nature), LNCS 1498, 13–22.

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

[8] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley.

[9] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* The University of Michigan Press.

[10] Horn, J., Goldberg, D. E., and Deb, K. (1994). Long path problems. Proc. of PPSN III (Parallel Problem Solving from Nature), LNCS 866, 149–158.

[11] 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.

[12] 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.

[13] 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. Submitted to IEEE Trans. on Evolutionary Computation.

[14] 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.

[15] Mitchell, M., Holland, J. H., and Forrest, S. (1994). When will a genetic algorithm outperform hill climbing. In: Cowan, J., Tesauro, G., and Alspector, J. (Eds): Advances in Neural Information Processing Systems. Morgan Kaufmann.

[16] Motwani, R., and Raghavan, P. (1995). *Randomized Algorithms.* Cambridge Univ. Press.

[17] Mühlenbein, H. (1992). How genetic algorithms really work. I. Mutation and hillclimbing. Proc. of PPSN II (Parallel Problem Solving from Nature), 15–25.

[18] Rabani, Y., Rabinovich, Y., and Sinclair, A. (1995). A computational view of population genetics. Random Structures and Algorithms 12, 314–334.

[19] Rosenberg, I. G. (1975). Reduction of bivalent maximization to the quadratic case. Cahiers du Centre d'Etudes de Recherche Operationelle 17, 71–74.

[20] Rudolph, G. (1997). How mutations and selection solve long path problems in polynomial expected time. Evolutionary Computation 4, 195–205.

[21] Schwefel, H.-P. (1995). *Evolution and Optimum Seeking.* Wiley.

[22] Wegener, I., and Witt, C. (2000). On the behavior of the (1+1) evolutionary algorithm on quadratic pseudo-boolean functions. Submitted to Evolutionary Computation.

[23] Wolpert, D. H., and Macready, W. G. (1997). No free lunch theorems for optimization. IEEE Trans. on Evolutionary 1, 67–82.