Simulated Annealing, its Parameter Settings and the Longest Common Subsequence Problem

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

ABSTRACT

Simulated Annealing is a probabilistic search heuristic for solving optimization problems and is used with great success on real life problems. In its standard form Simulated Annealing has two parameters, namely the initial temperature and the cooldown factor. In literature there are only rules of the thumb for choosing appropriate parameter values. This paper investigates the influence of different values for these two parameters on the optimization process from a theoretical point of view and presents some criteria for problem specific adjusting of these parameters.
With these results the performance of the Simulated Annealing algorithm on solving the Longest Common Subsequence Problem is analysed using different values for the two parameters mentioned above. For all these parameter settings it is proved that even rather simple input instances of the Longest Common Subsequence Problem can neither be solved to optimality nor approximately up to an approximation factor arbitrarily close to 2 efficiently.

Categories and Subject Descriptors

G.3 [**Probability and Statistics**]: Probabilistic algorithms; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures, Pattern matching*

General Terms

Algorithms, Performance, Theory

Keywords

Simulated Annealing, Run Time Analysis, Longest Common Subsequence Problem

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.

1. INTRODUCTION

Randomised search heuristics like Simulated Annealing, evolutionary algorithms and local search algorithms are applied successfully in many different fields. The most important area is optimization and approximization of problems where no efficient problem specific algorithm is known. In this field theoretical analyses of these heuristics are way behind their practical applications.
Rigorous performance and runtime analysis of evolutionary algorithms started about 15 years ago. At that time very simple search algorithms on artificial problems were considered. Today more complex algorithms are subject in this field and the analyses are no longer restricted to artificial problems. Concrete examples for combinatorial optimization problems considered in the last years are the Maximum Matching Problem [3], the computation of an Eulerian cycle [10], the computation of Minimum Spanning Trees [11], the Single Source Shortest Paths Problem [12], the Permutation Sorting Problem [12] and the Longest Common Subsequence Problem [6].
In this paper we focus on Simulated Annealing and analyse its performance on the Longest Common Subsequence Problem. With our investigations we target two different goals. On the one hand it is not clear how the performance of Simulated Annealing depends on its parameter settings. This topic is covered in the first part of this paper and leads to some general results that should be considered for adjusting the parameters in a problem specific way. On the other hand we investigate the runtime performance of Simulated Annealing with different parameter settings on chosen input instances of the Longest Common Subsequence Problem. This work is based on an analysis of evolutionary algorithms for the Longest Common Subsequence Problem [6]. It has been shown that certain input instances cannot be solved efficiently by a huge class of evolutionary algorithms. The main problem was that there are local optima with huge basins of attraction preventing the algorithms from finding optimal solutions or even $(2-\varepsilon)$-approximations efficiently. An open question was, if there are other (not problem specific) search heuristics that can perform better on those input instances or on the Longest Common Subsequence Problem in general. Since Simulated Annealing has the ability to leave local optima such an algorithm would be a promising approach.
We begin with a formal definition of the Simulated Annealing Algorithm we use in our investigations, including its parameters. After that we will prove some general results

about the influence of the parameters on the algorithm's performance. Then we will introduce the Longest Common Subsequence Problem, a coding for candidate solutions and three fitness functions in a formal way. With these components we can prove some performance bounds on certain input instances followed by a sketch of some further results. At the end we will discuss the obtained results and give an outlook about possible starting points for future work on this topic.

2. SIMULATED ANNEALING AND ITS PARAMETERS

For a rigurous theoretical analysis it is of course necessary to specify exactly what algorithm is analyzed. In this section we specify the algorithm that is subject of the following analyses together with its parameters.

The algorithm we investigate is a kind of standard Simulated Annealing Algorithm operating on fixed length bit strings. This algorithm has, besides the fitness function and the input length, two different parameters, namely the initial temperature and the cooldown factor. These two parameters do not change during the execution of the algorithm.

At the beginning the temperature is set to the value of the initial temperature and an initial solution is created uniformly at random. In each iteration the current solution is modified by means of 1-bit mutation (1-bit mutation flips exactly one bit, this bit is chosen uniformly at random). If the fitness value of the new solution is better or equal than the fitness value of the current solution, the new solution will replace the current solution for the next iteration, otherwise the new solution will replace the current solution only with a probability of $\exp\left(-\frac{\Delta f}{T}\right)$, where Δf is the (positive) difference of the fitness values and T is the current temperature. If the current solution after that step fulfills the stopping criterion, the algorithm stops, otherwise the temperature will be updated by multiplying the current temperature with the cooldown factor and the algorithm continues with the next iteration. The following definition is a precise description of this algorithm in pseudo-code for a maximization problem.

Definition 1. **Standard Simulated Annealing on fixed length bit strings**

Parameters are the initial temperature T_0, the cooldown factor α ($0 < \alpha < 1$), the bit string length (resp. the input size) $n \in \mathbb{N}$ and the fitness function $f : \{0,1\}^n \to \mathbb{R}$.

1. Create $x \in \{0,1\}^n$ uniformly at random.
2. Set the iteration count $t := 0$.
3. Create y from x by 1-bit mutation.
4. If $f(y) \geq f(x)$, set $x := y$,
   otherwise set $x := y$ with probability $\exp\left(-\frac{f(x)-f(y)}{T_0 \cdot \alpha^t}\right)$.
5. If the stopping criterion is fulfilled, then stop.
6. Set $t := t + 1$.
7. Go to step 3.

In our analyses we are interested in the number of iterations until the current solution is a global optimum or a sufficient approximation for the first time. As usual we are mainly interested in an asymptotic value for the number of iterations regarding the bit string length. For the remainder of this section, we will be occupied with the initial temperature and the cooldown factor and we will show some general results regarding these parameters.

In [14] the author suggests to use a high initial temperature and a cooldown factor between 0.8 and 0.99 for solving optimization problems in practice. Other sources suggest an initial temperature that implies an acceptance probability of about 80% for worse solutions at the beginning of the algorithm. For theoretical analyses these general suggestions are not very helpful, since the probability for accepting a worse solution is independent of the input size and so it decreases too fast below an exponentially small value. That means, the number of iterations to leave a local optimum is too small with constant values for the initial temperature and the cooldown factor. So far it is not clear, which parameter values are sensible and therefore we will investigate many different classes of parameter values in this paper.

In the remaining part of this section we will discuss some classes of parameter settings. We will assume that the difference of fitness values for any two solutions is either 0 or at least 1 (instead of 1 any constant $c > 0$ can be used, not changing the results). This assumption holds for the fitness functions $f_{\max}$, f_{lcs} and f_{jh}, that we will use later for the analyses of Simulated Annealing on the Longest Common Subsequence Problem, and it also holds for many other well known fitness functions. Examples for such fitness functions can be found in [5] for the Longest Common Subsequence Problem, in [12] for the Permutation Sorting Problem and in [3] for the Maximum Cardinality Matching Problem. Furthermore other combinatorial optimization problems have natural fitness functions with this property. So it is not a strong restriction and some of the following results also hold without this restriction. The second assumption is that we have to leave at least one local optimum during the run of the algorithm (with certainty or with high probability). This is not a restriction at all, because Simulated Annealing should have the property to leave local optima and if there are no local optima, Randomised Local Search would be the better choice.

First, we assume that $T_0 = o(\ln(\text{poly}(n))^{-1})$ holds for every polynomial $\text{poly}(n)$ in n. The cooldown factor α can be any value between 0 and 1. With this initial temperature, the probability to accept a worse solution in the first iteration is at most $\exp(-\frac{1}{T_0}) = o(\text{poly}(n)^{-1})$ for every polynomial $\text{poly}(n)$ in n. Since the acceptance probabilites are decreasing, the expected number of iterations for the acceptance of a worse solution is superpolynomial and so is the expected number of iterations for leaving a local optimum. Furthermore the probability to leave a local optimum within t iterations can be bounded from above by $t \cdot o(\text{poly}(n)^{-1})$ using any polynomial. That means such parameter settings do not lead to an efficient Simulated Annealing Algorithm for solving problems with local optima and so these settings are not part of the later analyses.

Now, let us assume that we have an arbitrary initial temperature and a cooldown factor of $\alpha = 1 - \mathcal{O}(c^{-n})$ with a constant $c > 1$. Then we need exponentially many iterations to achieve a cooldown of a constant factor. So it is likely that the algorithm behaves within exponentially many iterations like the Metropolis Algorithm, i.e. Simulated An-

nealing with a constant temperature. This fact in mind, we will handle this case in a separate analysis later.
The remaining parameter settings define the number of iterations, in which the algorithm is able to leave local optima. A low initial temperature and a small cooldown factor lead to a small time window for leaving local optima, a high initial temperature and a cooldown factor very near to 1 lead to a huge time window for leaving local optima. On the other hand, once the basin of attraction of a global optima is reached, a low initial temperature and a small cooldown factor lead to a faster convergence to this optimum in general, whereas an algorithm with a high initial temperature and a cooldown factor near to 1 needs more iterations to reach that optimum. So there is a kind of tradeoff between the number of iterations, within local optima can be left, and the number of iterations the algorithm needs to reach a global optimum once there is a solution in its basin of attraction.
With this very general investigations of the parameter settings, we can draw the following conclusions. A very low initial temperatur T_0, with $T_0 = o(\ln(\text{poly}(n))^{-1})$ for every polynomial $\text{poly}(n)$ in n, is not useful at all. With a cooldown factor of $\alpha = 1 - \mathcal{O}(c^{-n})$, with a constant $c > 1$, the algorithm behaves like the Metropolis Algorithm within exponentially many iterations. And the different values in the remaining class of parameter settings represent a tradeoff between the number of iterations, within local optima can be left, and the number of iterations the algorithm needs to reach a global optimum once there is a solution in its basin of attraction. So this values should be chosen in a problem specific way regarding the fitness function. Figure 1 illustrates these facts.

3. THE LONGEST COMMON SUBSEQUENCE PROBLEM

In this section we will give a precise definition of the Longest Common Subsequence Problem. After that we will present the search space and representation of individuals used in [7] together with the fitness functions used in [7] and [6] that are relevant for our analyses.
Before we are able to define the Longest Common Subsequence Problem we have to clarify some terms. The Longest Common Subsequence Problem is defined over some finite alpabet Σ. In computer science the alphabet $\Sigma = \{0, 1\}$ is commonly used, in bioinformatics an alphabet with 4 elements, $\Sigma = \{A, C, G, T\}$, for the coding of DNA sequences is of particular interest and for our analyses we use the alphabet $\Sigma = \{X, Y\}$ for a better distinction between bitstrings and the individuals they represent. We consider sequences of letters from Σ, also called strings, of finite lengths. For a given string s we denote by $|s|$ the length of s, i.e. the number of letters from Σ in s. For a letter $l \in \Sigma$ we write $|s|_l$ to refer to the number of occurrences of l in s. For example, for the string $s = XXXYXYX$ over the alphabet $\Sigma = \{X, Y, Z\}$ we have $|s| = 7$, $|s|_X = 5$, $|s|_Y = 2$ and $|s|_Z = 0$. Furthermore, we write $s_{(i)}$ with $i \in \{0, 1, \ldots, |s|\}$ for the prefix of length i of s. As usual, we use ε as the symbol for the empty string of length 0. For convenience, we use the notation s^i for a sequence s and $i \in \mathbb{N}_0$ for a repetition of s for i times. Thus, $X^3 = XXX$ and $(YX)^2 = YXYX$. Using this notation and writing concatenations of strings without any special symbol, we have $XXXYXYX = X^3(YX)^2$.
For an alphabet Σ, we refer to the set of all strings of lengths exactly i, $i \in \mathbb{N}_0$, with Σ^i. The set of all strings of finite length over Σ is called $\Sigma^\star = \bigcup_{i \in \mathbb{N}_0} \Sigma^i$.
Now some terms, that lead us directly to the definition of the Longest Common Subsequence Problem. Given two string $A = a_1 a_2 \ldots a_n \in \Sigma^n$ and $B = b_1 b_2 \ldots b_m \in \Sigma^m$ we call B a subsequence of A if there is a increasing sequence of indices $0 < i_1 < i_2 < \ldots < i_m \leq n$ such that $a_{i_j} = b_j$ holds for all $j \in \{1, 2, \ldots, m\}$. For example, $XXYXX$ is a subsequence of $XXXYXYX$ while $XXYYXX$ is not a subsequence of this string. The sequence of indices proving that some string is a subsequence of another string is not necessarily unique. For example $1, 2, 4, 5, 7$ and $1, 3, 4, 5, 7$ are both valid sequences that prove that $XXYXX$ is a subsequence of $XXXYXYX$.
For a given finite set of strings $A_1, A_2, \ldots, A_m \in \Sigma^\star$ over the same alphabet Σ, we call a string $B \in \Sigma^\star$ a common subsequence, if B is a subsequence of A_i for all $i \in \{1, 2, \ldots, m\}$. B is actually called a longest common subsequence, if it has maximum cardinality, that means, if for all other strings $B' \in \Sigma^\star$ that are common subsequences of $A_1, A_2, \ldots, A_m$ the condition $|B'| \leq |B|$ holds.
With these terms we are now able to define the Longest Common Subsequence Problem precisely. Given a finite number of strings $A_1, A_2, \ldots, A_m \in \Sigma^\star$ over some finite alphabet Σ, the task is to find a longest common subsequence of the given strings. As the problem formulation suggests, the solution is not uniquely determined in general. For example, considering $A_1 = YXXXYY$, $A_2 = XYYYXXXX$ and $A_3 = XXYYXX$, we see that XYY and YXX are both longest common subsequences.
At this point it should be mentioned that the Longest Common Subsequence Problem is one of the fundamental problems in computer science and it has many applications, especially in bioinformatics. There exists a problem specific algorithm (based on a dynamic programming approach) solving the problem for a fixed number of m sequences of lengths l_1, l_2, ..., l_m in time $\mathcal{O}\left(\prod_{i=1}^{n} l_i\right)$ [2].
To be able to use Simulated Annealing we need a precise definition of the search space and the coding of search points, in this context also called individuals. For our analysis we use a simple binary encoding that has been proposed and used by Julstrom and Hinkemeyer in [5] and [7]. Let $A_1, A_2, \ldots, A_m \in \Sigma^\star$ be the given strings. Without loss of generality we assume that A_1 is a shortest of this strings, i.e. $|A_1| \leq |A_i|$ for all $i \in \{1, 2, \ldots, m\}$. Clearly, the length of a longest common subsequence is bounded above by the length of the shortest sequence. Let $n = |A_1|$, then we can represent a candidate solution $B \in \Sigma^\star$ by some bitstring $s \in \{0, 1\}^n$ of fixed length n. A bit set to 1 indicates a letter in A_1 that is used for the candidate solution, while a bit set to 0 indicates, that the corresponding letter is left out. For example, for $A_1 = ACGTA$, 00010 represents T, 11101 represents $ACGA$. The all 1-string 1^n represents A_1, while the all 0-string 0^n represents ε. This way $s \in \{0, 1\}^n$ may represent strings 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. In the following we use a function $c : \{0, 1\}^n \to \Sigma^\star$ that maps bit strings to the candidate solutions they represent.
The last ingredient we need for using Simulated Annealing is a fitness function. In this paper we will focus on three fitness

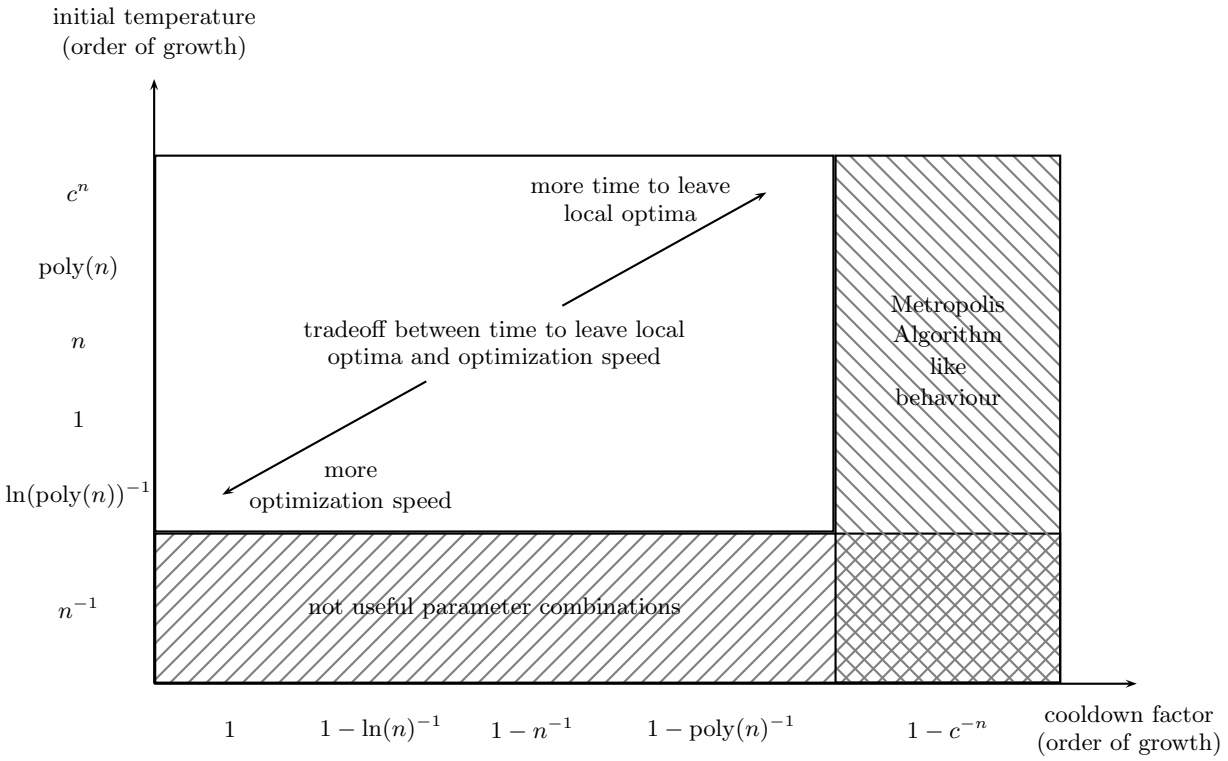

Figure 1: Simulated Annealing parameter settings

functions. The first one has been proposed by Julstrom and Hinkemeyer in [5] and [7], the other ones are more artificial and have been used for the analyses in [6].

We call the fitness function proposed by Julstrom and Hinkemeyer f_{jh}. This fitness function maps search points representing feasible candidate solutions to positive fitness values and search points representing infeasible candidate solutions to negative values. Assuming $A_1, A_2, \ldots, A_m \in \Sigma^\star$ are the input strings for the Longest Common Subsequence Problem, $|A_1| \leq |A_i|$ holds for all $i \in \{1, 2, \ldots, m\}$ and $c : \{0,1\}^n \to \Sigma^\star$ is defined as above, we can define the fitness function as follows.

Definition 2. **Fitness function** $f_{\text{jh}} : \{0,1\}^n \to \mathbb{Z}$

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

Here $k(s)$ is the number of strings of $A_1, A_2, \ldots, A_m$ for which $c(s)$ is a subsequence.

It is not clear, why the special case $|c(s)| = n$ gets extra reward and extra attention. In general, no feasible solution with $|c(s)| = n$ needs to exist. Fortunately this aspect is not relevant for our theoretical analyses.

The idea behind the next fitness function, $f_{\max}$, is to map the letters in $c(s)$ from left to right to letters in each of the strings $A_1, A_2, \ldots, A_m$. Each letter that can be mapped (for all strings $A_1, A_2, \ldots, A_m$) leads to a reward of $+1$, letters that cannot be mapped (for at least one of the strings) lead to a decrease by -1. For $i \in \{1, 2, \ldots, m\}$ we call the length of the longest prefix of $c(s)$ that is a subsequence of A_i

$$\text{pref}(i) = \max\{\, k \in \{0, 1, \ldots, |A_i|\} : c(s)_{(k)} \text{ is a subsequence of } A_i \}.$$

Then the number of letters that lead to a reward of $+1$ is $\text{gain}(s) := \min\{\text{pref}(i) : i \in \{1, 2, \ldots, m\}\}$ and the number of letters that cause a decrease by -1 is $|c(s)| - \text{gain}(s)$. With these notations we can define the fitness function in the following way.

Definition 3. **Fitness function** $f_{\max} : \{0,1\}^n \to \mathbb{Z}$

$$\begin{aligned} f_{\max}(s) &= \text{gain}(s) - (|c(s)| - \text{gain}(s)) \\ &= 2 \cdot \text{gain}(s) - |c(s)|. \end{aligned}$$

This fitness function also maps feasible solutions to non-negative values, but it is not the case that infeasible solutions have negative fitness values in general. Because of its easy structure, this fitness function has some properties that can be verified easily. Removing a letter from a feasible solution decreases its fitness value by -1. Adding a letter, that can be mapped leads to a reward of $+1$ and adding a letter that cannot be mapped leads to an infeasible solution with a lower fitness value. For infeasible solutions this is a bit more complicated. Adding letters can lead to a gain of $+1$ if this letter can be mapped successfully or a loss of at least -1 if this letter cannot be mapped. Removing a letter that could not be mapped leads to a reward of at least $+1$, while

removing a letter that could be mapped leads to a decrease of -1. The analyses in the following sections use these properties and distinguish between the seven different cases that could occur.
Our last fitness function is called f_{lcs}. In this fitness function the very simple left-to-right mapping is replaced by the best possible mapping, the mapping induced by a longest common subsequence. Since for such a mapping the Longest Common Subsequence Problem needs to be solved, such a fitness function makes no sense at all with respect to practical applications. But, since it is the best possible mapping, this fitness function is interesting from a theoretical point of view. Using a similar notation as for $f_{\max}$, for $i \in \{1, 2, \ldots, m\}$ we call the length of a longest common subsequence of $c(s)$ and A_i

$$\text{lcs}(i) = |l|, \quad \text{where } l \text{ is a longest common subsequence of } c(s) \text{ and } A_i.$$

Here the number of letters that lead to a reward of $+1$ is $\text{gain}'(s) := \min\{\text{lcs}(i) : i \in \{1, 2, \ldots, m\}\}$ and the number of letters that lead to a loss of -1 is $|c(s)| - \text{gain}'(s)$. Now we can define f_{lcs} in a formal way.

Definition 4. **Fitness function** $f_{\text{lcs}} : \{0,1\}^n \to \mathbb{Z}$

$$\begin{aligned} f_{\text{lcs}}(s) &= \text{gain}'(s) - (|c(s)| - \text{gain}'(s)) \\ &= 2 \cdot \text{gain}'(s) - |c(s)|. \end{aligned}$$

Having clarified which kind of Simulated Annealing algorithm and which fitness functions we are using, we can now proceed to the analyses.

4. SOME PERFORMANCE BOUNDS

In this section we will prove some bounds on the runtime of the Simulated Annealing Algorithm that hold with certainty or very high probability for all of the three fitness functions. Our analyses cover all useful combinations of the initial temperature and the cooldown factor. But before we can start with the analyses we need a result about biased random walks.
Let us assume that we start with a certain value of $x = n$. In each step we either decrease the value by -1 or increase the value by $+1$. If we have $0 < x \leq 2n$ the probability for decreasing the value is $\leq p^-$ and the probability for increasing the value is $\geq p^+$. p^- and p^+ are constants independent of n with $p^- < 1/2$ and $p^+ > 1/2$. The transition probabilities for values greater than n can be arbitrary. We are interested in an upper bound of the probability that we reach the value $x = 0$ within t steps.
Therefore we investigate n time steps starting at a value of $x = n$. We can model each of the n steps by a random variable X_i, $1 \leq i \leq n$. We set $X_i = 1$ if and only if the value increases by 1 in time step i. Since the value x in all of the n time steps we investigate is between 0 and $2n$, we have n independent 0/1 random variables each set to 1 with probability $\geq p^+$ and to 0 with probability $\leq p^-$. Now we focus on the sum of these random variables, $X = X_1 + X_2 + \ldots + X_n$. The expected value of X is $\mu_X \geq np^+$. With the use of Chernoff bounds, we can bound the probability that $X \leq n/2$.

With $\delta = \frac{2p^+ - 1}{2p^+}$ we have

$$\begin{aligned} \text{Prob}(X \leq n/2) &\leq \text{Prob}(X \leq (1-\delta)\mu_X) \\ &\leq \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mu_X} \\ &\leq c^{\Theta(n)} \text{ with } c < 1. \end{aligned}$$

That means starting at the value $x = n$ we do not reach the value 0 within n time steps and we do not have a smaller value than n after those steps with a probabiltiy of at least $1 - c^{\Theta(n)}$, with a constant $c < 1$. Calling n time steps, beginning with a value of $x = n$, a phase, we have at most t/n phases within t time steps. The probability that we do not reach the value 0 within t time steps can therefore be bounded from below by $1 - t/n \cdot c^{\Theta(n)}$. From another perspective, the probability that we reach a value of 0 within t time steps can be bounded from above by $t/n \cdot c^{\Theta(n)}$.
With this result for biased one dimensional random walks, we can begin with our actual analyses.
We start with an analysis of the Metropolis Algorithm on a very trivial input instance, namely the input instance $A = X^n$ consisting of only one string. Clearly this problem is equivalent to the well known OneMax for all three fitness functions. The following analysis also holds for the Simulated Annealing Algorithm with a cooldown factor exponentially close to 1 within exponentially many steps. We will now show that both algorithms have problems reaching the global optimum, if the acceptance probability for worse solutions is high enough. Therefore we assume that the probability for accepting worse solutions is $p_{\text{accept}} = \Omega(n^{-1/2})$. For the Simulated Annealing Algorithm with a cooldown factor exponentially close to 1 this property holds for at least exponentially many steps.
For an individual s with $|c(s)|_X \geq n - n^{1/3}$ the probability for creating a worse solution and then accepting it is $p^+ \geq \frac{n - n^{1/3}}{n} \cdot p_{\text{accept}}$. (Here, and also in later analyses, we use p^+ and p^- according to the notation of the biased one dimensional random walk.) The probability for creating a better solution is $p^- \leq \frac{n^{1/3}}{n}$. Ignoring steps that do not change the individual, we can model this behaviour by a biased random walk. With our choise of p_{accept} we have $p^+/p^- \geq 2$ for sufficiently large values of n and therefore we can apply the result from the beginning of this section obtaining the following theorem.

Theorem 1. *The probability that the Metropolis Algorithm finds an optimal solution for the Longest Common Subsequence Problem on the input instance $A = X^n$ for any of the three fitness functions f_{max}, f_{lcs} and f_{jh} within t iterations is bounded from above by $t \cdot c^{\Theta(n^{1/3})}$ for sufficiently large values of n and a constant $c < 1$, if the acceptance probability for worse solutions is $p_{accept} = \Omega(n^{-1/2})$.*
Within exponentially many iterations, this result also holds for the Simulated Annealing Algorithm with a cooldown factor of $\alpha = 1 - \mathcal{O}(c^{-n})$, $c > 1$, if the initial probability for accepting worse solutions is $p_{accept} = \Omega(n^{-1/2})$.

Of course, we could obtain a stronger result with a more precise analysis, in particular the fraction p^+/p^- is of order $\Theta(n^{1/6})$ and not only a constant ≥ 2. But for our purpose, to show that this trivial problem instance cannot be solved

efficiently, we are satisfied with this result; and we also use this approach for the following analyses.
Now we still have to analyse the case $p_{\text{accept}} = o(n^{-1/2})$. Here the idea is that it is hard to leave local optima with large basins of attraction. We will see that the following proofs for $f_{\max}$ and f_{lcs} are independent of the acceptance probability and so they hold for any combination of the parameters initial temperature and cooldown factor. For f_{jh} this is slightly more complicated and we need to distinguish the two cases $p_{\text{accept}} = \Omega(n^{-1/2})$ and $p_{\text{accept}} = o(n^{-1/2})$. Moreover for f_{jh} we need a different analysis for values of the cooldown factor that are not exponentially close to 1. More about these analyses later, now we will give detailed proofs for $f_{\max}$ and f_{lcs}.

THEOREM 2. *With a probability exponentially close to* 1 *the probability that the Simulated Annealing Algorithm (resp. the Metropolis Algorithm) finds an optimal solution for the Longest Common Subsequence Problem on the input instance* $A = X^{(3/5)n}Y^{(2/5)n}$, $B = Y^n X^{(13/40)n}$ *for the fitness function* f_{lcs} *within* t *iterations is bounded from above by* $t \cdot c^{\Theta(n)}$ *for sufficiently large values of* n *and a constant* $c < 1$.

PROOF. With Chernoff bounds we can show that for the initial solution s the properties

$$|c(s)|_X \geq (11/40)n \quad \text{and} \quad |c(s)|_Y \leq (9/40)n$$

hold with probability exponentially close to 1. In particular we have $|c(s)|_X > |c(s)|_Y$. The crucial point in this analysis is that this property is preserved with high probability during the run of the algorithm. More precisely we show that $|c(s)|_X \geq (21/80)n$ and $|c(s)|_Y \leq (19/80)n$ is preserved with high probability. As long as we have $|c(s)|_X > |c(s)|_Y$ we can analyse the behaviour of the numbers of letters X and Y separately.
At first we focus on the behaviour of the number of the letter X. For an individual s with $(21/80)n \leq |c(s)|_X \leq (22/80)n$ an additional letter X leads to a better fitness value whereas removing an X leads to a worse fitness value. Since the number of bits that can be flipped in order to add an X is bounded from below by $(26/80)n$ and the number of bits that can be flipped in order to remove an X is bounded from above by $(22/80)n$ we have, independent of the acceptance probability, a biased random walk in this interval with favor for larger numbers of the letter X. Analogue we can show that we have a biased random walk on the number of letters Y in the interval between $(18/80)n$ and $(19/80)n$ favoring smaller numbers of the letter Y, also independent of the acceptance probability.
Since the optimal solution, $Y^{(2/5)n}$ does not have the property mentioned above, we can bound the number of iterations until an optimal solution is reached for the first time from below by the number of iterations until the property is not preserved for the first time. Together with the result about biased random walks this observation concludes the proof. □

THEOREM 3. *With a probability exponentially close to* 1 *the probability that the Simulated Annealing Algorithm (resp. the Metropolis Algorithm) finds an optimal solution for the Longest Common Subsequence Problem on the input instance* $A = X^{(1/4)n}Y^{(3/4)n}$, $B = Y^{(3/4)n}X^{(1/4)n}$ *for the fitness function* f_{max} *within* t *iterations is bounded from above by* $t \cdot c^{\Theta(n)}$ *for sufficiently large values of* n *and a constant* $c < 1$.

PROOF. For an initial solution s we can show with Chernoff bounds that $|c(s)|_X \geq (3/32)n$ holds with probability exponentially near to 1. The global optimum for the given problem instance is $Y^{(3/4)n}$ and so we need to remove all letters X from our candidate solution to reach that optimum. The decisive aspect in this analyse is that the number of letters X develops independently of the number of letters Y as long as the number of X is greater than 0. So we can focus on iterations that only affect the number of letters X in the candidate solution and do not count iterations not changing the candidate solution or only chaning the number of letters Y.
On this assumption, for individuals s with $|c(s)|_X \geq (3/32)n$, the probability p^- to create and then accept an individual s' with $|c(s')|_X = |c(s)|_X - 1$ is bounded above by $(3/8)p_{\text{accept}} \leq 3/8$. On the other hand the probability p^+ to create an individual s' with $|c(s')|_X = |c(s)|_X + 1$ is bounded from below by 5/8. Again we can conclude the proof by applying the result about the biased random walk in this particular situation. □

As we mentioned above for f_{jh} the situation is more complicated, because infeasible solutions have negative fitness values and so we do not have such simple properties as we had for $f_{\max}$ and f_{lcs}. Fortunately for the acceptance probabilities we consider in this case, the probability of creating and accepting infeasible solutions can be neglected. Due to space restrictions we will only sketch the following proof.

THEOREM 4. *With a probability exponentially close to* 1 *the probability that the Metropolis Algorithm finds an optimal solution for the Longest Common Subsequence Problem on the input instance*

$$A = X^{(1/4)n}Y^{(3/4)n},\ B = Y^{(3/4)n}X^{(1/4)n}Y^{(13/32)n}$$

for the fitness function f_{jh} *within* t *iterations is bounded from above by* $t \cdot c^{\Theta(n)}$ *for sufficiently large values of* n *and a constant* $c < 1$, *if the acceptance probability for worse solutions is* $p_{accept} = o(n^{-1/2})$.
Within exponentially many iterations, this result also holds for the Simulated Annealing Algorithm with a cooldown factor of $\alpha = 1 - \mathcal{O}(c^{-n})$, $c > 1$, *if the initial acceptance probability for worse solutions is* $p_{accept} = o(n^{-1/2})$.

PROOF. (Sketch) This proof is similar to that for the fitness function $f_{\max}$. Unfortunately f_{jh} penalizes infeasible solutions with negative values and once such a solution is reached it is very likely that many bits are flipped from 1 to 0 until we have again a feasible solution.
During the analysis on $f_{\max}$ we only had to consider solutions of length $\Theta(n)$. And fortunately the difference of fitness values of feasible and infeasible solutions of length $\Theta(n)$ is also $\Theta(n)$. With an (initial) acceptance probability for a worse solution which fitness value differs by exactly 1 of $p_{\text{accept}} = o(n^{-1/2})$ the (initial) acceptance probability for a solution which fitness value differs by $\Theta(n)$ is exponentially small in n. Since we start with probability exponentially close to 1 with a feasible solution (Chernoff bounds), we can rate the creation and acceptance of an infeasible solution as a success for the algorithm. So the algorithm can succeed by removing all letters X in the candidate solution or by creating and accepting an infeasible solution. Both events occur only with an exponentially small probability in n. Omitting technical details, this concludes the proof. □

Although we only focused on the Metropolis Algorithm and the Simulated Annealing Algorithm with a cooldown factor exponentially near to 1 we have formulated and proved results for the fitness functions $f_{\max}$ and f_{lcs} that hold for the Simulated Annealing Algorithm and all usable combinations of the parameters initial temperature and cooldown factor. For the fitness function f_{jh} the situation is more complicated. Our proofs so far only cover the Metropolis Algorithm and the Simulated Annealing Algorithm with a cooldown factor exponentially near to 1 for this fitness function. Until the end of this section we will focus on the general case for the fitness function f_{jh}.
The problem in the general case is that infeasible solutions could be created and accepted with a probability that can not be neglected anymore. We will use the same problem instance,

$$A = X^{(1/4)n}Y^{(3/4)n},\ B = Y^{(3/4)n}X^{(1/4)n}Y^{(13/32)n},$$

for the analysis of the general case but we have to modify the proof. We will split the run of the algorithm into two phases. The main idea for the analysis is that in the first phase infeasible solutions will only be created with very low probability and in the second phase the acceptance probability for infeasible solutions will be exponentially small. The second idea is that we focus on the cardinality of the letters X and Y in the candidate solution string independently.
As long as no infeasible solution occurs, we can bound the probability that the number of letters X reach the value 0 within t steps as in the previous proof. The candidate solution gets infeasible if the number of letters Y exceeds the value of $(13/32)n$. For the initial solution s we can show with Chernoff bounds that $|c(s)|_Y \leq (25/64)n$ with probability exponentially close to 1. Now we apply the idea formulated above. As long as the temperature is high enough and the acceptance probability is $p_{\text{accept}} \geq 24/25$, the behaviour of the number of letters Y in the interval between $(25/64)n$ and $(26/64)n$ can be modeled by a biased random walk favoring less letters of Y. So it is not likely that infeasible solutions occur in this case. If the acceptance probability drops below $24/25$, we can no longer model the behaviour that way. But fortunately in that case the acceptance probability for worse solutions which fitness value differs by $\Theta(n)$ is exponentially small in n. All these observations build the foundation of the proof.
Omitting technical details, this leads us to the following result.

THEOREM 5. *With a probability exponentially close to* 1 *the probability that the Simulated Annealing Algorithm (resp. the Metropolis Algorithm) finds an optimal solution for the Longest Common Subsequence Problem on the input instance*

$$A = X^{(1/4)n}Y^{(3/4)n},\ B = Y^{(3/4)n}X^{(1/4)n}Y^{(13/32)n}$$

for the fitness function f_{jh} within t iterations is bounded from above by $t \cdot c^{\Theta(n)}$ for sufficiently large values of n and a constant $c < 1$.

In this section we have shown that there are problem instances where the Metropolis Algorithm and the Simulated Annealing Algorithm with certain parameter combinations cannot find the global optimum efficiently. Our investigations cover the Metropolis Algorithm with all possible temperature settings and the Simulated Annealing Algorithm with all usable combinations of the paramters initial temperature and cooldown factor. These results will be generalized in the next section.
With the order of our analyses we have tried to illustrate the optimization process and the influence of the different parameters of the Simulated Annealing Algorithm and to emphasize the different reasons for the observed behaviour.

5. FURTHER RESULTS

In this section we generalize the results from the previous section to results about the runtime until a $(2-\varepsilon)$-approximation is reached for the first time. After that we show, that the results even hold, if we do not allow more than a constant number of repetitions of the same letter in each input string.
To generalize the results to $(2-\varepsilon)$-approximations, we can use the same proofs as in the previous section and we only need to modify the input instances a little bit. For example, in one analysis with the fitness function f_{lcs} we used the input instance

$$A = X^{(3/5)n}Y^{(2/5)n},\ B = Y^{n}X^{(13/40)n}.$$

There we showed, that it is not likely to get individuals s with $|c(s)|_X < |c(s)|_Y$. Without reaching an individual with this property, we cannot get a fitness value of more than $(13/40)n$. Since the fitness value of the global optimum is $(16/40)n$, we only get 16/13-approximations. We now can modify the ratio between the number of X and Y in the first input string and adjust the second input string properly.
With

$$A'(k) = X^{((k+1)/(2k+1))n}Y^{(k/(2k+1))n},$$
$$B'(k) = Y^{(k/(2k+1))n}X^{(1/k)n}Y^{((4k+5)/(16k+8))n},$$

$k \geq 2$, we have a family of input instances, which can be used in the same proof. In this way we can generalize the results about the runtime until an optimal solution is reached for the first time to results about the runtime until an a-approximation, for any approximation ratio $a < 2$, is reached for the first time. In particular, for each $a < 2$ there exists a $k \geq 2$ such that we can proof the desired result by using the existing proof on the input instance $A'(k)$, $B'(k)$. The other results can be transformed in a similar way to results about $(2-\varepsilon)$-approximations.
Until the end of this section we will consider certain subsets of input instances. At first, we have restricted the analyses to input instances consisting of exactly two input strings with two different letters. Of course, input instances consisting of only one string or of strings with only one letter would be trivial (that does not mean trivial to solve for all paramter settings, as we have seen).
We can create input instances with more strings by duplicating the second input string and not changing the results. Furthermore we can increase the alphabet size and append each of the new letters in the same order to each of the input strings and again preserve the results. So we have analyzed the easiest, non-trivial class of input instances regarding the number of input strings and the alphabet size.
The next restriction to our class of input instances would be to allow only a constant number of repetitions of the same letter in each input string. In all our analyses we used input instances containing repetitions of length $\Theta(n)$ of the same letter. But analogous to [6] we can construct input instances

for all of our problems (now with 3 or 4 different letters) that only consist of a constant number of repetitions of the same character and use the same proofs as before. So all our results also hold for certain instances of this restricted subset of input instances.

6. CONCLUSIONS

In this paper we have analysed the influence of the two parameters initial temperature and cooldown factor on the performance of the Simulated Annealing Algorithm from a theoretical point of view. We have worked out that some parameter settings are not useable at all and that others lead to a Metropolis Algorithm like behaviour. The usable parameter settings represent a tradeoff between the number of iterations, within local optima can be left, and the number of iterations the algorithm needs to reach a global optimum once there is a solution in its basin of attraction. These results could be helpful for theoreticians and practitioners for adjusting the parameters in an appropriate problem specific way.

The second part of the paper has been about theoretical investigations of the runtime performance of Simulated Annealing on chosen input instances of the Longest Common Subsequence Problem. Therefore we have used a coding due to Julstrom and Hinkemeyer [7] and the three fitness functions f_{jh} from [7], $f_{\max}$ and f_{lcs} from [6]. For the Simulated Annealing Algorithm we proved exponential lower bounds on the expected optimization time for all usable parameter settings. Then we generalized these results to $(2-\varepsilon)$-approximations. At the end we could restrict the input instances to strings where the longest repetition of the same letter is bounded above by a constant. Therefore the alphabet size has to be increased to 4.

We do not believe that our problem instances are worst case instances in the sense that there are no other problem instances, where the algorithm performes worse. The goal was to show that there are problem instances that cannot be solved or approximated efficiently and this goal has been achieved. Perhaps there are other problem instances which give us more insight into the optimization process of the Simulated Annealing Algorithm, but these problem instances are rather easier ones or classes of easier ones than real worst case instances. The other point is, that we do not believe that the approximation ratio of $(2-\varepsilon)$ is in any sense tight. Perhaps new results regarding the approximation ratio can be achieved in the future.

In [6] the authors pointed out that it would be interesting to find standard (i.e. not problem specific) evolutionary algorithms that perform well on the given input instances. For these input instances local optima with large basins of attraction have been the main problem. Unfortunately Simulated Annealing has similar problems handling these input instances and the different analyses should have made this behaviour clear. We think that most local search algorithms would have similar problems on these input instances and so it is still very interesting, if there are standard local search algorithms that perform better in this setting.

Other interesting aspects would be the analysis of modified Simulated Annealing Algorithms with other cooling schedules or other mutation operators. We believe that the results can be generalized to other general (not problem specific) mutation operators such as k-bit mutation or standard-bit-mutation and to other cooling schedules without changing the proof ideas significantly.

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

[3] 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.

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

[5] 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.

[6] T. Jansen and D. Weyland. Analysis of evolutionary algorithms for the longest common subsequence problem. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 1, pages 939–946, London, 7-11 July 2007. ACM Press.

[7] 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.

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

[9] 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.

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

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

[12] 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.

[13] I. Wegener. Simulated annealing beats metropolis in combinatorial optimization. In *ICALP*, pages 589–601, 2005.

[14] K. Weicker. *Evolutionäre Algorithmen*. Teubner Verlag, 2002.