Theoretical Analysis of Rank-based Mutation - Combining Exploration and Exploitation

Pietro S. Oliveto, Per Kristian Lehre, Frank Neumann

Abstract— **Parameter setting is an important issue in the design of evolutionary algorithms. Recently, experimental work has pointed out that it is often not useful to work with a fixed mutation rate. Therefore it was proposed that the population be ranked according to fitness and the mutation rate of an individual should depend on its rank. The claim is that this allows the algorithm to explore new regions in the search space as well as progress quickly towards optimal solutions. Complementing the experimental investigations, we examine the proposed approach by presenting rigorous theoretical analyses which point out the differences of rank-based mutation compared to a standard approach using a fixed mutation rate. To this end we theoretically explain the behaviour of rank-based mutation on various fitness landscapes proposed in the experimental work and present new significant classes of functions where the use of rank-based mutation may be both beneficial or detrimental compared to fixed mutation strategies.**

I. Introduction

Determining the optimal parameters for an evolutionary algorithm is a challenging task that has been widely studied in the field of evolutionary computation [8]. There are many parameters in an evolutionary algorithm and many studies have focused on how parameters such as representation, population size or variation operator rates affect the algorithm's performance.

In this paper, we focus on the mutation rate used in an evolutionary algorithm. The optimal mutation rate is known only for very simple problems such as ONEMAX [2]. Often it is useful to work not only with one fixed mutation rate but to adapt it during the optimization process. This is usually done in continuous optimization where the mutation strength depends on the progress that the algorithm has achieved during the last iterations. On the other hand, in combinatorial optimization it is less common to adapt the mutation rate during the optimization process. In fact most computational complexity analyses of evolutionary algorithms for combinatorial optimization consider algorithms with fixed mutation rates (see [10] for a review of results).

Nevertheless, the use of different mutation rates with respect to the runtime behaviour of evolutionary algorithms has already been studied in literature. Jansen and Wegener [7] have examined the choice of the mutation probability in the (1+1) EA and proposed a dynamic (1+1) EA that uses different mutation probabilities at different time steps. In [4] the effect of bit-wise neutrality with respect to the mutation rates has been examined and it has been shown that it may be helpful to use different mutation rates for each gene in the genotype. Recently, in [13] an immune inspired mutation operator has been analysed for the ONEMAX function where the mutation rate of an individual is inversely proportional to its fitness.

Instead of focusing on algorithms that work with a single solution, we examine population-based algorithms where the different individuals have different mutation rates as recently proposed in [1]. The individuals in the current population are ranked with respect to their fitness and the mutation rate increases with the rank of an individual. The idea behind this is that good individuals should produce offspring that are close whereas bad individuals should explore regions of the search space that are very different. The use of this approach has been examined experimentally in [1]. In particular, based on their experiments, the authors claim that using rank-based mutation allows the algorithm to have a good balance between exploration and exploitation. In this paper, we want to show the impact that rank based mutation has on the optimization process in a rigorous manner.

After having defined the algorithms considered throughout the paper in Section II, we start our theoretical analysis in Section III by pointing out some general results of the mentioned approach. These relate it to the use of random search by proving a general upper bound on the expected optimization time on any pseudo-Boolean function. This bound will be proved to be tight further on in the paper (i.e. Section V).

After that we analyse the use of rank-based mutation on landscapes with different difficulties that have already been examined experimentally in [1]. Our analyses point out the different effects that the use of rank-based mutation has on simple unimodal functions (Section IV) as well as on difficult deceptive trap functions (Section V). In Section IV, through an analysis for the ONEMAX function, we show that the rank-based mutation strategy is effective in climbing up slopes. In Section V we give theoretical evidence of the better performance of rank-based mutation rates compared to fixed mutation rates for the trap functions considered in [1]. However, we also show that there exist classes of functions which are deceptive for rank-based mutation leading to exponential runtime while fixed-mutation rate algorithms are efficient with high probability. In any case, when rank-based mutation has a better performance, the runtime required by both strategies to solve the trap functions is exponential in

Per Kristian Lehre and Pietro S. Oliveto are with the Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), University of Birmingham, Edgbaston, Birmingham, B15 2TT, UK (email:{P.S.Oliveto,P.K.Lehre}@cs.bham.ac.uk). Frank Neumann is with the Department 1: Algorithms and Complexity, Max-Planck-Institut für Informatik, Saarbrücken, Germany (email: firstname.lastname@mpi-inf.mpg.de).

978-1-4244-2959-2/09/$25.00 © 2009 IEEE

the problem size.

To this end, in the last part of the paper (Section VI), we present a class of functions where we can point out that rank-based mutation significantly helps to speed up the optimization process. These functions include sub-problems with different difficulties (i. e. a deceptive and a unimodal part).

II. Algorithms

We study a simple (μ+1) EA using rank-based mutation. The algorithm produces in each iteration one offspring by choosing an individual of the current parent population uniformly at random. To this individual the mutation operator is applied which flips each bit with the probability given by its rank in the population. Afterwards an individual with the lowest fitness among the $\mu + 1$ individuals is deleted such that a new parent population of size μ is obtained.

Algorithm 1: (μ+1) EA_R

1) Let $t = 0$ and initialize P_0 with μ individuals chosen uniformly at random.
2) Repeat
   a) Rank the individuals $\{x_1 \dots x_\mu\}$ s.t. $f(x_i) \geq f(x_{i+1})$.
   b) Choose $x_i \in P_t$ uniformly at random.
   c) Create y by mutating each bit in x_i with probability p_i.
   d) If $f(y) \geq f(x_\mu)$ then $P_{t+1} = P_t \setminus \{x_\mu\} \cup \{y\}$; else $P_{t+1} = P_t$.

As in [1], the mutation rate of an individual at position i in the ranked population is assigned according to the following formula:

$$p_i = p_{min} + (p_{max} - p_{min}) \cdot (i-1)/(m-1)$$

where m is the number of different mutation rates used by the algorithm. In [1] the following parameters were used: $p_{min} = 0$, $p_{max} = 1$ and m was set as the size of the population plus one. Since an elitist selection strategy is used by the algorithm, there is no advantage in having a mutation rate of $p = 0$. In fact, not allowing the best individual to mutate would probably slow down the optimization process at least when the algorithm is hill-climbing. Given the elitist strategy, we believe that unless the global optimum has already been found it is always preferable to mutate the best individual even if with a very small mutation rate. So we set $p_{min} = 1/n$, $p_{max} = 1$ and $m = \mu$ where μ is the population size. This way the mutation rates are linearly distributed between $p_{min} = 1/n$ and $p_{max} = 1$.

We want to point out the different effects of using rank based mutation. The algorithm in [1] uses fitness-proportional selection. However, on one hand it has been proved in [6, 9] that simple evolutionary algorithms using fitness-proportional selection are not able to optimize even simple linear pseudo-Boolean functions such as OneMax in polynomial time. In fact in Section IV it will be proved that the (μ+1) EA_R algorithm is efficient for OneMax. On the other hand, since the goal of the paper is to understand the effects of the rank-based mutation rates, we feel that it is easier to understand how mutation is operating if good solutions generated by mutation are not removed from the population by the selection operator. In other words, the mutation operator may create a good solution but at the same time the selection operator may not consider it for the next generation. This may happen commonly with fitness-proportional selection as shown in [6, 9]. When this effect happens the mutation operator could be "blamed" for an action for which the selection operator is responsible, hence the wrong conclusions could be derived about the effects of rank-based mutation rates. These are the reasons for using an elitist-strategy in our algorithm.

If the mutation rate does not depend on the rank but is the same for all individuals, then the algorithm generalises to the (μ+1) EA. Here each bit is mutated with a fixed probability p that is independent of the rank of the individual. In the literature, this algorithm has been examined for the choice of $p = 1/n$ by Witt [12] for pseudo-Boolean functions and for combinatorial optimization problems having practical applications such as Vertex Cover in [11]. If only one mutation rate is used throughout the optimization process such a rate seems to be reasonable. In fact, also in practical applications, when only one mutation rate is used it is usually low. The (μ+1) EA is obtained from Algorithm 1 by replacing line c with the following one:

Algorithm 2: Mutation operator for the (μ+1) EA

c') Create y by mutating each bit in x_i with probability p.

We examine the algorithms with respect to their runtime behaviour on functions with different properties to point out the effects of using rank-based mutation. We will consider some functions used in [1] to explain theoretically the results obtained from the experiments. Furthermore we will analyse other functions of interest and generalise our results to greater function classes. The measure of interest is the number of fitness function evaluations until the algorithm has produced an optimal search point for the first time. Since randomised algorithms are of stochastic nature, this number varies from run to run. We are interested in the expectation of the random variable representing the number of fitness evaluations. We call this expectation the *expected optimization time* of the algorithm on the examined function. Sometimes, the expected optimization time is not a sufficiently accurate measure to understand the performance of the algorithm for a given function. In fact, it may happen that the expected optimization time is exponential but at the same time the probability that in each run the algorithm finds the optimum be high, for example a constant. In those cases we will also consider the *success probability* of the algorithm, which is defined as the probability that the optimization time is within a given time bound.

III. General Computational Complexity

It is well known that the expected time until the (1+1) EA finds the global optimum of any fitness function is at most n^n steps [3]. Droste et al. have also proved that the bound

is tight. A general result will be derived here for the (μ+1) EA$_R$. The following theorem gives an upper bound for the expected runtime of the (μ+1) EA$_R$ for any function. It shows that the algorithm is on any function only by a constant factor slower than random search whose expected optimization time on any function is 2^n. In Section V it will be shown that there exist functions for which the bound is tight up to a constant factor. This means that the (μ+1) EA$_R$ algorithm performs better than the (1+1) EA in the worst case. However the runtime is exponential in the function size.

Theorem 1: Let $\mu > 2$ and $\mu = poly(n)$. The expected optimization time of the (μ+1) EA$_R$ algorithm for an arbitrary fitness function is at most $O(2^n)$.

Proof: The proof will follow the line of thought used by Droste et al. in [3] for the (1+1)-EA.

An individual of rank i flips each bit with probability:

$$p_i = \frac{1}{n} + \left(1 - \frac{1}{n}\right) \cdot \frac{i-1}{\mu - 1} = \frac{i-1}{\mu - 1} + \frac{1}{n}\left(1 - \frac{i-1}{\mu - 1}\right)$$

We consider all individuals x_i of the population with $\mu/3+1 \leq i \leq (1/2)\mu$.

Using $\mu/3 + 1 \leq i \leq \mu/2$ we get

$$\frac{1}{3} \leq p_i \leq \frac{1}{2} + o(1) \leq \frac{2}{3}$$

Let x^* be a global optimum of the function to be optimized and $H(x_i, x^*) < n$ be the Hamming distance between the bit-string representing x_i and that representing x^*. Hence, the probability that each individual x_i, with $\mu/3 + 1 \leq i \leq \mu/2$ is turned into the global optimum in one mutation step is

$$\left(p_i\right)^{H(x_i,x^*)} \cdot \left(1 - p_i\right)^{n - H(x_i,x^*)}$$

$$\geq \left(1/3\right)^{H(x_i,x^*)} \cdot \left(1 - 2/3\right)^{n - H(x_i,x^*)} \geq \left(1/3\right)^n = 3^{-n}$$

Since, the probability bound holds whatever the current bit-string representing the x_i individuals is, 3^n expected mutation steps of these x_i individuals are required for the optimum to be found. The probability that an individual x_i with $\mu/3 + 1 \leq i \leq \mu/2$ is chosen for mutation in each generation is

$$\frac{\mu/2 - \mu/3 - 1}{\mu} = \frac{\mu/6 - 1}{\mu} \geq \frac{\mu/7}{\mu} = \frac{1}{7}$$

giving an expected time of 7 generations for this event to happen. Multiplying, the expected time for the optimum to be found is at most $7 \cdot 3^n = O(2^n)$. ∎

In Section VI functions will be introduced where the (μ+1) EA$_R$ algorithm performs better than the (1+1) EA and the difference in runtime is a more practical polynomial versus super-polynomial.

IV. ONEMAX

In this section we will show that the (μ+1) EA$_R$ algorithm is efficient for the ONEMAX function by proving a runtime of $O(\mu n \log n)$.

Theorem 2: If the population size is bounded by $\mu = poly(n)$, then the expected optimization time of the (μ+1) EA$_R$ on the ONEMAX function is $O(\mu n \log n)$.

Proof: To prove the upper bound, note that the first individual in the ranked population (i.e. x_1) flips each bit with probability $1/n$. We can therefore follow the ideas of the proof of the (1+1) EA for the ONEMAX function used in [3]. The best individual in the ranked population gets selected for mutation in each generation with probability $1/\mu$. This implies it is expected to be chosen once in μ generations. Since the fittest individual x_1 requires $O(n \log n)$ steps to reach the optimum (i.e. [3]), we get an upper bound of $O(\mu n \log n)$ for the optimum to be found. ∎

In [12], Witt proves that the expected time for the (μ+1) EA to optimize ONEMAX is $O(\mu n + n \log n)$. The (μ+1) EA obtains a short runtime because at each fitness level L (i.e. there are L ones in the best individual of the population), many copies of the best individual are obtained (i.e. the whole population or at least $n/(n-L)$) in time $O(\mu \log(n/(n-L)))$. Then any of these individuals may reach the next fitness level, rather than only the best. It could be that the (μ+1) EA$_R$ algorithm cannot always take advantage of these multiple copies to quickly climb up the ONEMAX function because the individuals that flip each bit with high probability end up turning many one-bits into zero-bits when approaching the optimum. Hence, by applying different mutation rates according to the rank of the individuals, the algorithm may climb up slopes more slowly than a population of individuals that flip each bit with probability $1/n$. In any case the process needs to be analysed more carefully to understand whether the given bound is tight or not. We leave a theorem about the lower bound on ONEMAX as an interesting open question for future work.

V. DECEPTIVE FUNCTIONS

In this section we consider the performance of the (μ+1) EA$_R$ algorithm on deceptive functions. Trap functions have been considered several times in the analysis of EAs to show how this class of algorithms may be attracted by a local optima which leads the population far away from the global optimum. As a consequence the expected runtime of the algorithms is exponential.

First we will address a question that appears from the analysis of the (μ+1) EA$_R$ for ONEMAX presented in the previous section. Although, the (μ+1) EA requires $O(\mu n + n \log n)$ expected time to optimise the ONEMAX function, a $O(\mu n \log n)$ bound of has been proved in Theorem 2 for the (μ+1) EA$_R$. The best individual in the ranked population flips each bit with probability $1/n$ and the expected time for it to be selected for mutation is μ. So, it may be assumed that the upper bounds obtained in the analysis of the (1+1) EA could be extended to the analysis of the (μ+1) EA$_R$ by multiplying the upper bounds of the former algorithm by μ to obtain an upper bound on the runtime of the latter algorithm. To show that this is not the case we consider a function that we call LEADINGTRAPJUMP. Theorem 3 proves that the (μ+1) EA is efficient for this class of functions with overwhelming probability, while the expected runtime of the (μ+1) EA$_R$ is exponential in the problem size. Hence, not only is the upper bound of the (1+1) EA not generalisable to

the (μ+1) EA$_R$, but a class of functions is presented where the former algorithm (and the (μ+1) EA) is efficient while the latter is not.

After the analysis of the LEADINGTRAPJUMP function we will consider the trap function used in [1] which we choose to call TRAP$_1$. It will be proved in Theorem 4 that the (μ+1) EA$_R$ is efficient for the function. However, this only occurs because the global optimum is located at a Hamming distance of n from the local optimum. As it will be shown in the proof of Theorem 4, the position of the local optimum gives the (μ+1) EA$_R$ a rather unfair advantage over the (μ+1) EA. To this end, we consider a more generic trap function which we call TRAP$_2$. The only difference between the two trap functions is that we place the global optimum in a generic point having lower Hamming distance from the local optimum. Theorem 5 shows that the expected runtime of the (μ+1) EA$_R$ on the TRAP$_2$ function is $\Theta(2^n)$ which is exponential in the problem size. This means that, although its expected runtime is better than that of the(μ+1) EA (i.e. $\Omega(n^n)$) its performance is no better than that of Random Search. Theorem 5 also shows that the generic upper bound given in Theorem 1 of Section III (i.e. $O(2^n)$), which holds for every pseudo-Boolean function is tight. Hence, the expected runtime of the (μ+1) EA$_R$ algorithm on a generic pseudo-Boolean function is $\Theta(2^n)$.

Now we present the LEADINGTRAPJUMP function class to tackle the first goal proposed in this section. The LEADINGTRAPJUMP is a class of functions designed to show that whatever the population size of the (μ+1) EA$_R$ may be (as long as polynomial in the problem size), there exists a class of functions where the (μ+1) EA is efficient while the (μ+1) EA$_R$ is not. This is obtained by considering that if the population size of the (μ+1) EA$_R$ is $\mu = poly(n) = n^k$ with k a constant, then as proved in Theorem 3 its expected runtime is exponential for the following function:

$$\text{LEADINGTRAPJUMP}(x) = \begin{cases} 0 & \text{if } x = 1^{(9/10)n}* \\ \text{LO}(x_i | i > 2k+1) + 2k + 1 & \text{if } x = 0^{2k+1}1^{(9/10)n-2k-1}* \\ n-1 & \text{if } x = 0^{n/10}* \\ \text{LO}(x) & \text{otherwise.} \end{cases}$$

The LEADINGTRAPJUMP function consists of a leading ones path incrementing the fitness by one for each leading one until $(9/10)n - 1$ leading ones are reached. Then at least $2k$ leading zeroes need to be created to increment the fitness. Once these leading zeroes have been obtained by an individual, it may insert the last $n/10$ leading ones to reach the optimum. However, there is a trap having $n/10$ leading zeroes. The only better point than this one is the global optimum.

Theorem 3: Let $2 \leq \mu \leq n^k$ and k a constant. With constant probability the (μ+1) EA$_R$ optimises the LTJ function in time $2^{\Omega(n)}$. With probability $1 - 2^{-\Omega(n)}$ the (μ+1) EA optimises the LTJ in time $O(\mu n^{2k+2})$.

Proof: The probability that both algorithms are initialised with strings having the first $2k + 1$ bits set to zero and the next $(9/10)n - 2k - 2$ set to one is $2^{-\Omega(n)}$ which is exponentially small. The same asymptotic probability holds for both algorithms being initialised with the first $n/10$ bits set to zero. The rest of the proof of the statement regarding the (μ+1) EA follows.

We consider the following three phases:

1) The phase lasts until a solution with $(9/10)n - 1$ leading ones has been found by at least one individual for the first time;
2) Starting with at least one individual with $(9/10)n - 1$ leading ones, the phase lasts until a solution with $2k + 1$ leading zeroes has been found by at least one individual;
3) The phase lasts until the global optimum of the LEADINGTRAPJUMP function has been found.

Now we calculate the expected runtimes for each of the phases conditional to the event that the trap point is not found in the mean time. Then we will calculate the probability of the event that the trap is found first.

The expected time for the (μ+1) EA to find the point with $(9/10)n - 1$ leading ones (i.e. the end of the first phase), if the trap is not reached first is $O(\mu n^2)$. This is because at each time step the probability the individual with most leading ones is selected for mutation is $1/\mu$ and it creates the next leading one with probability $1/n$ and does not flip any other bit with probability $(1-1/n)^{n-1} \geq 1/e$, giving an expected time of at most $e\mu n$ for each improvement. Since at most $(9/10)n - 1$ leading ones need to be created the expected time is less than $t = e(10/9)\mu n^2 = O(\mu n^2)$. In fact with slightly more sophisticated arguments an upper bound of $O(\mu n \log n + n^2)$ can be proved for the leading ones part [12], but is not necessary here.

Following arguments in Droste et al. [3], there exists a constant $c > 0$ such that the probability that $9n/10$ leading ones have not been obtained within $c\mu n^2$ iterations is $e^{-\Omega(n)}$.

The expected time to conclude the second phase is $O(\mu n^{2k+1})$, because $1/\mu$ is the probability the individual with $(9/10)\mu - 1$ leading ones is selected and $1/(en^{2k+1})$ is a lower bound on the probability that the first $2k+1$ bits are mutated into zeroes. This gives an expected time of $e\mu n^{2k+1}$. The probability that this does not happen in time $e\mu n^{2k+2}$ is

$$\left(1 - \mu n^{-(2k+1)}\right)^{\mu n^{2k+2}} \leq \left(\frac{1}{e}\right)^n,$$

meaning that phase 2 is concluded in time $O(\mu n^{2k+2})$ with probability $1 - e^{-\Omega(n)}$ if a trap point is not found first.

Once the $2k + 1$ leading zeroes have been found, the last $n/10 + 1$ leading ones may be added, phase 3 concluded and the optimum found. Just like for phase 1 this happens in time $O(\mu n^2)$ with probability $1 - e^{-\Omega(n)}$ if a trap point is not found first. Summing up we get a total runtime of $O(\mu n^{2k+2})$ with probability at least $1 - e^{-\Omega(n)}$ to reach the optimum conditional to not finding the trap first.

Now we calculate the probability of finding the trap before $O(\mu n^{2k+2})$ steps. As discussed at the beginning of the proof, a trap point is not created during initialisation with

probability $1-2^{-\Omega(n)}$. We consider any individual with x leading ones and the remaining $n/10-x$ bits which are uniformly distributed. The probability that a trap point is created in a step is less than

$$\big(1/n\big)^{x}\cdot\big(1/2\big)^{n/10-x}\leq\big(1/2\big)^{n/10}=2^{-\Omega(n)}$$

Hence the probability is highest during initialisation. The higher the number of leading ones the lower is the probability that a trap point is created. The above discussion holds until the end of phase 2.

For the individuals with $2k+1$ leading zeroes (i.e. phase 2 has ended) the probability that they reach a trap point is less than

$$\big(1/n\big)^{n/10-(2k+1)}\leq\big(1/n\big)^{n/20}\leq n^{-\Omega(n)}$$

because at least $n/10-(2k+1)$ one-bits have to flip into zero-bits. Summing up in each step the probability of reaching a trap point is less than $2^{-\Omega(n)}$. This means that the probability that in $O(\mu n^{2k+1})$ steps the trap is found is less than:

$$O(\mu n^{2k+1})\cdot 2^{-\Omega(n)}=O(n^{k+2k+1})\cdot 2^{-\Omega(n)}=2^{-\Omega(n)}$$

This completes the proof of the second statement of the theorem.

Each individual of the (μ+1) EA$_R$ algorithm will not be initialised with a gap point with probability $1-2^{\Omega(n)}$ because $(8/10)n$ consecutive ones are required. Like in the proof of the first part of the theorem we consider the following phases:

1) The phase lasts until all the population of the (μ+1) EA$_R$ reaches the point with $(9/10)n-1$ leading ones or a trap point;
2) The phase lasts until all the population reaches a trap point;

We will consider the time required to end each phase assuming that the gap is not jumped first. Afterwards, we will consider the probability that an individual jumps over the gap before the two phases have finished.

Since the best ranked individual mutates each bit with probability $1/n$ and it gets selected with probability $1/\mu$, the ideas from the proof of the (1+1)-EA for LEADINGONES [3] may be adapted here. This individual will maintain the same mutation probability unless some individual gets more leading ones and gets ranked in first position. Since we are assuming the gap is not jumped over, in time at most $e\mu n^2$ the first individual in the population reaches $(9/10)n-1$ leading ones. At this point, a copy of the best individual of the population is created with probability $(1/\mu)\big(1-1/n\big)^n\geq 1/(4\mu)$ and μ copies are created in at most time $4\mu^2$. Summing up the expected time for phase 1 to finish is

$$e\mu n^2+4\mu^2=en^{2k}+4n^2k\leq 7n^{2k}.$$

Now we consider the second phase assuming that no trap points have been found yet first. Since the population has converged, the last ranked individual will mutate all its bits with probability 1 when selected, hence create a point with $n/10$ leading zeroes. Actually $(9/10)n-1$ leading zeroes will be created and only the last $n/10+1$ bits will be uniformly distributed. Since the probability for the last ranked individual to be selected is $1/\mu$ this event has an expected time of $\mu=n^k$. Then, the only improvement the individual may obtain is to reach the optimum which does not happen by hypothesis (i.e. it is a solution on the other side of the gap). Furthermore, no other individual may be ranked better unless it reaches the trap (i.e. under the assumption that the gap is not overtaken). Hence, just like at the end of phase 1, in expected time less than $4n^{2k}$ all the population will have been copied into a trap point and phase 2 concluded. This second part holds even when at least a trap point had been created before the end of phase 1. Summing up we get a total expected runtime of at most $11n^{2k}+n^k\leq 12n^{2k}$ steps for the two phases to end. By Markov's inequality with a probability of $1/2$ the phases are concluded in time $24n^{2k}$.

Now we consider the failure probability (i.e. the gap is jumped over before the two phases are concluded).

First we consider the probability if the number of leading ones in an individual is less than $n/10$. Then, the probability to jump over the gap is less than

$$\big(p_m\big)^{(2/10)n}\big(1-p_m\big)^{(2/10)n}\leq\big(1/2\big)^{-\Omega(n)}$$

because more than $(6/10)n$ uniformly distributed bits have to be turned into leading ones. This means that in expectation there are at least $(3/10)n$ zero-bits (and also at least $(3/10)n$ one-bits) and by Chernoff bounds they are at least $(2/10)n$ with probability $1-e^{-\Omega(n)}$.

Now we assume that there are more than $(n/10)$ leading ones. Then the probability to jump over the gap is less than the following

$$\big(p_m\big)^{2k+1}\big(1-p_m\big)^{(1/10)n-(2k+1)}\leq\big(1/n\big)^{2k+1}$$

because at least the $2k+1$ leading ones need to be turned into zeroes and the remaining $(1/10)n-(2k+1)$ ones should not be flipped. This implies that in each step the probability that a jump over the gap occurs is less than $n^{-(2k+1)}$. Hence the probability that the gap is not jumped over in $24n^{2k}$ steps is

$$\big(1-(1/n)^{2k+1}\big)^{24n^{2k}}\geq 1/e$$

Multiplying, the probability that the two phases occur in $24n^{2k}$ steps without any gap-jumps is $1/(2e)=\Omega(1)$.

Now, the only way to escape from the trap is to flip back at least all the $(8/10)n-2k$ zero-bits into one-bits without flipping any of the one-bits of the last $n/10$ bits of the string which are uniformly distributed. Such a probability is upper bounded as follows:

$$p_m^{(7/10)n}\cdot\big(1-p_m\big)^{n/30}\leq(1/2)^{n/30}=2^{-\Omega(n)}$$

This proves the exponential runtime for the (μ+1) EA$_R$ algorithm with probability at least $\Omega(1)$. ■

Now that it has been proved that there exist functions that are deceptive for the (μ+1) EA$_R$ but not for the (μ+1) EA, we will concentrate on the simple trap functions considered in [1]. The following trap function was considered.

$$\text{TRAP}_1(x) = \begin{cases} n+1 & \text{if } x = 0^n \\ \text{ONEMAX}(x) & \text{otherwise.} \end{cases}$$

The function consists of a ONEMAX path except for the optimum which is the bit string with all zeroes.

In the experiments performed in [1] the Rank-GA using mutation, crossover and fitness proportional selection required exponential time to optimise the TRAP_1 function. The following theorem proves that the (μ+1) EA_R is efficient for the function.

Theorem 4: Let $\mu > 1$. The expected optimization time of the (μ+1) EA_R on the TRAP_1 function is $O(\mu^2 + \mu n \log n)$.

Proof: The function consists of a ONEMAX path except for the 0^n bit-string which is the global optimum. If the global optimum is not found first, from Theorem 2 we know that the local optimum consisting of the 1^n bit-string will be found in time $O(\mu n \log n)$. From this point of time, due to the elitist nature of the selection mechanism this solution will not be removed from the population until the global optimum has been found because it has higher fitness than any other point in the search space. For the same reason, any copy of the local optimum will be accepted if the optimum has not been found,

As shown in the proof of Theorem 3, the expected time for the whole population to be a copy of the 1^n bit-string conditional to no fitness improvement (i.e. the optimum not being found) is $O(\mu^2)$. Now, since individual x_μ flips each bit with probability $(n/n) = 1$, it will flip all its bits (which are all ones) into zero-bits with probability 1 when it is selected for mutation. The expected time for x_μ to be selected for mutation is $O(\mu)$. Summing up, if the optimum is not found previously, it will be found in time $O(\mu^2 + \mu n \log n)$ ■

The above theorem proves that the (μ+1) EA_R is efficient for the TRAP_1 function. However, this only happens because the global optimum is the opposite of the local optimum (or if it is placed at a constant Hamming distance from the opposite). The following function changes the location of the local optimum to permit a fair comparison between the two algorithms.

$$\text{TRAP}_2(x) = \begin{cases} n+1 & \text{if } x = \{0^{n/4}1^{(3/4)n}\} \\ \text{ONEMAX}(x) & \text{otherwise.} \end{cases}$$

The following theorem shows that the expected optimization time of the (μ+1) EA_R on the TRAP_2 function is exponential in the function size.

Theorem 5: Let $\mu = poly(n)$. The expected optimization time of the (μ+1) EA_R on the TRAP_2 function is $\Theta(2^n)$.

Proof: The proof of the upper bound follows directly from Theorem 1.

The probability that the optimum is generated during the initialisation phase is $(1/2)^n$ for each individual. The expected number of zero bits for each individual is $n/2$. By Chernoff bounds, with overwhelming probability each individual has at least $n/3$ zero bits after initialisation. Hence for the optimum to be found at least $n/12$ zero bits have to be flipped into one bits. The probability that each individual is mutated into the optimum is

$$p_m^{H(x_i,x^*)} \cdot \left(1-p_m\right)^{n-H(x_i,x^*)}$$
$$\leq p_m^{n/12} \cdot (1-p_m)^{(11/12)n} \leq (1/2)^{n/12} = 2^{-(n/12)}$$

Hence, the expected number of mutation steps for the optimum to be found is at least $2^{n/12}$.

From the proof of Theorem 4 we know that the expected time for the best ranked individual (i.e. x_1) to find the 1^n bitstring and then to create μ identical copies of itself are respectively $O(\mu n \log n)$ and $O(\mu^2)$. By using Markov's inequality, we prove that with probability $1-o(1)$ the whole population consists of copies of the 1^n bitstring in time $\mu^2 n$. Once this point has been reached, the probability that any individual is mutated into the optimum is upper bounded as follows.

$$p_m^{(n/4)} \cdot \left(1-p_m\right)^{(3/4)n} \leq (1/2)^n = 2^{-n}$$

Hence, the expected time for the optimum to be found is $2^{\Omega(n)}$. ■

The expected time of the (μ+1) EA on the trap function is $\Omega(n^n)$ [3] meaning that the (μ+1) EA_R does require less time to optimize the function. However, none of the two algorithms perform better than Random Search on this function, which means they are inefficient for the function. In the next section a class of functions will be introduced were the better performance of the (μ+1) EA_R compared to the (μ+1) EA is a more practical gap between polynomial and super-polynomial runtimes. Thus a practical advantage of using the (μ+1) EA_R rather than the (μ+1) EA on that class of functions will be proved.

VI. COMBINING RANDOM AND GUIDED SEARCH

In this section, we want to point out where using rank-based mutation considerably speeds up the optimization process compared to algorithms using a fixed mutation rate. Here, we will show that the different individuals using different focuses on exploration and exploitation can significantly help to deal with landscapes that require different mutation rates at different stages of the optimization process.

To exemplify where the use of rank-based mutation can make the difference between a super-polynomial and polynomial runtime we consider the function TRAP-ONEMAX introduced in [5].

$$\text{TRAP-ONEMAX}(x) = \left(\prod_{i=1}^{k} x_i\right)\left(\sum_{i=k+1}^{n} x_i\right) + \sum_{i=1}^{k}(1-x_i).$$

We call the first k-bits the TRAP-part and the remaining $n-k$ bits the ONEMAX-part of a bitstring. The function has the property that the ONEMAX-part can only be optimized after the optimum of the TRAP-part has been found. Otherwise, the function leads an algorithm to search points that have a large Hamming distance in the TRAP-part from the set of optimal solution with respect to the TRAP-part which consists of all search points having at least k leading ones.

As done in [4, 5], we consider the function for the case $k = \log n$ and show that the use of rank-based mutation can considerably reduce the runtime. A similar effect has already been observed in [4] where the effect of using neutrality in evolutionary algorithms has been analyzed. In this paper it has been pointed out that a variant of the (1+1) EA has for each fixed mutation rate a super-polynomial expected optimization time. The arguments used in the proof of this lower bound can be generalized to (μ+1) EA if the population size is not too large. For a sufficiently large population size, e. g. $\mu = n^2 \log n$, and mutation rate $1/n$ the expected optimization time becomes polynomial as with high probability at least one individual in the initial population is optimal with respect to the TRAP-part.

In [4] it has been shown that the function TRAP-ONEMAX may also be optimized by EAs with a small population size. Incorporating neutrality into the (1+1) EA an upper bound on the runtime of $O(n^2 \log n)$ has been shown in this paper. The model of neutrality examined in this paper ensures that each bit in the TRAP-part is flipped with probability $1/2$ while each bit in the ONEMAX-part is flipped with probability $1/n$. For the ONEMAX-part the choice of the mutation rate is optimal. However, after having reached a solution with k leading ones and at least $k+1$ ones in the ONEMAX-part it is better to work with a smaller mutation rate in the TRAP-part as a mutation rate of $1/2$ implies that a solution with k leading ones is just re-sampled after an expected number of $\Theta(n)$ and the optimization of the ONEMAX-part is slowed down by $\Theta(n)$. Using these arguments together with the lower bound for the (1+1) EA on ONEMAX given in [3], it is not to hard to prove a matching lower bound of $\Omega(n^2 \log n)$ for the algorithm using bit-wise neutrality investigated in [4] on TRAP-ONEMAX.

We show that the (μ+1) EA_R optimizes the function TRAP-ONEMAX in time $O(n \log n)$ if the population size is constant. The improvement compared to the use of neutrality investigated in [4] is due to the fact that the mutation rate of the best individual is $1/n$ which implies that the optimum of the TRAP-part is re-sampled with a constant probability if the best individual of the population is chosen for mutation. Basically, our proof consists of the idea that individuals with a high mutation rate are necessary to sample the optimum of the TRAP-part for the first time. Later on, the ONEMAX-part is optimized by considering the best individual in the population, i. e. the individual with mutation rate $1/n$.

Theorem 6: Choosing $\mu > 2$, the expected optimization time of the (μ+1) EA_R on the TRAP-ONEMAX function with $k = \log n$ is $O(\mu n \log n)$.

Proof: To prove the theorem, we consider two phases. The first phase ends when a search point consisting of k 1-bits in the TRAP-part and at least $k+1$ 1-bits in the ONEMAX-part has been found for the first time. After having reached this intermediate goal the second phase begins and ends when the optimal search point has been found for the first time.

In the first phase we consider the individuals of rank i where $\mu/3+1 \leq i \leq \mu/2$. The probability of choosing such an individual in the next iteration is $\frac{\mu/2-\mu/3-1}{\mu} = \Omega(1)$. The TRAP-part consists of $\log n$ bits. Therefore, an expected number of at most $3^{\log n} = O(n)$ mutation steps applied to such individuals is necessary to reach a solution consisting of k leading ones. Such a solution is accepted if it has at least $k+1$ 1-bits in the ONEMAX-part.

As long as the TRAP-part has not been optimised, the ONEMAX-part does not contribute to the fitness, and the last $n-k$ bits in each individual are therefore uniformly distributed. Hence, by a Chernoff bound, with exponentially high probability, the ONEMAX-part contains at least $n/4 - k/4 > k+1$ ones when the TRAP-part has been optimised.

Altogether, the expected time until a solution with k 1-bits in the TRAP-part and at least $k+1$ 1-bits in the ONEMAX-part has been achieved is upper bounded by $O(n)$.

To optimize the ONEMAX-part, we can follow the ideas in the proof of Theorem 2 to obtain the upper bound of $O(\mu n \log n)$ on the expected time until an optimal solution has been achieved. ■

VII. DISCUSSION AND CONCLUSION

We have presented a rigorous analysis of rank-based mutation EAs on function classes with significant structures. We have considered the (μ+1) EA_R, which is a rank-based mutation steady state EA with elitism, and we have compared it with the (1+1) EA and the (μ+1) EA. The experiments performed in [1] discussed the impact of rank-based mutation rates by using an algorithm called Rank-GA with fitness-proportional selection and a crossover operator which chooses the mating individuals according to their rank. This could imply that some of the effects that were seen through the experiments were caused by the selection or the crossover operators. Now we discuss how the results presented in this paper compare with those obtained experimentally in [1].

The first result we have presented is a general bound of $O(2^n)$ for the (μ+1) EA_R which holds for any pseudo-Boolean function. This implies that the (μ+1) EA_R algorithm may only perform by a constant factor worse than Random Search. This runtime reflects the one obtained experimentally by the Rank-GA (i.e. with fitness-proportional selection and crossover) in [1] for functions such as NEEDLE and TRAP_1 since the number of fitness evaluations reported in the paper appear like exponential in the problem sizes (i.e. 10^5 for $n = 16$). This seems to imply that for these functions neither the selection or the crossover operator are of any help because there seems to be no evident runtime improvement compared to our upper bound.

In Theorem 2 we prove an upper bound of $O(\mu n \log n)$ for the (μ+1) EA_R for ONEMAX meaning that the algorithm is efficient for the function although it may be slower compared to the (μ+1) EA. The question of whether the bound is tight is left open for future work. In any case the algorithm does not require more than $O(\mu n \log n)$ expected time. This seems in line with the experimental results regarding the Rank-GA for ONEMAX presented in [1]. The algorithm seems to be

efficient for the ONEMAX function with a bitstring length of $n = 100$.

However, this does not explain the much worse performance of the Rank-GA for the TRAP_1 function. In Theorem 4 of Section V we prove that the (μ+1) EA_R is efficient for the TRAP_1 function. This happens because, once all the individuals in the population reach the local optimum by climbing up the ONEMAX path of the function, the individuals ranked badly flip many bits and end up on the global optimum which has Hamming distance n from the local optimum. In fact the last ranked individual will end up on the optimum with probability 1, once selected, because it flips all its bits. If the Rank-GA is really efficient for the ONEMAX function as claimed in [1], then it would be expected that once the top of the ONEMAX function is reached then the algorithm should be able to jump to the global optimum of the TRAP_1 function. However, this does not seem to be the case. One answer could be that, although the optimum of the ONEMAX part is found by the algorithm, the population of the Rank-GA algorithm does not converge to the top of the ONEMAX part. Hence the individuals with low rank and high mutation rate do not have a chance of jumping to the optimum in polynomial time. From the experimental results of [1] the answer is not clear.

For the functions discussed above, when the rank-based mutation performs better than fixed-mutation the runtime is exponential in the problem size and not better than Random Search. These results do not justify any practical advantage of using rank-based mutation rather than fixed mutation with a sensible mutation probability such as $p = 1/n$. For this reason in Section VI we have presented functions where we highlight that the optimization time of rank-based mutation is polynomial with good probability while algorithms with fixed-mutation rates are inefficient.

For fairness in the comparison, in Section V we have also proved the existence of functions where the (μ+1) EA_R is inefficient while the (μ+1) EA is efficient with a success probability converging fast to 1.

ACKNOWLEDGEMENT

Pietro S. Oliveto was supported by an EPSRC grant (EP/C520696/1). Per Kristian Lehre was supported by an EPSRC grant (EP/D052785/1).

The authors are grateful to Jonathan E. Rowe for an interesting discussion on the use of rank-based variation operators.

REFERENCES

[1] J. Cervantes and C. R. Stephens. Rank based variation operators for genetic algorithms. In *Proc. of GECCO '08*, pages 905–912. ACM Press, 2008.

[2] S. Droste, T. Jansen, and I. Wegener. A rigorous complexity analysis of the (1 + 1) evolutionary algorithm for separable functions with boolean inputs. *Evolutionary Computation*, 6(2):185–196, 1998.

[3] S. Droste, T. Jansen, and I. Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theor. Comput. Sci.*, 276:51–81, 2002.

[4] T. Friedrich and F. Neumann. When to use bit-wise neutrality. *Natural Computing*, 2009. To appear. A preliminary version appeared in Proc. of CEC 2008.

[5] W. J. Gutjahr and G. Sebastiani. Runtime analysis of ant colony optimization with best-so-far reinforcement. *Methodology and Computing in Applied Probability*, 10 (3):409–433, 2008.

[6] E. Happ, D. Johannsen, C. Klein, and F. Neumann. Rigorous analyses of fitness-proportional selection for optimizing linear functions. In *Proc. of GECCO '08*, pages 953–960. ACM Press, 2008.

[7] T. Jansen and I. Wegener. On the choice of the mutation probability for the (1+1) EA. In *Proc. of PPSN '00*, pages 89–98. Springer, 2008.

[8] F. G. Lobo, C. F. Lima, and Z. Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*. Springer, 2007. ISBN 978-3-540-69431-1.

[9] P. S. Oliveto and C. Witt. Simplified drift analysis for proving lower bounds in evolutionary computation. In *In Proceedings of the 10th International Conference on Parallel Problem Solving From Nature (PPSN X)*, pages 82–91, 2008.

[10] P. S. Oliveto, J. He, and X. Yao. Computational complexity analysis of evolutionary algorithms for combinatorial optimization: A decade of results. *International Journal of Automation and Computing*, 4(3):281–293, 2007.

[11] P. S. Oliveto, J. He, and X. Yao. Analysis of population-based evolutionary algorithms for the vertex cover problem. In *In proceedings of the 2008 IEEE world congress on computational intelligence (WCCI2008)*, pages 1563–1570. IEEE, 2008.

[12] C. Witt. Runtime analysis of the (μ + 1) EA on simple pseudo-boolean functions. *Evolutionary Computation*, 14(1):65–86, 2006.

[13] C. Zarges. Rigorous runtime analysis of inversely fitness proportional mutation rates. In *Proc. of PPSN '08*, pages 112–122. Springer, 2008.