Analysis of Evolutionary Algorithms for the Longest Common Subsequence Problem

Thomas Jansen
Universität Dortmund
Fachbereich Informatik
LS Effiziente Alg. u. Komplexitätstheorie
44221 Dortmund, Germany
Thomas.Jansen@udo.edu

Dennis Weyland
Fernuniversität in Hagen
Fakultät für Mathematik und Informatik
Lehrgebiet Komplexe Analysis
58084 Hagen, Germany
Dennis.Weyland@FernUni-Hagen.de

ABSTRACT

In the longest common subsequence problem the task is to find the longest sequence of letters that can be found as subsequence in all members of a given finite set of sequences. The problem is one of the fundamental problems in computer science with the task of finding a given pattern in a text as an important special case. It has applications in bioinformatics, problem-specific algorithms and facts about its complexity are known. Motivated by reports about good performance of evolutionary algorithms for some instances of this problem a theoretical analysis of a generic evolutionary algorithm is performed. The general algorithmic framework encompasses EAs as different as steady state GAs with uniform crossover and randomized hill-climbers. For all these algorithms it is proved that even rather simple special cases of the longest common subsequence problem can neither be solved to optimality nor approximately solved up to an approximation factor arbitrarily close to 2.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Pattern Matching*; G.3 [**Probability and Statistics**]: Probabilistic Algorithms

General Terms

Algorithms, Performance, Theory

Keywords

longest common subsequence problem, run time analysis, crossover

1. INTRODUCTION

Evolutionary algorithms are robust general search heuristics that are applied in many different contexts. One particularly important area of application is optimization, typically for problems that are poorly understood and where no problem-specific algorithms are known. There is a wide variety of different kinds of evolutionary algorithms and it is common practice to enhance and tune the algorithms in order to improve their empirical performance leading to very sophisticated and sometimes rather complicated heuristics. In comparison, theoretical analysis of evolutionary algorithms is way behind. One branch of evolutionary algorithm theory is concerned with the analysis of the expected time needed to solve some problem either to optimality or approximately up to some specified approximation factor. This corresponds to the classical field of design and analysis of algorithms where one is concerned with the (expected) run time of algorithms. Such analyses for evolutionary algorithms are often concerned with very simple evolutionary algorithms dealing with very simple fitness functions artificially designed to serve as example. This field started off 15 years ago with the analysis of a simple randomized hill-climber, called the (1+1) evolutionary algorithm, on a toy problem called OneMax [11]. Since then an arsenal of powerful tools and methods for the analysis of evolutionary algorithms has been developed [1]. Today, analyses of evolutionary algorithms' run time are no longer restricted to artificial example functions. They are also concerned with "real" problems, often classical combinatorial optimization problems. Concrete examples include the maximum matching problem [5], computation of an Eulerian cycle [12], computation of minimum spanning trees [13], the maximum clique problem [15], the single source shortest paths problem, and sorting [14]. In addition to these analytical studies empirical investigations provide helpful information [2]. One combinatorial optimization problem where results of empirical investigations are known [7, 9] but theoretical analysis has not yet been performed is the longest common subsequence problem (LCS). There the task is to find the longest sequence of letters from some finite alphabet Σ that can be found as subsequence in all members of a given finite set of sequences over the same alphabet Σ. The problem is one of the fundamental problems in computer science and has applications in bioinformatics. Note that the task of finding a given pattern in a text is included as an important special case. There is a problem-specific algorithm (implementing a dynamic programming approach) solving the problem for a fixed number of n sequences of lengths l_1, l_2, $\ldots$, l_n in time $O\left(\prod_{i=1}^{n} l_i\right)$ [3]. Julstrom and Hinkemeyer

GECCO'07 July 7–11, 2007, London, England, United Kingdom.

[9] present a coding of this problem suitable for evolutionary algorithms. Using genetic algorithms they report good results, even in comparison with the problem-specific dynamic programming approach. However, these empirically convincing results are not accompanied by an analysis actually proving that some evolutionary algorithm is to some degree efficient on some kinds of instances of LCS in some coding. Here, such an analysis is performed. We investigate a general framework for evolutionary algorithms using an elitist selection (known as plus-selection from evolution strategies), uniform crossover, and standard bit mutation. Setting the parent and offspring population size as well as the crossover probability a wide range of different evolutionary algorithms can be obtained as instances of this EA framework. This includes a steady state genetic algorithm as well as the so-called (1+1) EA. We use the same coding for LCS as described by Julstrom and Hinkemeyer [9] and consider the fitness function proposed there together with two different fitness functions. In this general setting, we prove a couple of lower bounds on the performance of these evolutionary algorithms by considering specific LCS instances and analyzing the EAs' performance on these instances. This is in accordance with the usual worst case perspective used when analyzing the (expected) run time of (randomized) algorithms [3].

We begin with precise formal definitions of the problem, the specific coding we consider and the fitness functions we use in Section 2. After that we give a precise definition of the kind of evolutionary algorithms investigated (Section 3). Using this framework we prove several lower bounds on the performance showing that these evolutionary algorithms are not at all efficient for this LCS coding in the worst case (Section 4). Reconsidering the LCS instances used in that section, we discuss restricted classes of LCS instances and show that these restrictions do not imply much improved worst case performance (Section 5). We discuss our findings and point out open problems in Section 6.

2. DEFINITION OF THE PROBLEM

The longest common subsequence problem (LCS) is defined over some finite alphabet Σ. In computer science, the binary alphabet $\Sigma = \{0,1\}$ is commonly used. In bioinformatics, the alphabet $\Sigma = \{\mathrm{A},\mathrm{C},\mathrm{G},\mathrm{T}\}$ for DNA coding is of particular interest. We consider sequences of letters from Σ of finite lengths. For such a sequence s we write $|s|$ for the length of s, i. e., the number of letters from Σ in s. Moreover, we write $|s|_l$ for a sequence s and a letter $l \in \Sigma$ to refer to the number of occurrences of l in s. For example, for the sequence 0001010 we have $|0001010| = 7$, $|0001010|_0 = 5$, $|0001010|_1 = 2$, and $|0001010|_2 = 0$. We write $s_{(i)}$ with $i \in \mathbb{N}_0$ for the prefix of length i of s, e. g., $0001010_{(3)} = 000$. We use ε as symbol for the empty sequence of length 0, i. e., $|\varepsilon| = 0$. For convenience, we use the notion s^i for a sequence s and $i \in \mathbb{N}_0$ for a repetition of s for i times. Thus, $0^3 = 000$ and $(01)^2 = 0101$. Using this notation and writing concatenations of sequences without any special symbol, we have $0001010 = 0^3(10)^2$. For an alphabet Σ, we refer to the set of all sequences of lengths exactly i as Σ^i. The set of all sequences of finite length over Σ is called Σ^*, i. e., $\Sigma^* = \bigcup_{i \geq 0} \Sigma^i$.

For two sequence $Y = y_1 \cdots y_m \in \Sigma^m$ and $X = x_1 \cdots x_n \in \Sigma^n$ (with $m \leq n$) we call Y a subsequence of X if there is a sequence of indices $0 < i_1 < i_2 < \cdots < i_m \leq n$ such that $y_j = x_{i_j}$ holds for all $j \in \{1, 2, \ldots, m\}$. For example, 00100 is a subsequence of 0001010 while 001100 is no subsequence of 0001010. Note that the sequence of indices need not be unique, for our first example the sequences 1, 2, 4, 5, 7 and 1, 3, 4, 5, 7 are both valid sequences that prove that 00100 is a subsequence of 0001010.

For m sequences $X_1, X_2, \ldots, X_m$ over the same alphabet Σ, a sequence Y over the same alphabet Σ is called a common subsequence, if Y is a subsequence of X_i for all $i \in \{1, 2, \ldots, m\}$. It is called a longest common subsequence, if all other $Y' \in \Sigma^*$ that are common subsequences do not have greater length, i. e., $|Y| \geq |Y'|$ for all common subsequences Y'.

In the longest common subsequence problem, one is given m sequences $X_1, X_2, \ldots, X_m \in \Sigma^*$. The objective is to find a sequence $Y \in \Sigma^*$ that is a longest common subsequence. Note that there need not be a uniquely determined solution. For example, considering $X_1 = 100011$, $X_2 = 01110000$, and $X_3 = 001100$, we see that 011 and 100 are both longest common subsequences.

When using evolutionary algorithms for this problem, a coding is needed. Clearly, there are countless possible ways of defining a search space and mappings from points in the search space to candidate solutions. We consider a simple binary encoding here that is used by Julstrom and Hinkemeyer, too [9]. For m sequences $X_1, X_2, \ldots, X_m \in \Sigma^*$, let X_1 be a shortest sequence, i. e., we assume $|X_1| \leq |X_i|$ for all $i \in \{1, 2, \ldots, n\}$ without loss of generality. Let $|X_1| = n$ be this length. Clearly, the length of a longest common subsequence is bounded above by the length of the shortest sequence n. Therefore, we can represent a candidate solution $Y \in \Sigma^*$ by some $s \in \{0,1\}^n$, i. e., a bit string of fixed length n where the bits set to 1 indicate letters in X_1 in the candidate solution while bits set to 0 indicate letters left out. For example, for $X_1 = \mathrm{ACGTA}$, 00010 represents T, 11101 represents ACGA. The all 1-string 1^n represents X_1, the all 0-string 0^n represents ε. Note that this way $s \in \{0,1\}^n$ may represent sequences that are not common subsequences. We will have to take care of such infeasible solutions when we define a fitness function. Note that with the all 0-string 0^n a trivial feasible candidate solution is known. We use a function $c\colon \{0,1\}^n \to \Sigma^*$ that maps bit strings to the candidate solutions they represent.

Julstrom and Hinkemeyer [9] define a fitness function $f_{\mathrm{JH}}\colon \{0,1\}^n \to \mathbb{Z}$ that maps feasible points to positive fitness values and infeasible points to negative values.

Definition 1. For sequences $X_1, X_2, \ldots, X_m \in \Sigma^*$ with $|X_1| \leq |X_i|$ for all $i \in \{2, \ldots, m\}$, let $n = |X_1|$. For $s \in \{0,1\}^n$ let $c(s)$ be the candidate solution represented by s and let $k(s)$ be the number of sequences $X_1, X_2, \ldots, X_m$ such that $c(s)$ is a subsequence. The function $f_{\mathrm{JH}}\colon \{0,1\}^n \to \mathbb{Z}$ is defined in the following way:

$$f_{\mathrm{JH}}(s) = \begin{cases} 3000(|c(s)| + 30k(s) + 50) \\ \quad \text{if } |c(s)| = n \text{ and } k(s) = m \\ 3000(|c(s)| + 30k(s)) \\ \quad \text{if } |c(s)| < n \text{ and } k(s) = m \\ -1000(|c(s)| + 30k(s) + 50)(m - k(s)) \\ \quad \text{if } |c(s)| = n \text{ and } k(s) < m \\ -1000(|c(s)| + 30k(s))(m - k(s)) \\ \quad \text{if } |c(s)| < n \text{ and } k(s) < m \end{cases}$$

While it is reported that this fitness function leads to good results we consider it to be somewhat contrived. In particular, it is unclear why the special case $|c(s)| = n$ gets extra reward and extra attention: in general, no feasible solution with $|c(s)| = n$ need to exist. We propose a simpler fitness function based on the following idea. We map the letters in $c(s)$ to letters in the sequences $X_1, X_2, \ldots, X_m$ from left to right. Each letter that can be mapped leads to a reward of $+1$, letters that cannot be mapped lead to a decrease by -1. The formal definition follows.

Definition 2. For sequences $X_1, X_2, \ldots, X_m \in \Sigma^*$ with $|X_1| \leq |X_i|$ for all $i \in \{2, \ldots, m\}$, let $n = |X_1|$. For $s \in \{0,1\}^n$ let $c(s)$ be the candidate solution represented by s and let a function MAX be defined by MAX($c(s)$, X_1, X_2, …, X_m) = min{max{k | $c(s)_{(k)}$ is subsequence of X_i} | $i \in \{1, \ldots, m\}$}.

The function $f_{\max}\colon \{0,1\}^n \to \mathbb{Z}$ is defined in the following way: $f_{\max}(s) = \text{MAX}(c(s), X_1, X_2, \ldots, X_m) - (|c(s)| - \text{MAX}(c(s), X_1, X_2, \ldots, X_m))$

It is easy to see that $f_{\max}(c) = 2\text{MAX}(c(s), X_1, X_2, \ldots, X_m\}) - |c(s)|$ holds. While $f_{\max}$ also has the property that feasible solutions are mapped to non-negative values (with $f_{\max}(\varepsilon) = 0$), it is not the case that infeasible solutions necessarily have negative fitness values. It is, however, the case that for infeasible solutions removing a letter that cannot be mapped, i. e., setting the corresponding bit to 0, increases the function value.

Since the mapping done in $f_{\max}$ is a very simple left-to-right mapping, one may speculate that a more clever mapping could yield more valuable information for an evolutionary algorithm and lead to a better performance. Clearly, an optimal mapping would compute the longest common subsequence of the sequences $X_1, X_2, \ldots, X_n$ and the candidate solution $c(s)$. Since for such a mapping LCS needs to be solved, such a fitness function makes no sense at all with respect to practical applications. Since, on the other hand, it is the best possible mapping and allows to investigate of how much use this information is for an evolutionary algorithm, it is an interesting fitness function from a theoretical point of view. Therefore, we give a formal definition here and include it in our considerations.

Definition 3. For sequences $X_1, X_2, \ldots, X_m \in \Sigma^*$ with $|X_1| \leq |X_i|$ for all $i \in \{2, \ldots, m\}$, let $n = |X_1|$. For $s \in \{0,1\}^n$ let $c(s)$ be the candidate solution represented by s and let a function LCS be defined that yields as LCS($c(s)$, X_1, X_2, …, X_m) the length of a longest common subsequence of $c(s)$, X_1, X_2, …, X_n.

The function $f_{\text{LCS}}\colon \{0,1\}^n \to \mathbb{Z}$ is defined in the following way: $f_{\text{LCS}}(s) = \text{LCS}(c(s), X_1, X_2, \ldots, X_m) - (|c(s)| - \text{LCS}(c(s), X_1, X_2, \ldots, X_m))$

Again, it is easy to see that $f_{\text{LCS}}(c) = 2\text{LCS}(c(s), X_1, X_2, \ldots, X_m\}) - |c(s)|$ holds. Since f_{LCS} is similar in spirit to $f_{\max}$, it comes as no surprise that it has similar properties.

3. DEFINITION OF THE EVOLUTIONARY ALGORITHMS

There is a large variety of evolutionary algorithms that can have very different properties. For theoretical analyses, it is of course necessary to specify exactly what evolutionary algorithm is analyzed. While in practical applications it makes sense to improve an evolutionary algorithm by adding more heuristic ideas such advanced evolutionary algorithms are typically almost impossible to analyze theoretically, at least for non-trivial fitness functions. For theoretical analyses it makes therefore sense to concentrate on rather simple evolutionary algorithms that are close to the more basic variants [4]. For the development of analytical tools, it makes even sense to restrict the analysis to extremely simple search heuristics like randomized mutation hillclimbers (also known as (1+1) EA). While such analyses provide valuable insights, they have the drawback to tell the practitioner not so much about the variants she usually applies. Here, we try to find some middle ground by considering a class of simple evolutionary algorithms that is rather general and encompasses typical steady state genetic algorithms.

The evolutionary algorithms we consider make use of two variation operators, namely standard bit mutation and uniform crossover. Standard bit mutation creates an offspring y as a copy of its one parent x and, independently for each bit in y, changes the value of this bit with probability $1/n$. While it is known that mutation probabilities different from $1/n$ can be useful, $1/n$ is the most common and most recommended choice. Uniform crossover creates an offspring y from two parents x_1 and x_2, by copying, independently for each bit in y, the value of the bit with equal probability either from x_1 or x_2. These two variation operators are embedded in an evolutionary algorithm following the $(\mu+\lambda)$-selection paradigm from evolution strategies: The parent population size is μ, the offspring population size is λ. In each generation, λ offspring are created independently and identically distributed by selecting parents uniformly at random and applying variation operators to those parents. Then, the next population is selected from both, the parents and the offspring, selecting the μ best. If there are ties, offspring are preferred over parents. If there are still ties, those are broken randomly. We give a precise definition in pseudo-code.

Definition 4. **Generic Evolutionary Algorithm**

1. For $i \in \{1, 2, \ldots, \mu\}$
   Select $x_i \in \{0,1\}^n$ uniformly at random.
2. For $i \in \{1, 2, \ldots, \lambda\}$
   Select $z_1 \in \{x_1, \ldots, x_\mu\}$ uniformly at random.
   With probability p_c
   Select $z_2 \in \{x_1, \ldots, x_\mu\}$ uniformly at random.
   $y :=$ uniform crossover(z_1, z_2)
   $y_i :=$ mutation(y)
   Else (with the remaining prob. of $1 - p_c$)
   $y_i :=$ mutation(z_1)
3. Sort $x_1, \ldots, x_\mu, y_1, \ldots, y_\lambda$ descending according to fitness breaking ties by sorting offspring in front of parents breaking remaining ties randomly.
4. Replace $x_1, \ldots, x_\mu$ by the first μ elements from this sorted sequence.
5. Continue at line 2.

The setting $\mu = \lambda = 1$ and $p_c = 0$ yields the well-known (1+1) EA as already analyzed by Mühlenbein [11] and many others. With $\mu > 1$, $\lambda = 1$ and $p_c > 0$ we get some typical steady state genetic algorithm with uniform crossover.

When concerned with exact optimization, we investigate the time an evolutionary algorithm as defined in Definition 4 needs to find a longest common subsequence. As is common

practice in theoretical analyses of evolutionary algorithms we assume that time can be measured by counting the number of function evaluations. This can easily be connected to the number of iterations of the main loop (lines 2–5), also called generations. Line 1 is executed only once and infers μ function evaluations. In each generation it suffices to evaluate the offspring, so that we get a total of $\mu + g \cdot \lambda$ function evaluations in g generations. We call the random number of function evaluations until an optimal solution is found the optimization time T and are mostly interested in its expectation $\mathrm{E}(T)$. For lower bounds, we can use $\mathrm{E}(T) \geq t \cdot \mathrm{Prob}(T \geq t)$ and concentrate on $\mathrm{Prob}(T \geq t)$. When concerned with approximation, we investigate the time until some common subsequence Y is found such that for all common subsequences Y' we have $|Y'|/|Y| \leq r$ for some pre-specified approximation ratio r. Again we use the number of function evaluations as measure for time. As in complexity theory, we consider only polynomial optimization times to be efficient. Therefore, we restrict our attention to population sizes that are bounded above by some polynomial p where p is a polynomial in n, the length of the shortest sequence in the input. We consider the time needed as a function of this parameter n. This corresponds to usual analyses of run times of algorithms where the run time is described as a function of the input size. Adopting the usual worst case perspective we are looking for a function $t\colon \mathbb{N} \to \mathbb{N}$ such that for any LCS instance where the shortest sequence has length n the expected time $\mathrm{E}(T)$ is bounded by $O(t(n))$.

4. LOWER BOUNDS ON THE EVOLUTIONARY ALGORITHM PERFORMANCE

In this section we prove a couple of lower bounds on the performance of an evolutionary algorithm as defined in Definition 4 for LCS. We consider both, exact optimization and approximation. For a lower bound, it suffices to define a specific instance of the longest common subsequence problem and analyze the EA for this specific instance.

Before we begin with a concrete lower bound, we bring structure to LCS by characterizing instances according to some of their properties. One of the obvious properties of an LCS instance is the number of sequences in the input. Clearly, the problem becomes trivial if this number equals 1 since each sequence is the longest subsequence of itself. For larger numbers of sequences, the problem difficulty can only increase with an increasing number of sequences. In particular, by copying a sequence we can obviously increase the number of sequences without increasing the difficulty of the instance. We see that instances with exactly two sequences are the easiest non-trivial LCS instances. Note that the problem-specific LCS algorithm using a dynamic programming approach has polynomial worst case run time for such instances [3].

Another obvious property is the number of letters in the alphabet Σ. We may assume w. l. o. g. that Σ does not contain any letters that appear in none of the input sequences, otherwise Σ can easily be made smaller. Clearly, the problem becomes trivial if this number $|\Sigma|$ equals 1 since then the longest common subsequence is given by the shortest sequence of the instance. For larger numbers of letters, the problem difficulty can only increase with an increasing $|\Sigma|$. Thus the instances with $|\Sigma| = 2$ are the easiest non-trivial LCS instances.

We restrict ourselves in the following to problem instances with $\Sigma = \{0, 1\}$ and exactly two sequences in each instance. Thus, we are dealing with a simple sub-class of all LCS instances. In particular, optimal solutions to all our instances can be found deterministically in polynomial time. Large lower bounds for these simple instances are therefore a strong indication that the kind of evolutionary algorithms we consider is not an efficient heuristic for LCS.

Our worst case instances and our proofs will all follow the same pattern. The two sequences define a fitness landscape with a local optimum with a huge basin of attraction and a global optimum that is due to this structure of the local optimum difficult to find. In particular, we partition the search space into three disjoint sets S_0, S_1, S_2, such that with probability very close to 1 the initial population is completely within S_1. The optimal solution is in S_2 and all points in S_2 have large Hamming distance from all points in S_1. All other points belong to S_0 and have worse fitness in comparison to all points in S_1. Due to the strict plus-selection employed in the evolutionary algorithms we consider, the population cannot enter S_0 and thus has to find a way from S_1 to S_2 directly. Due to the large Hamming distance this is unlikely to happen due to mutation. And since the strings in S_1 are either sufficiently similar or S_2 is sufficiently small, this is unlikely to happen due to uniform crossover, either.

Detailed proofs for the results we present a quite lengthy. Since all our proofs follow basically the same structure, we will present a proof for the first statement in full detail. For the other statements, the proof ideas and differences to this first proof are outlined. We invite the reader to fill in the missing details.

THEOREM 1. *The probability that an evolutionary algorithm as defined in Definition 4 finds an optimal solution to a worst case instance of LCS $X, Y \in \{0,1\}^*$ with $X \in \{0,1\}^n$ using the fitness function f_{JH}, $f_{\max}$, or f_{LCS} within t function evaluations is bounded above by $t \cdot e^{-\Omega(n)}$ for all settings of p_c, $\mu = n^{O(1)}$, and $\lambda = n^{O(1)}$.*

PROOF. We begin with the proof for the fitness function f_{JH}. For each value of n we present an LCS instance with two sequences over the alphabet $\{0, 1\}$ where the shorter sequence has length n. We assume that n is a multiple of 32. If this is not the case, we replace in the following n by $n' := 32 \cdot \lfloor n/32 \rfloor$ and append a sequence of $n - n'$ letters to the two sequences we define. This does not change our results if n is sufficiently large. Since we are proving an asymptotic result (using O- and Ω-notation) we can assume that n is sufficiently large.

Consider the two sequences

$$X = 0^{(1/4)n}1^{(3/4)n}$$

and

$$Y = 1^{(3/4)n}0^{(5/32)n}1^{(13/32)n}$$

where the shorter sequence, X, obviously has length n. We partition the evolutionary algorithm's search space $S = \{0,1\}^n$ in three disjoint sets S_0, S_1, S_2 in the following way (with $\gamma := 180,000 + 3,000 \cdot (14/32)n$):

$$\begin{aligned} S_0 &:= \{s \mid (f_{\mathrm{JH}}(s) < \gamma)\} \\ S_1 &:= \left\{s \mid (f_{\mathrm{JH}}(s) \geq \gamma) \wedge \left(|c(s)|_0 > 0\right)\right\} \\ S_2 &:= \left\{s \mid (f_{\mathrm{JH}}(s) \geq \gamma) \wedge \left(|c(s)|_0 = 0\right)\right\} \end{aligned}$$

Note that we have $f_{\mathrm{JH}}(s) = \gamma$ for $s \in \{0,1\}$ that represents a common subsequence of X and Y of length $(14/32)n$. Thus, S_0 contains all s representing infeasible solutions and feasible solutions of length less than $(14/32)n$. The other feasible solutions are divided into S_1 and S_2. The set S_1 contains all s representing solutions with at least one letter $0 \in \Sigma$, the set S_2 contains all other solutions. Observe that all common subsequences are of the form 0^i1^j (with $i, j \in \mathbb{N}_0$) as enforced by X. Thus, a common subsequence containing the letter 0 can have length at most $(5/32)n + (13/32)n = (9/16)n$: this is the longest such sequence in Y. We conclude that the length of all solutions represented by bit strings in S_1 have length at least $(14/32)n$ and at most $(9/16)n$. Now consider sequences 1^j with $(9/16)n < j \leq (3/4)n$. Clearly, these sequences are all common subsequences of X and Y. Due to the definition of the partition, bit strings representing these sequences are all in S_2. Since the fitness increases with length, the global optimum of f_{JH} is a string representing $1^{(3/4)n}$, the longest common subsequence.

Now we prove that the initial population is completely within S_1 with probability exponentially close to 1. Our main tool in the proof are Chernoff bounds [10]. Chernoff bounds are so-called tail estimations, they give upper bounds for the probability that the sum of a number of independent 0-1-valued random variables deviates from its expected value. We make use of the following result. For m independent 0-1-valued random variables that all take value 1 with probability p $(0 < p < 1)$ and, thus, the value 0 with probability $1 - p$, the probability that the sum of these m random variables deviates from its expected value pm by a constant factor is at least $e^{-\Omega(pm)}$. Chernoff bounds are in fact slightly more general than this. Moreover, for the special case we consider slightly stronger bounds are known [6]. For our purpose, however, these results are sufficiently strong.

The initial population consists of μ independently and identically distributed bit strings from $\{0,1\}^n$, where each bit string is independently selected according to the uniform distribution. We identify a bit string $s \in \{0,1\}^n$ with the candidate solution it represents, $c(s)$. We see that the number of occurrences of the letter 0 is binomially distributed with parameters $(1/4)n$ and $1/2$, since we have a letter 0 for each of the left-most $(1/4)n$ bits that are set to 1. In the same way we see that the number of occurrences of the letter 1 is binomially distributed with parameters $(3/4)n$ and $1/2$. We can apply Chernoff bounds to bound the probability for deviations from the expected number of 0s and 1s in the following way. For the i-th bit, we define a 0-1-valued random variable B_i. This way, the number of occurrences of the letter 0 is the sum of $(1/4)n$ independent 0-1-valued random variables that all take value 0 with probability $1/2$. We consider one initial individual and see that it has the form 0^i1^j with $(3/32)n < i < (5/32)n$ and $(11/32)n < j < (13/32)n$ with probability $1 - e^{-\Omega(n)}$. Thus, this initial individual has length greater than $(14/32)n$ and therefore cannot belong to S_0. Since it contains more than $(3/32)n$ occurrences of the letter 0, it cannot belong to S_2. Thus, an initial individual belongs to S_1 with probability $1 - e^{-\Omega(n)}$. We are dealing with a population of size μ where μ is polynomially bounded. Thus, using the simple union bound, we see that the probability that there is an individual not belonging to S_1 in the initial population is bounded above by $\mu \cdot e^{-\Omega(n)} = e^{\ln \mu - \Omega(n)} = e^{-\Omega(n)}$.

We see that the initial population is completely contained in S_1 with probability $1 - e^{-\Omega(n)}$. Due to the strict plus-selection employed in lines 3–4 of Definition 4, the population can never contain an element of S_0. Thus, the time needed to reach the longest common subsequence is bounded below by the time needed to reach a solution in S_2 based on a population consisting of solutions from S_1, only.

In each generation, λ offspring are created independently and identically distributed. We assume that the current population is completely contained in S_1. Let A denote the event to produce an offspring that belongs to S_2 given that mutation is used to produce the offspring. Let B denote the event to produce an offspring that belongs to S_2 given that uniform crossover (with subsequent mutation) is used to produce the offspring. Then the probability to produce an offspring belonging to S_2 is given by $(1-p_c) \cdot \mathrm{Prob}\,(A) + p_c \cdot \mathrm{Prob}\,(B) \leq \mathrm{Prob}\,(A) + \mathrm{Prob}\,(B)$. We give bounds on $\mathrm{Prob}\,(A)$ and $\mathrm{Prob}\,(B)$.

First we deal with the case that the offspring is produced by a mutation. We claim that for each $s \in S_1$ we have $|c(s)|_0 \geq (1/32)n$. By definition of S_1, $s \in S_1$ implies $f_{\mathrm{JH}}(s) \geq \gamma$ (with $\gamma = 180,000 + 3,000 \cdot (14/32)n$) and $|c(s)|_0 \geq 1$. As we have seen above, $f_{\mathrm{JH}}(s) \geq \gamma$ implies $|c(s)| \geq (14/32)$. On the other hand, $|c(s)|_0 \geq 1$ implies $|c(s)|_1 \leq (13/32)n$, otherwise $c(s)$ is not a subsequence of Y. Thus, $|c(s)|_0 \geq (1/32)n$ follows as claimed. These observations imply the following for mutations leading from S_1 to S_2. In such a mutation, all $|c(s)|_0$ bits set to 1 in s and leading to the letter 0 in $c(s)$ have to be flipped to 0 making the letter 0 in $c(s)$ vanishing. The probability to mutate b specific bits in a single mutation is bounded above by $(1/n)^b$. Since we have $|c(s)|_0 \geq (1/32)n$, such a mutation has probability $\leq n^{-(1/32)n} = e^{-\Omega(n \log n)}$. Since we create λ offspring in each generation (and λ is polynomially bounded), the probability to create such an offspring in one generation is bounded above by $\lambda \cdot e^{-\Omega(n \log n)} = e^{\ln(\lambda) - \Omega(n \log n)} = e^{-\Omega(n \log n)}$ (again simply making use of the union bound). Clearly, this is sufficiently small to have the bound $t \cdot e^{-\Omega(n)}$ for t generations claimed in the theorem.

We are left with the case where the offspring is produced by uniform crossover. As we know from our considerations for offspring created by mutation, it is the 1-bits in $s \in S_1$ leading to the letter 0 in $c(s)$ that all need to be changed to 0 in the offspring $y \in \{0,1\}^n$ that make the step from S_1 to S_2 difficult. Consider the two parents z_1 and z_2 selected from the current population for crossover (see Definition 4). Since the complete population is contained in S_1, we have $|c(z_1)|_0 \geq n/32$ and $|c(z_2)|_0 \geq n/32$. Consider the i-th position in both, z_1 and z_2. If both parents have the same value b at this position, then the offspring (after uniform crossover prior to mutation) has value b at this i-th position, too. If the parents have different values at this position, then the offspring (after uniform crossover and prior to mutation) has value 0 or value 1 with equal probability $1/2$. This is due to the definition of uniform crossover and holds independently for all positions. Now we consider the at least $n/32$ positions in z_1 and the at least $n/32$ positions in z_2 where critical 1-bits are. We call a position a common position if both parents, z_1 and z_2, have a critical 1-bit there. Let C be the number of such common positions. We distinguish two cases with respect to C. First, assume $C \geq n/64$. In this case the offspring y (after uniform crossover and prior to mutation) is guaranteed to have at least $n/64$ positions set to 1, thus

$|c(y)|_0 \geq n/64$ holds. The final offspring after mutation may still belong to S_2, but to this end a mutation of these $|c(y)|_0$ bits is necessary. We know that such a mutation has probability at most $n^{-|c(y)|_0} \leq n^{-n/64} = n^{-\Omega(n \log n)}$. Thus, as in the case where the offspring is created by mutation alone, the probability to create an offspring in S_2 is sufficiently small. Now we are only left with the case that $C < n/64$ holds. Then there are $|c(z_1)|_0 - C + |c(z_2)|_0 - C$ positions in y where the bit value is 0 or 1 with equal probability 1/2. Since we have $C < n/64$ and $|c(z_i)|_0 \geq n/32$ for $i \in \{1, 2\}$, we have that this number of positions is bounded below by $n/32 + n/32 - 2 \cdot n/64 = n/32$. Clearly, the number of bits with value 1 is bounded below by a random variable that is binomially distributed with parameters $n/32$ and 1/2. We can apply Chernoff bounds as above and see that with probability $1 - e^{-\Omega(n)}$ we have $|c(y)|_0 \geq n/128$. Clearly, this offspring y is subject to mutation and may be mutated to some point in S_2. However, such a mutation has probability at most $n^{-n/128} = e^{-\Omega(n \log n)}$. Thus, together we have $\text{Prob}(B) = e^{-\Omega(n)}$.

Since we have $\text{Prob}(A) = e^{-\Omega(n \log n)}$ and $\text{Prob}(B) = e^{-\Omega(n)}$, we see that in each generation an optimal solution is found with probability $e^{-\Omega(n)}$. This completes the proof for f_{JH}.

For $f_{\max}$, we can use the same LCS instance given by

$$X = 0^{n/4}1^{(3/4)n}$$

and

$$Y = 1^{(3/4)n}0^{(5/32)n}1^{(13/32)n}$$

as for f_{JH}. We define a partition S_0, S_1, S_2 of the search space $S = \{0,1\}^n$ by

$$\begin{aligned} S_0 &:= \{s \mid (f_{\max}(s) < (14/32)n)\} \\ S_1 &:= \left\{s \mid (f_{\max}(s) \geq (14/32)n) \wedge \left(|c(s)|_0 > 0\right)\right\} \\ S_2 &:= \left\{s \mid (f_{\max}(s) \geq (14/32)n) \wedge \left(|c(s)|_0 = 0\right)\right\} \end{aligned}$$

and see that this coincides with the partition we defined for f_{JH}. Therefore, the rest of the proof follows the same way. This demonstrates that for plus-selection the rather contrived fitness function f_{JH} and the more natural fitness function $f_{\max}$ do not differ essentially.

For f_{LCS}, the proof does not carry over that easily. We define a different LCS instance, here. This time, we assume that n is a multiple of 40. As above, if this is not the case, we replace in the following n by $n' := 40 \cdot \lfloor n/40 \rfloor$ and append a sequence of $n - n'$ letters to the two sequences we define. This does not change our results if n is sufficiently large. We define the LCS instance

$$X = 0^{(3/5)n}1^{(2/5)n}$$

and

$$Y = 1^n 0^{(13/40)n}$$

and observe that the shorter sequence X has length n as desired. Clearly, common subsequences are either of the form 0^i with $0 \leq i \leq (13/40)n$ or of the form 1^i with $0 \leq i \leq (3/5)n$. Again, we define a partition S_0, S_1, S_2 of the search space $S = \{0,1\}^n$, this time in the following way:

$$\begin{aligned} S_0 &:= \{s \mid (f_{\text{LCS}}(s) < n/20)\} \\ S_1 &:= \left\{s \mid (f_{\text{LCS}}(s) \geq n/20) \wedge \left(|c(s)|_0 \geq |c(s)|_1\right)\right\} \\ S_2 &:= \left\{s \mid (f_{\text{LCS}}(s) \geq n/20) \wedge \left(|c(s)|_0 < |c(s)|_1\right)\right\} \end{aligned}$$

Consider the short sequence $X = 0^{(3/5)n}1^{(2/5)n}$. Since $(3/5)n$ is significantly larger than $(2/5)n$, we can expect to have $|c(s)|_0 > |c(s)|_1$ for all individuals in the initial population. Since the population size μ is polynomially bounded application of Chernoff bounds yields that this is the case for the complete initial population with probability $1 - e^{-\Omega(n)}$. Moreover, with the same probability each member of the initial population represents a candidate solution with $|c(s)|_0 - |c(s)|_1 \geq n/20$ and a fitness value of at least $n/20$. Thus, with probability $1 - e^{-\Omega(n)}$ the initial population is completely contained in S_1. Due to the strict plus-selection no member of the population can ever belong to S_0 in this case. Thus, we only have to estimate the probability that S_2 is reached via mutation or uniform crossover from S_1.

The crucial observation is that for $s \in \{0,1\}^n$ the following holds. If we have $|c(s)|_0 \geq |c(s)|_1$, then $f_{\text{LCS}}(s) = |c(s)|_0 - |c(s)|_1$ holds. Otherwise, for $|c(s)|_0 < |c(s)|_1$, $f_{\text{LCS}}(s) = |c(s)|_1 - |c(s)|_0$ holds. We know that the current population belongs completely to S_1. This implies that $|c(s)|_0 \geq |c(s)|_1$ and $f_{\text{LCS}}(s) \geq n/20$ both hold. We conclude that we have $|c(s)|_0 - |c(s)|_1 \geq n/20$. In the same way we see that for all $s \in S_2$, $|c(s)|_1 - |c(s)|_0 \geq n/20$ holds. Thus, for a mutation leading from S_1 to S_2, at least $n/10$ bits need to flip in a single mutation. Such a mutation has probability $e^{-\Omega(n \log n)}$. Thus, optimization by means of mutations is sufficiently unlikely. Using a reasoning similar to the one above (with a slightly different distinction of cases) we can show that the probability to reach S_2 from S_1 via uniform crossover and a subsequent mutation is bounded above by $e^{-\Omega(n)}$. This completes the proof for f_{LCS}. □

We remark that we can prove stronger bounds when we consider specific parameterizations of the generic evolutionary algorithms. We remark that it is also not difficult to prove stronger bounds when willing to invest more work in the proofs. Since we have proved that the worst case run time is exponential with probability exponentially close to 1, we do not consider such improvements worth the effort.

Often, it is easier to find solutions that are only approximations of optimal solutions. For example, for the maximum matching problem, it is known that the simple (1+1) EA can find an $(1+\varepsilon)$-approximation to a maximum matching for any constant $\varepsilon > 0$ on average in polynomial time while there are instances known where finding an optimal solution takes on average an exponential number of steps [5]. Therefore, it makes sense to investigate if the difficulty of LCS for evolutionary algorithms is significantly reduced if we are satisfied with solutions that are only slightly worse than optimal solutions. For LCS, however, we can prove that this is not the case. Even finding common subsequences that are only slightly longer than half as long as a longest common subsequence is not significantly simpler than finding longest common subsequences — using the worst case perspective as usual.

THEOREM 2. *For any constant $\varepsilon > 0$, the probability that an evolutionary algorithm as defined in Definition 4 finds an $(2-\varepsilon)$-approximation to an optimal solution for a worst case instance of LCS $X, Y \in \{0,1\}^*$ with $X \in \{0,1\}^n$ using the fitness function f_{JH}, $f_{\max}$, or f_{LCS} within t function evaluations is bounded above by $t \cdot e^{-\Omega(n)}$ for all settings of p_c, $\mu = n^{O(1)}$, and $\lambda = n^{O(1)}$.*

PROOF. The proof is structurally identical to the proof of Theorem 1. Note that there we did not only prove a

lower bound on the time needed to find a longest common subsequence. In fact we proved that on average it takes very long to find any candidate solution from S_2. We can use the same proof technique if our LCS instance and the partition of the search space $S = \{0,1\}^n$ into disjoint sets S_0, S_1, S_2 ensures that only bit strings in S_2 represent $(2-\varepsilon)$-approximations.

Due to the structural similarity of $f_{\max}$ and f_{JH}, we can consider the same LCS instance for these two fitness functions. Here we have some constant $\varepsilon > 0$ and have to define instances for sufficiently large n where "sufficiently large" may depend on ε (since this is a constant). We define $l := \lceil (3/\varepsilon) - 1/2 \rceil$ and assume that n is a multiple of $8l$. Since $8l$ only depends on ε, we can use the same technique as in the proof of Theorem 1 if this is not the case.

Consider the LCS instance[1]

$$X = 0^{n/l}1^{((l-1)/l)n} \text{ and } Y = 1^{((l-1)/l)n}0^{(5/(8l))n}1^{((4l-3)/(8l))n}$$

where, clearly, the shorter sequence X has length n. Defining the partition (with $\gamma := 180,000+3,000\cdot((4l-2)/(8l))n$)

$$\begin{aligned} S_0 &:= \{s \mid (f_{\text{JH}}(s) < \gamma)\} \\ S_1 &:= \left\{s \mid (f_{\text{JH}}(s) \geq \gamma) \wedge \left(|c(s)|_0 > 0\right)\right\} \\ S_2 &:= \left\{s \mid (f_{\text{JH}}(s) \geq \gamma) \wedge \left(|c(s)|_0 = 0\right)\right\} \end{aligned}$$

for f_{JH} and the partition

$$\begin{aligned} S_0 &:= \{s \mid (f_{\max}(s) < ((4l-2)/(8l))n)\} \\ S_1 &:= \left\{s \mid (f_{\max}(s) \geq ((4l-2)/(8l))n) \wedge \left(|c(s)|_0 > 0\right)\right\} \\ S_2 &:= \left\{s \mid (f_{\max}(s) \geq ((4l-2)/(8l))n) \wedge \left(|c(s)|_0 = 0\right)\right\} \end{aligned}$$

for $f_{\max}$, we can show analogously to the proof of Theorem 1 that S_2 is reached with probability $t \cdot e^{-\Omega(n)}$ in t generations. For $s \in S_1$, we have $|c(s)| \leq (5/(8l))n + ((4l-3)/(8l))n$ while the longest common subsequence has length $((l-1)/l)n$. Thus, without reaching S_2, only a $\frac{((l-1)/l)n}{((4l-3)/(8l))}$-approximation can be found. We observe that $\frac{((l-1)/l)n}{((4l-3)/(8l))} = \frac{4l-4}{2l+1}$ holds. We have $l \geq (3/\varepsilon) - 1/2$, so $(4l-4)/(2l+1) \geq 2-\varepsilon$ follows as claimed.

As in the proof of Theorem 1, we consider a different instance for f_{LCS}. We set $l := \lceil (5/(2\varepsilon)) - 5/4 \rceil$ assume that n is a multiple of $16l+8$. If this is not the case we can proceed as usual. For the proof we consider the LCS instance

$$X = 0^{((l+1)/(2l+1))n}1^{(l/(2l+1))n} \text{ and } Y = 1^n 0^{((4l+5)/(16l+8))n}$$

and define the partition

$$\begin{aligned} S_0 &:= \{s \mid (f_{\text{LCS}}(s) < (2/(16l+8))n)\} \\ S_1 &:= \left\{s \mid (f_{\max}(s) \geq (2/(16l+8))n) \wedge \left(|c(s)|_0 \geq |c(s)|_1\right)\right\} \\ S_2 &:= \left\{s \mid (f_{\max}(s) \geq (2/(16l+8))n) \wedge \left(|c(s)|_0 < |c(s)|_1\right)\right\} \end{aligned}$$

as partition of the search space $S = \{0,1\}^n$. Using arguments of the kind used above we see that without reaching S_2 at best a $\frac{(l/(2l+1))n}{((4l+6)/(16l+8))}$-approximation is possible. Due to our definition of l, this completes the proof. □

[1]The given instance works for approximation factors $2-\varepsilon$ with $0 < \varepsilon < 6/7$. For even larger ε, the simpler instance $X = 0^{n/4}1^{(3/4)n}$, $Y = 1^{(3/4)n}0^{(5/32)n}1^{(13/32)n}$ works. Due to space restrictions, we skip the proof for this simpler case.

5. RESTRICTED NUMBERS OF REPETITIONS

In the previous section, we argued that we considered a quite restricted and rather simple class of LCS instances. We restricted ourselves to instances over the alphabet $\Sigma = \{0,1\}$ consisting of only two sequences. Reconsidering our worst-case instances, we recognize that all our instances are structured similarly. The most prominent feature are long series consisting of the same letter without any interruption. One may speculate that it is this property that makes these instances difficult for evolutionary algorithms. We can classify LCS instances according to the longest sequence of a single letter occurring in the shortest sequence. In our examples, we always have linear length $\Theta(n)$ of these subsequences. Here, we restrict our interest to instances where this length of a longest sequence of a single letter is restricted quite drastically. In particular, we consider only instances where this length is $O(1)$, i. e., restricted by some constant independent of n. It turns out that this further restriction is still not sufficient to make LCS easy for the kind of evolutionary algorithms we consider. We do make use, however, of a slightly larger alphabet. We use two additional letters (leading us to $\Sigma = \{0,1,3,4\}$) that enable us to extend the instances discussed in the previous section such that they fit the restriction imposed here. We do not claim that there are no other ways showing similar results for even smaller alphabet sizes $|\Sigma|$. It should be noted that $|\Sigma| = 4$ is not at all large, in particular, it coincides with the well-known alphabet $\Sigma_{\text{DNA}} = \{\text{A}, \text{C}, \text{G}, \text{T}\}$.

THEOREM 3. *The probability that an evolutionary algorithm as defined in Definition 4 finds an optimal solution to a worst case instance of LCS $X, Y \in \{0,1,2,3\}^*$ with $X \in \{0,1\}^n$ using the fitness function f_{JH}, $f_{\max}$, or f_{LCS} within t function evaluations is bounded above by $t \cdot e^{-\Omega(n)}$ for all settings of p_c, $\mu = n^{O(1)}$, and $\lambda = n^{O(1)}$, even if in each sequence the length of subsequences of the same letter is bounded above by $O(1)$.*

PROOF (SKETCH). In the proof of Theorem 1, we considered the instance

$$X = 0^{n/4}1^{(3/4)n} \text{ and } Y = 1^{(3/4)n}0^{(5/32)n}1^{(13/32)n}$$

for f_{JH} and $f_{\max}$. Let $k \in \mathbb{N}_0$ be some constant with $k > 17$. For f_{JH} and $f_{\max}$, we consider the instance

$$X = \left(0^k 2\right)^{(n/4)/(k+1)} \left(1^k 2\right)^{((3/4)n)/(k+1)},$$

$$Y = \left(1(23)^{\frac{n}{k+1}}\right)^{\frac{k(3/4)n}{k+1}} \left(0(23)^{\frac{n}{k+1}}\right)^{\frac{k(5/32)n}{k+1}} \left(1(23)^{\frac{n}{k+1}}\right)^{\frac{(13/32)n}{k+1}}.$$

We see that the structure of the LCS instance from the proof of Theorem 1 is preserved while the length of uninterrupted sequences of the same letter is reduced to k. A lengthy proof without significant new ideas shows that the desired properties are preserved, too. For f_{LCS}, again, a different LCS instance is used to show an analogous result using

$$X = \left(0^k 2\right)^{((3/5)n)/(k+1)} \left(1^k 2\right)^{((2/5)n)/(k+1)},$$

$$Y = \left(1^k 3\right)^{n/(k+1)} \left(0^k 3\right)^{((13/40)n)/(k+1)}$$

instead of

$$X = 0^{(3/5)n}1^{(2/5)n} \text{ and } Y = 1^n 0^{(13/40)n}.$$

□

6. CONCLUSIONS

We considered the performance of evolutionary algorithms for the longest common subsequence problem from a theoretical point of view. Using a coding of the problem due to Julstrom and Hinkemeyer [9] who reported good EA performance and considering two additional fitness functions we proved that a rather large class of evolutionary algorithms is not at all efficient for this problem. This evolutionary algorithms make use of plus-selection, standard bit mutation, and uniform crossover. Very different evolutionary algorithms, simple ones like the (1+1) EA and more advanced ones like steady-state genetic algorithms with uniform crossover, belong to this class of evolutionary algorithms. We proved exponential lower bounds on the expected optimization time for these evolutionary algorithms. This holds even when we restrict the LCS instances to instances over the alphabet $\Sigma = \{0, 1\}$, containing exactly two sequences. We can even prove such results for instances having uninterrupted sequences of single letters of length only $O(1)$ if we increase the alphabet size to 4. Furthermore, we proved that the worst case run time is still exponential if we do not ask for optimal solutions but are satisfied with $(2 - \varepsilon)$-approximations of a longest common subsequence.

Clearly, this contribution lacks any positive results. We have not identified non-trivial classes of LCS instances where some evolutionary algorithm fitting within Definition 4 has polynomial expected run time. It would also be interesting to prove positive results on approximations. We have ruled $(2 - \varepsilon)$-approximations. We doubt that the approximation factor $2-\varepsilon$ is in any sense tight. Thus, even positive results for approximation factors growing with n are of interest. A different sensible route for future research is the identification of evolutionary algorithms that perform better on this problem. We doubt that the use of a less strict selection or the use of a different general purpose crossover operator (like 1-point crossover) alone are sufficient to achieve this. When considering other evolutionary algorithms one has to be careful not to start to design problem-specific evolutionary algorithms: problem-specific algorithms for LCS are already known. The point is not to prove for a very specific evolutionary algorithm that it performs well on LCS. The point is to identify a rather "standard" evolutionary algorithm that performs well on LCS without much tuning.

The most obvious open question clearly is related with our starting point, the work by Julstrom and Hinkemeyer [7, 9]. Our results are strongly negative and in clear contradiction with the positive empirical results reported. Unfortunately, the documentation given is insufficient to allow for reproducing the good results. It is not even clear what problem instances have been successfully solved by the evolutionary algorithm. This is not an uncommon problem within the evolutionary computation community [8]. We hope that either from the side of practitioners or the side of theoreticians, perhaps following the route outlined above, the gap between these contradictory results can be closed.

7. ACKNOWLEDGMENTS

The first author was supported by the DFG (Deutsche Forschungsgemeinschaft) as part of the collaborative research center "computational intelligence" (SFB 531).

8. REFERENCES

[1] H.-G. Beyer, H.-P. Schwefel, and I. Wegener. How to analyze evolutionary algorithms. *Theoretical Computer Science*, 287(1):101–130, 2002.

[2] P. Briest, D. Brockhoff, B. Degener, M. Englert, C. Gunia, O. Heering, T. Jansen, M. Leifhelm, K. Plociennik, H. Röglin, A. Schweer, D. Sudholt, S. Tannenbaum, and I. Wegener. Experimental supplements to the theoretical analysis of eas on problems from combinatorial optimization. In *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII)*, pages 21–30, 2004.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[4] K. DeJong. *Evolutionary Computation: A Unified Approach*. MIT Press, 2006.

[5] O. Giel and I. Wegener. Maximum cardinality matchings on trees by randomized local search. In *Proceedings of the 8th Annual Genetic and Evolutionary Computation Conference (GECCO 2006)*, pages 539–546, 2006.

[6] T. Hagerup and C. Rüb. A guided tour of chernoff bounds. *Information Processing Letters*, 33(6):305–308, 1990.

[7] B. Hinkemeyer and B. A. Julstrom. A genetic algorithm for the longest common subsequence problem. In *Proceedings of the 8th Annual Genetic and Evolutionary Computation Conference (GECCO 2006)*, pages 609–610, 2006.

[8] T. Jansen and R. P. Wiegand. Bridging the gap between theory and practice. In *Proceedings of the 8th International Conference on Parallel Problem Solving From Nature (PPSN VIII)*, pages 61–71, 2004.

[9] B. A. Julstrom and B. Hinkemeyer. Starting from scratch: Growing longest common subsequences with evolution. In *Proceedings of the 9th International Conference on Parallel Problem Solving From Nature (PPSN IX)*, pages 930–938, 2006.

[10] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[11] H. Mühlenbein. How evolutionary algorithms really work: Mutation and hillclimbing. In *Proceedings of the 2nd International Conference on Parallel Problem Solving from Nature (PPSN II)*, pages 15–26, 1992.

[12] F. Neumann. Expected runtimes of evolutionary algorithms for the eulerian cycle problem. *Computers and Operations Research*, 2007. To appear.

[13] F. Neumann and I. Wegener. Minimum spanning trees made easier via multi-objective optimization. *Natural Computing*, 5(3):305–319, 2006.

[14] J. Scharnow, K. Tinnefeld, and I. Wegener. The analysis of evolutionary algorithms on sorting and shortest paths problems. *Journal of Mathematical Modelling and Algorithms*, 3:349–366, 2004.

[15] T. Storch. How randomized search heuristics find maximum cliques in planar graphs. In *Proceedings of the 8th AnnualGenetic and Evolutionary Computation Conference (GECCO 2006)*, pages 567–574, 2006.