UNIVERSITY OF DORTMUND

REIHE COMPUTATIONAL INTELLIGENCE

COLLABORATIVE RESEARCH CENTER 531

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

Randomized Search Heuristics as an Alternative to Exact Optimization

Ingo Wegener

No. CI-168/04

Technical Report ISSN 1433-3325 February 2004

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

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

Randomized Search Heuristics as an Alternative to Exact Optimization

Ingo Wegener*

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

Abstract. There are many alternatives to handle discrete optimization problems in applications. Problem-specific algorithms vs. heuristics, exact optimization vs. approximation vs. heuristic solutions, guaranteed run time vs. expected run time vs. experimental run time analysis. Here, a framework for a theory of randomized search heuristics is presented. After a brief history of discrete optimization, scenarios are discussed where randomized search heuristics are appropriate. Different randomized search heuristics are presented and it is argued why the expected optimization time of heuristics should be analyzed. Afterwards, the tools for such an analysis are described and applied to some well-known discrete optimization problems. Finally, a complexity theory of so-called black-box problems is presented and it is shown how the limits of randomized search heuristics can be proved without assumptions like NP $\neq$ P. This survey article does not contain proofs but hints where to find them.

1 Introduction

For our purposes it makes no sense to look for the ancient roots of algorithmic techniques. The algorithmic solution of large-scale problems was not possible before computers were used to run the algorithm. Some of the early algorithms of this period like Dantzig's simplex algorithm for linear programming from 1947 and Ford and Fulkerson's network flow algorithm from 1956 based on improvements along augmenting paths were quite successful. In the fifties and early sixties of the last century one was satisfied when the algorithm was running efficiently – in most experiments. Nowadays, we know that the simplex algorithm has an exponential worst-case run

* Supported in part by the Deutsche Forschungsgemeinschaft (DFG) as part of the Collaborative Research Center "Computational Intelligence" (SFB 531) and by the German Israeli Foundation (GIF) in the project "Robustness Aspects of Algorithms".

time and that small randomized pertubations in the input turn all inputs into easy ones with respect to the input distribution based on the pertubation (Spielman and Teng (2001)). The first network flow algorithm was only pseudo-polynomial. Now, we know of many polynomial network flow algorithms. However, the original simplex algorithm still finds many applications and the best network flow algorithms use ideas of Ford and Fulkerson.

Later, algorithm design was accompanied by a run time analysis. Many algorithms have good worst-case run times. Randomized algorithms are often simpler and faster than deterministic ones. The NP-completeness theory proves that there are many important problems which are intractable – with respect to the worst case run time (see Garey and Johnson (1979)). At the same time, approximation algorithms guaranteeing good solutions in short time were presented and analyzed (see Hochbaum (1997)). All the experts believe that NP $\neq$ P ant it makes sense to argue under this hypothesis. In the seventies many people believed that we cannot solve NP-hard optimization problems for large instances. As computers got faster, applications were showing that this belief is wrong. There are problem-specific algorithms solving the "typical cases" of difficult problems sufficiently fast. In most cases, it was impossible to describe the "easy instances" for such algorithms. Nowadays, we have a toolbox of algorithmic techniques to design such problem-specific algorithms. Among them are greedy algorithms, dynamic programming, branch-and-bound, relaxation techniques, and many more. Randomized algorithms are now very common and even derandomization techniques exist. The community working on the design and analysis of algorithms has broadened its scope. The experimental tuning of algorithms named algorithm engineering is part of this discipline. However, the community is still focused on problem-specific algorithms.

Randomized search heuristics (RSH) in their pure form are algorithmic techniques to attack optimization problems with one general strategy. They are heuristics in a strong sense. Typically, they do not guarantee any (non-trivial) time bound and they do not guarantee any quality of the result. Randomized local search (RLS) is a member of this family of algorithms if it does not use problem-specific neighborhoods. Simulated annealing (SA) and all kinds of evolution-

ary algorithms (EA) (including evolution strategies (ES) and genetic algorithms (GA)) are typical RSHs. The idea was to simulate some successful strategy of engineering (annealing) or nature (evolution). The history of these RSHs dates back to the fifties and sixties of the last century. Despite some early successes they did not find so many applications before the late eighties since computers were not fast enough. Nowadays, people in applications like these heuristics and people in the algorithm community have ignored them for a long time. Indeed, these RSHs were considered as the black sheeps in the algorithm community. There were several reasons for this. RSHs were often claimed to be the best algorithm for a problem which was wrong because of very clever problem-specific algorithms. Arguments supporting in particular EAs were based on biology and not on a run time analysis. So there were many "soft" arguments and the two communities were using different languages. The approach to analyze RSHs like all other algorithms is a bridge between the two communities and both of them have started to use this bridge.

In this overview, we try to describe this approach. First, it is important to discuss the scenarios where we recommend RSHs in their pure form. This is necessary to avoid many common misunderstandings, see Section 2. In Section 3, we present the general form of a black-box algorithm and discuss some types of SA and EA. As already described, we propose the run time analysis of RSHs. In Section 4, we discuss what we expect from such an approach and compare this approach with the other theoretical results on RSHs. In Section 5, we describe the tool-box for the analysis of RSHs and, in Section 6, we present some results in order to show that we can realize our goals – at least in some cases. In the classical scenario, design and analysis of algorithms is accompanied by complexity theory showing the limits of the algorithmic efforts. In Section 7, we argue that an information-restricted scenario, namely the scenario of black-box algorithms, can be used to prove some limits of RSHs. Since the available information is limited, these limits can be proved without complexity theoretical hypotheses like NP $\neq$ P. We finish with some conclusions.

2 Scenarios for the Application of Randomized Search Heuristics

The first fact we have to accept is that no algorithmic tool will be the best for all problems. We should not expect that some RSH will beat quicksort or Dijkstra's shortest path algorithm. Besides these obvious examples there are many problems where the algorithm community has developed very efficient algorithms. The following statement seems to be true in almost all situations. If a problem has a structure which is understood by the algorithm community and if this community has made some effort in designing algorithms for the problem, then an RSH in its pure form will be worse with respect to its run time and/or the quality of its results. Many people working on RSHs do not agree with this statement. This is due to a misunderstanding. The statement argues about RSHs in their pure form. If they have problem-specific components, they can be competitive but then they are already hybrid algorithms combining ideas from RSHs with problem-specific ideas. If such hybrid algorithms are possible, one should consider them as alternative. For hybrid algorithms, it is difficult to estimate whether the general idea of an RSH or a problem-specific module is the essential part.

The question is whether there are scenarios where one should apply RSHs in their pure form. We think of two such scenarios.

In many projects, optimization problems are subproblems and a good algorithm has to be presented in short time. If no expert has considered the problem before and if there are no experts in algorithm design in the project and if there is not enough time and/or money to ask a group of experts, then an RSH can be a good choice. In real applications, the small amount of knowledge about the problem, should be used. RSHs have turned out to be a robust tool which for many problems leads to satisfactory results. In this scenario, we expect that better algorithms exist but are not available. In many applications, there is no knowledge about the problem. As an example, think of the engineering problem to "optimize some machinery". The engineer can choose the values of certain free parameters and each choice "realizes a machine". There exists a function $f\colon M \to \mathbb{R}$ where $f(x)$ is the quality of the machine $x \in M$. Because of the complexity of the application, the function f is not known and $f(x)$

has to be estimated by an experiment or its simulation with a computer. Here, a problem-specific approach is impossible and RSHs are a good choice.

Summarizing, there are scenarios where RSHs are applied in (almost) pure form. Whenever possible, one should apply all available problem-specific knowledge.

3 How Randomized Search Heuristics Work

We consider the maximization of some function $f\colon S \to \mathbb{R}$ where S is a finite (discrete) search space. The main difference between an RSH and a problem-specific algorithm is the following. The RSH does not assume knowledge about f. It may sample some information about f by choosing search points $x \in S$ and evaluating $f(x)$. Therefore, we may think of a black box containing f and the algorithm not knowing f can use the black box in order to get the answer $f(x)$ to the query x. Hence, the general form of a black-box algorithm can be described as follows.

The general black-box algorithm (BBA)

The initial information I_0 is the empty sequence. In the tth step, based on $I_{t-1} = ((x_1, f(x_1)), \ldots, (x_{t-1}, f(x_{t-1})))$, the algorithm computes a probability distribution p_t on S, chooses x_t according to p_t, and uses the black box to obtain $f(x_t)$. If a stopping criterion is fulfilled, the search is stopped and a search point x_i with maximal f-value is presented as result.

All known RSHs fit into this framework. Most often, they are even more specialized. The information I_t is considered as (unordered) multiset and not all the information is kept in the storage. There is a bound $s(n)$ on the number of pairs $(x, f(x))$ which can be stored and in order to store the new pair $(x_t, f(x_t))$ one has to throw away another one. This is called an $s(n)$-BBA. RLS and SA work with $s(n) = 1$ and, for EAs, $s(n)$ usually is not very large.

If $s(n) = 1$, we need a search operator which produces a new search point x' depending on the current search point x and its f-value. Then we need a selection procedure which decides whether x'

replaces x. RLS and SA restrict the search to a small neighborhood of x. SA allows that worse search points replace better ones. The special variant called Metropolis algorithms accepts x' with a probability of $\exp(-(f(x) - f(x'))/T)$ for some fixed parameter $T \geq 0$, called the temperature, if $f(x') < f(x)$, and it accepts x', if $f(x') \geq f(x)$. SA varies T using a so-called cooling schedule. This schedule depends on t and we have to store the point of time besides the search point. EAs prefer more global search operators. A so-called mutation step on $S = \{0,1\}^n$ flips each bit independently with a probability p_n, its typical calue is $1/n$. Because of this global search operator EAs can decide to accept always the best search points (the so-called plus strategy). The $s(n)$ search points kept in the storage are called population and it is typical to produce more than one search point before selecting those which survive. These phases are called generations. There are many selection procedures that we do not have to discuss in detail. If the population size is larger than 1, we also need a selection procedure to select those search points (called individuals) that are the objects of mutation. Finally, larger populations allow search operators depending on more than one individual. Search operators working on one individual are called mutation and search operators working on at least two (in most cases exactly two) individuals are called crossover or recombination. This leads to many free parameters of an EA and all the parameters can be changed over time (dynamic EAs for a fixed schedule and adaptive EAs otherwise). In the most general form of self-adaptive EAs, the free parameters are added to the search points (leading to a larger search space) and are changed by mutation and recombination. The class of pure EAs is already large and there is the freedom of adding problem-specific modules.

4 Arguments for the Analysis of the Expected Optimization Time

Do we need a theory of RSHs? They are heuristics – so let us try them! There is a common belief that a theory would improve the understanding of RSHs and, therefore, the choice of the parameters and the application of RSHs. Theory is contained in all the monographs

on RSHs. The question is what type of theory is the right choice. The following discussion presents the different types of theoretical approaches (with an emphasis on EA theory) and some personal criticism in order to motivate our approach.

In general, we investigate a class of functions or, equivalently, a problem consisting of a class of instances. Many RSHs are quite complex but even simple variants like the Metropolis algorithm or a mutation-based EA with population size 1 have a complex behavior for most problems. People in classical algorithm theory design an algorithm with the aim that the algorithm is efficient and the aim that they are able to prove this. Here, the algorithm realizes a general search strategy which does not support its analysis. Thus, the analysis of RSHs has to overcome different obstacles than classical algorithm analysis.

This has led many researchers to simplify the situation by analyzing a so-called model of the algorithm. Afterwards, experiments "verify" the quality of the model. This procedure is common in physics. There, a model of "nature" is necessary since "reality" cannot be described accurately. An algorithm and, therefore, an RSH is nothing from nature but an abstract device made by humans. It is specified exactly and, ignoring computer failures, we can analyze the "true algorithm" and a model is not necessary. An analysis of the algorithm can give precise results (at least in principle) and this makes a verification superfluous. Moreover, the dimension of our "world", namely the search space, is not limited. Experiments can tell us only something about those problem dimensions n that we can handle now. A theoretical analysis can give theorems for all n.

In order to handle large populations, people have studied the model of infinite populations (described by probability vectors) which are easier to handle. There are only few papers investigating the error of this model with respect to a restricted population size and time (Rabani, Rabinovich, and Sinclair (1998) and Ollivier (2003)).

An RSH is a stochastic process and one can study the dynamics of this process. Such results are often on a high technical level but the results are hard to interpret if we are interested in the process as an optimization algorithm.

There are also many papers analyzing quite precisely the result of one step of the algorithm. How much do we improve the best known

f-value? How close (in distance in the search space) do we approach an optimal search point? The complete knowledge of the one-step behavior is (in principle) enough to derive the complete knowledge of the global behavior of the process. However, this derivation is the hard step. As known from classical algorithm analysis, we should be happy to understand the asymptotic behavior of algorithms and, for this, it is not necessary to have a precise understanding of the one-step behavior. Well-known concepts as schema theory or building block hypothesis have led people to wrong conjectures about algorithms.

Finally, there are many papers proving the convergence of RSHs to the optimum. Without an estimate of the speed of convergence, these results are of limited interest – at least in finite search spaces. Convergence is guaranteed easily if we store the best search point ever seen. Then an enumeration of the search space suffices as well as a mutation operator giving a positive probability to reach every search point.

Our approach is to analyze an RSH as any other randomized algorithm. The only difference is that an RSH cannot know whether it has found an optimal search point. In most applications, the stopping criterion is not a big problem. Therefore, we investigate an RSH as an infinite stochastic process (without stopping criterion) and analyze random variables of interest.

A typical example is the random variable describing the first point of time where an optimal search point is passed to the black box. Its expected value is called the expected run time or optimization time of the algorithm. In order to analyze restart techniques, it is also interesting to analyze the success probability, namely the probability of the event that the run time is bounded by some given time bound. We have to be careful since we only count the number of search points and, therefore, the number of f-evaluations, and not the effort to compute the search points. For most RSHs, this is a good measure. If one applies time-consuming procedures to compute search points, one has to analyze also these resources.

5 Tools for the Analysis of Randomized Search Heuristics

As in classical algorithm analysis, one has to develop a good intuition how the given RSH works for the considered problem. Then one can describe "typical runs" of the RSH. They are approaching the optimum by realizing sequentially certain subgoals. One tries to estimate the run time for reaching subgoal i assuming that subgoal $i-1$ has been realized. This can lead to a time bound $t_i(n)$ for Phase i. The aim is to estimate the error probability $\varepsilon_i(n)$ that Phase i is not finished within $t_i(n)$ steps. The sum of all $\varepsilon_i(n)$ is an upper bound on the probability that the run is not typical, e. g., that it takes longer than the sum of all $t_i(n)$. Obviously, the last subgoal has to be the goal to find an optimal search point. Often it is possible to obtain exponentially small $\varepsilon_i(n)$ implying that even polynomially many subgoals lead to an exponentially small error probability. (See Jansen and Wegener (2001) for a typical application of this method.)

A related proof technique is to define an f-based partition $(A_1, \ldots, A_m)$ of the search space, i. e. $x \in A_i$, $y \in A_j$, and $j > i$ imply $f(x) < f(y)$ in the case of maximization and A_m contains exactly the optimal search points. Then one has to estimate the expected time $t_i(n)$ of finding a search point in some A_j, $j > i$, given that a search point in A_i is known. Here we have to assume that we never forget the best of the search points ever seen.

As already mentioned, the stochastic process P describing an RSH can be quite complex. A simple and powerful idea (which typically is difficult to realize) is to construct a stochastic process P' which is provably slower than P but easier to analyze than P. In order to obtain good bounds, P' should not be "much slower" than P. The notion slower can be defined as follows. A random variable X is called not larger than X' if $\text{Prob}(X \leq b) \geq \text{Prob}(X' \leq b)$ for all b. The process P is not faster than P' if the random variable T describing a good event for P is not larger than the corresponding random variable T' for P'. (See Wegener and Witt (2003) for a typical application of this method.)

Finally, it is not always the right choice to measure the progress of the RSH with respect to the given function f (often called fitness function) but with respect to some pseudo-fitness function g known

in classical algorithm design as potential function. Note that the algorithm still works on f and g is only used in the analysis of the algorithm. (See Droste, Jansen, and Wegener (2002) for a typical application of this method.)

During all the calculations, one needs technical tools from probability theory, among them all the famous tail inequalities (due to Markoff, Tschebyscheff, Chernoff, or Hoeffding). Other tools are the coupon collector's theorem or results on the gambler's ruin problem.

Finally, we mention a special technique called delay sequence arguments. This technique has been introduced by Ranade (1991) for routing problems and has been applied only once for RSHs (Dietzfelbinger, Naudts, van Hoyweghen, and Wegener (2002)). The idea is the following. If we expect that a stochastic process is finished in time $t(n)$ with large probability, we look for an event which has to happen if the stochastic process is delayed, i. e., if it runs for more than $t(n)$ steps. Then we estimate the probability of the chosen event. Despite of the simplicity of the general idea, this method has been proven to be quite powerful.

Most RSHs apply in each step the same techniques of search and selection. This implies that the same experiment is applied quite often. The experiments are influenced by the available information I_t but in many aspects these experiments are independent or almost independent. Because of the large number of experiments during a search or a subphase of a search, many interesting random variables are highly concentrated around their expected values. This simplifies the analysis and can lead to sharp estimates.

6 The Analysis of Randomized Search Heuristics on Selected Problems

The analysis of the expected optimization time of RSHs is far behind the analysis of classical algorithms. One reason is that the number of researchers in this area is still much smaller and that they have started to work on this subject much later. It will take several years until the research on the analysis of RSHs can reach the status of classical algorithm analysis. Nevertheless, it is not possible to give here an overview of all the results obtained so far. We discuss some

results to motivate the reader to take a closer look to the original papers and this section has a focus on results obtained in our research group.

In order to build a bridge between the RSH community gathering in conferences like GECCO, PPSN, CEC, and FOGA and the classical algorithm community gathering in conferences like STOC, FOCS, ICALP, STACS, SODA, and ESA, we have tried to work on topics of the following kind:

- solving open problems on RSHs discussed in the RSH community,
- analyzing RSHs on functions which are considered as typical in the RSH community,
- analyzing RSHs on classes of functions described by structural properties, and
- analyzing RSHs on well-known problems of combinatorial optimization.

Concerning EAs, the American school (Holland (1975), Goldberg (1989), Fogel (1995)) proposed GAs where crossover is the most important search operator and the European school (Rechenberg (1994), Schwefel (1995)) proposed ESs merely based on mutation. Concerning combinatorial optimization, several results have shown that mutation is essential. What about crossover? Even after 30 years of debates there was no example known where all mutation-based EAs have a super-polynomial optimization time while a generic GA works in expected polynomial time. Jansen and Wegener (2002), the conference version appeared 1999, were the first to present such a result. The considered function is quite simple but the analysis is complicated. Crossover can be useful only between two quite different search points. Selection prefers good search points which can decrease the diversity of the population. The GA which has been analyzed does not use special modules to preserve the diversity and one has to prove that the population nevertheless is diverse enough. This paper proves a trade-off of super-polynomial versus polynomial. This has been improved in a later paper (Jansen and Wegener (2001)) to exponential versus polynomial. This is due to the definition of an artificial function which supports the analysis of the GA.

In this survey article, it does not make sense to introduce functions that have been discussed intensively in the RSH community.

Each pseudo-boolean function $f\colon \{0,1\}^n \to \mathbb{R}$ can be described uniquely as a polynomial

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

Its degree is the maximal $|A|$ where $w_A \neq 0$. We can consider functions with bounded degree. Degree-1 functions, also called linear functions, were investigated in many papers. In particular, the expected run time of the (1+1) EA (population size 1, mutations flipping the bits independently with probability $1/n$, replacing x by x' if $f(x') \geq f(x)$ in the case of maximization) was of interest. Droste, Jansen, and Wegener (2002) have proved that the expected run time is always $O(n \log n)$ and $\Omega(n \log n)$ if $\Omega(n^\varepsilon), \varepsilon > 0$, weights are non-zero. The proof applied the techniques of potential functions and the investigation of slower stochastic processes.

The maximization of degree-2 or quadratic polynomials is NP-hard. Wegener and Witt (2002) have investigated RSHs on certain quadratic functions. They have proved that restart techniques can decrease the expected optimization time from exponential to polynomial and they have presented a quadratic polynomial where all mutation-based EAs are slow. Moreover, they have investigated the case of monotone quadratic polynomials. A polynomial is called monotone if it can be represented with non-negative weights after replacing some x_i by $1 - x_i$. This allows that the polynomial is monotone increasing with respect to some variables and monotone decreasing with respect to the other variables. Wegener and Witt (2003) have generalized this approach to degree-d polynomials. For RLS and the (1+1) EA with a small probability of flipping bits they have proved an upper bound of $O(2^d \cdot (n/d) \cdot \log(n/d))$ which is optimal since there is a corresponding lower bound for the sum of all $x_{id+1} \cdots x_{(i+1)d}$, $0 \leq i \leq \lfloor n/d \rfloor - 1$. The proof of the upper bound is mainly based on the investigation of slower stochastic processes.

Finally, there is some recent progress in the analysis of EAs on combinatorial optimization problems. Many earlier results consider RLS and some results consider SA. Scharnow, Tinnefeld, and Wegener (2002) have shown that the complexity of the problem depends on the choice of the corresponding fitness function. The sorting problem is the best investigated computer science problem. It can be

considered as the problem of minimizing the unsortedness. There are many measures of unsortedness known from the design of adaptive sorting algorithms. E. g., we count the number of inversions, i. e., of pairs of objects in incorrect order or we count the number of runs or, equivalently, the number of adjacent pairs of objects in incorrect order. A variant of the (1+1) EA working on the search space of all permutations can sort in expected time $O(n^2 \log n)$ using the number of inversions (and several other measures of unsortedness) but it needs exponential time using the number of runs. The same effect has been shown in the same paper for the single-source-shortest-paths problem. The search points are directed trees rooted at the source and, therefore, containing paths to all other vertices. If the fitness is described by the sum of the lengths of the paths from the source to all other vertices, each black-box search heuristic needs exponential time while time $O(n^3)$ is enough on the average if the fitness is described by the vector containing all the path lengths.

The (1+1) EA is surprisingly efficient for the minimum spanning tree problem (Neumann and Wegener (2003)). For graphs on n vertices and m edges and edge weights bounded by $w_{\max}$, the expected run time is bounded by $O(m^2(\log n + \log w_{\max}))$.

The maximum matching problem is more difficult for RSHs. There are example graphs where SA (Sasaki and Hajek (1988)) and the (1+1) EA (Giel and Wegener (2003)) need expected exponential time. In the latter paper it is shown that the (1+1) EA is efficient on certain simple graphs. More important is the following result. The (1+1) EA is a PRAS (polynomial-time randomized approximation scheme), i. e., an approximation ratio of $1 + \varepsilon$ can be obtained in expected polynomial time where the degree of the polynomial depends on $1/\varepsilon$. Due to the examples mentioned above this is the best possible result. The true aim of search heuristics is to come close to the optimum, exact optimization is not necessary.

7 Black-Box Complexity – The Complexity Theoretical Background

The general BBA works in an information-restricted scenario. This allows the proof of lower bounds which hold for any type of BBA.

This theory based on Yao's minimax principle (Yao (1977)) has been developed and applied by Droste, Jansen, and Wegener (2003). A randomized BBA is nothing but a probability distribution on the set of deterministic BBAs. This is obvious although it can be difficult to describe this probability distribution explicitly if a randomized BBA is given. For a deterministic BBA storing the whole history it does not make sense to repeat a query asked earlier. For a finite search space, the number of such deterministic BBAs is finite. In many cases, also the set of problem instances of dimension n is finite.

In such a situation, Yao's minimax principle states that we obtain a lower bound on the expected optimization time of each randomized BBA by choosing a probability distribution on the set of problem instances and proving a lower bound on the average optimization time of each *deterministic* BBA where the average is taken with respect to the random problem instance. The key idea is to consider algorithm design as a zero-sum game between the algorithm designer and the adversary choosing the problem instance. The algorithm designer has to pay 1 Euro for each query. Then Yao's minimax principle is a corollary to the minimax theorem for two-person zero-sum games.

Applying this theory leads to several interesting results. It is not too hard to prove that randomized BBAs need $\Omega(2^d + n/\log n)$ queries on the class of monotone d-degree polynomials proving that RLS and the considered variant of (1+1) EA are close to optimal for these functions. Also, the exponential lower bound for the single-source-shortest-paths problem follows from this theory.

Many people have stated that EAs are particularly efficient on unimodal functions $f\colon \{0,1\}^n \to \mathbb{R}$. Such functions have a unique global optimum and no further local optimum with respect to the neighborhood of Hamming distance 1. We investigate unimodal functions $f\colon \{0,1\}^n \to \{0,\ldots,b\}$ with a bounded image set. The upper bound for EAs is the trivial bound $O(nb)$ which also can be realized by deterministic local search. This bound indeed is not far from optimal. In a different framework, Aldous (1983) has investigated one-to-one unimodal functions and has proved that their black-box complexity is $2^{n/2}$ up to polynomial factors. In applications, the case of limited b is more interesting. Droste, Jansen, and Wegener (2003) have proven a lower bound of $\Omega(b/\log^2 b)$ if $2n \leq b = 2^{o(n)}$.

Sometimes, upper bounds on the black-box complexity are surprisingly small. This happens if the algorithm uses much time to evaluate the information contained in I_t. This can lead to polynomially many queries for NP-equivalent problems. These algorithms use exponential time to evaluate the information. If we restrict this time to polynomial, lower bounds are based on hypotheses like NP≠P. It is an open problem to prove in such cases exponential lower bounds for $s(n)$-BBAs and small $s(n)$.

Conclusions

Applications in many areas have proved that RSHs are useful in many situations. A theory on RSHs in the style of the theory on problem-specific algorithms will improve the understanding of RSHs and will give hints how to choose the free parameters depending on the problem structure. Moreover, RSHs will be included in lectures on efficient algorithms only if such a theory is available. It has been shown that such a theory is possible and that first results of such a theory have been obtained.

References

1. Aldous, D. (1983). Minimization algorithms and random walk on the d-cube. The Annals of Probability 11, 403–413.
2. Dietzfelbinger, M., Naudts, B., van Hoyweghen, C., and Wegener, I. (2002). The analysis of a recombinative hill-climber on H-IFF. Accepted for publication in IEEE–Trans. on Evolutionary Computation.
3. Droste, S., Jansen, T., and Wegener, I. (2002). On the analysis of the (1+1) evolutionary algorithm. Theoretical Computer Science 276, 51–81.
4. Droste, S., Jansen, T., and Wegener, I. (2003). Upper and lower bounds for randomized search heuristics in black-box optimization. Accepted for publication in Theory of Computing Systems.
5. Fogel, D. B. (1995). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press, Piscataway, NJ.
6. Garey, M. R. and Johnson, D. B. (1979). *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W. H. Freeman.
7. Giel, O. and Wegener, I., (2003). Evolutionary algorithms and the maximum matching problem. Proc. of 20th Symp. on Theoretical Aspects of Computer Science (STACS), LNCS 2607, 415–426.
8. Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA.
9. Hochbaum, D. (ed.) (1997). *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, Boston.

10. Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* Univ. of Michigan, MI.
11. Jansen, T. and Wegener, I. (2001). Real royal road functions – where crossover is provably essential. Proc. of the Genetic and Evolutionary Computation Conference (GECCO '2001). Morgan Kaufmann, San Mateo, CA, 375–382.
12. Jansen, T. and Wegener, I. (2002). The analysis of evolutionary algorithms – a proof that crossover really can help. Algorithmica 34, 47–66.
13. Neumann, F. and Wegener, I. (2003). Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. Submitted for publication.
14. Ollivier, Y. (2003). Rate of convergence of crossover operators. Random Structures and Algorithms 23, 58–72.
15. Rabani, Y. Rabinovich, Y., and Sinclair, A. (1998). A computational view of population genetics. Random Structures and Algorithms 12, 314–330.
16. Ranade, A. G. (1991). How to emulate shared memory. Journal of Computer and System Sciences 42, 307–326.
17. Rechenberg, I. (1994). *Evolutionsstrategie '94.* Frommann-Holzboog, Stuttgart.
18. Sasaki, G. H. and Hajek, B. (1988). The time complexity of maximum matching by simulated annealing. Journal of the ACM 35, 387–403.
19. Scharnow, J., Tinnefeld, K., and Wegener, I. (2002). Fitness landscapes based on sorting and shortest paths problems. Proc. of 7th Conf. on Parallel Problem Solving from Nature (PPSN-VII), LNCS 2439, 54–63.
20. Schwefel, H.-P. (1995). *Evolution and Optimum Seeking.* Wiley, New York.
21. Spielman, D. A. and Teng, S.-H. (2001). Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. Proc. of 33rd ACM Symp. on Theory of Computing (STOC), 296–305.
22. Wegener, I. and Witt, C. (2002). On the analysis of a simple evolutionary algorithm on quadratic pseudo-boolean functions. Accepted for publication in Journal of Discrete Algorithms.
23. Wegener, I. and Witt, C. (2003). On the optimization of monotone polynomials by simple randomized search heuristics. Accepted for publication in Combinatorics, Probability and Computing.
24. Yao, A. C. (1977). Probabilistic computations: Towards a unified measure of complexity. Proc. of 17th IEEE Symp. on Foundations of Computer Science (FOCS), 222–227.