The Lower Bounds on the Runtime of the $(1 + (\lambda, \lambda))$ GA on the Minimum Spanning Tree Problem

Matvey Shnytkin
ITMO University
St. Petersburg, Russia

Denis Antipov
ITMO University
St. Petersburg, Russia

ABSTRACT

Plenty of inspiring runtime analysis results have been recently obtained for the $(1 + (\lambda, \lambda))$ genetic algorithm (GA) on different benchmark functions. These results showed the efficiency of this GA, but we still do not have much understanding of its behavior on the real-world problems. To shed some light on this problem, we analyze the $(1 + (\lambda, \lambda))$ GA on the minimum spanning tree problem. We prove a lower bound of $\Omega(m^2/\lambda)$ fitness evaluations on its runtime, which shows that the considered GA with constant values of parameter λ does not significantly outperform the simple $(1 + 1)$ EA on this problem.

CCS CONCEPTS

• **Theory of computation** → **Theory of randomized search heuristics**; **Random search heuristics**;

KEYWORDS

Runtime Analysis, Minimum Spanning Tree, Crossover, Theory

ACM Reference Format:
Matvey Shnytkin and Denis Antipov. 2021. The Lower Bounds on the Runtime of the $(1 + (\lambda, \lambda))$ GA on the Minimum Spanning Tree Problem. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3449726.3463220

1 INTRODUCTION

In the field of evolutionary computation the theoretical community has made a decent contribution into the understanding of the working principles of the evolutionary algorithms (EAs). Surprisingly, even the runtime analysis of simple algorithms on simple benchmark problems turned out to be fruitful for helpful recommendations on how to use EAs in practice. Sometimes, the theoretical analysis of simple instances even resulted into developing new effective algorithms, such as the $(1 + (\lambda, \lambda))$ genetic algorithm (GA) [7].

The $(1 + (\lambda, \lambda))$ GA is a crossover-based algorithm which unlike most GAs uses crossover after the mutation. The main idea of this algorithm is to first use an aggressive mutation which is likely to find some beneficial changes in the current individual and then use a crossover as a repairing mechanism which reverts all the harmful changes produced by the strong mutation. This unusual mechanism was inspired by the study of the black-box complexity of the OneMax problem and from the observation that the standard algorithms do not benefit from inferior (in terms of fitness) solutions when solving this problem.

The theoretical analysis has showed that the $(1 + (\lambda, \lambda))$ GA has a good performance on multiple classes of benchmark functions. For OneMax it was shown in [6] that with population size λ slightly less than $\Theta(\sqrt{\log(n)})$ and with proper parameters for the mutation and crossover operators the runtime is slightly less than $O(n\sqrt{\log(n)})$, which is better than the runtime of any mutation-based algorithm on the same problem. It was also shown in [6] and [1] that the dynamic choice of λ leads to the $\Theta(n)$ runtime, which is the asymptotically best known runtime for the crossover-based algorithms on OneMax. For the LeadingOnes problem it was shown in [2] that the $(1 + (\lambda, \lambda))$ GA does not perform better than the simple $(1 + 1)$ EA, but at the same time it does not perform worse (in an asymptotical sense). The analysis of the $(1 + (\lambda, \lambda))$ GA on the non-unimodal class of Jump functions in [3] has showed that the proper choice of parameters (different from the recommendations given in [7]) can yield a runtime which is approximately the square root of the runtime of the best mutation-based algorithm on the same problem and it is also much better than the runtime of the classic GAs.

While all mentioned results were obtained for different classes of benchmark functions, there are only few results exist which consider the $(1 + (\lambda, \lambda))$ GA on the more practical problems. In one such work [5] it was shown that the algorithm works well on the MAX3-SAT problem, where the goal is to maximize the number of satisfied clauses in a 3-CNF boolean formula. Namely, it was shown that for the formulas with a small density of clauses the behavior of the $(1 + (\lambda, \lambda))$ GA is very similar to the one on OneMax. Several empirical studies also support the effectiveness of the $(1 + (\lambda, \lambda))$ GA on the RoyalRoad functions [7], on a broader class of MAX3-SAT instances [13] and on some combinatorial optimization problems [14].

At the same time there are plenty of results for the $(1 + 1)$ EA considering its runtime on different real-world problems [8, 10–12, 15–20]. The reason why the same theoretically proven results are still not obtained for the $(1 + (\lambda, \lambda))$ GA is that the core mechanism of the $(1 + (\lambda, \lambda))$ GA relies on the strong correlation between fitness and the distance to the optimum, which is necessary for the detection of the beneficial changes after an aggressive mutation via the fitness of the mutated individuals.

Our results. In this paper we aim at doing the first steps towards understanding of how the $(1 + (\lambda, \lambda))$ GA performs on the

GECCO '21, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8351-6/21/07…$15.00
https://doi.org/10.1145/3449726.3463220

real-world problems on graphs and if its mechanisms can help to outperform the classic algorithms. For this we analyse this GA on the minimum spanning tree problem (MST). We prove that the runtime of the $(1+(\lambda,\lambda))$ GA on all MST instances with a unique solution is at least $\Omega(\frac{m^2}{\lambda})$ fitness evaluations. This shows that for constant λ we cannot benefit more than a $\log(m\omega_{\max})$ factor compared to the hill-climbing $(1+1)$ EA, which has a runtime of $O(m^2\log(n\omega_{\max}))$ [16]. For the greater values of λ our lower bound is more optimistic, however, we conjecture that the actual lower bound is even higher in this case.

2 PRELIMINARIES

In this section we formally describe the analysed $(1+(\lambda,\lambda))$ GA, the MST problem and the fitness function we use.

2.1 The $(1+(\lambda,\lambda))$ GA

The $(1+(\lambda,\lambda))$ GA is a crossover-based algorithm which was specifically developed for the pseudo-Boolean optimization (however, recently a modification for the problems on permutations was proposed in [4]). The algorithm has three parameters, which are the *mutation rate* $p \in [0,1]$, the *crossover bias* $c \in [0,1]$ and the *population size* $\lambda \in \mathbb{N}$. The $(1+(\lambda,\lambda))$ GA stores one individual x, which is initialized with a random bit string of length n, where n is the problem size. Then it performs iterations, which consist of a mutation phase and a crossover phase. In the mutation phase the algorithm first chooses $\ell \sim \mathrm{Bin}(n,p)$, which is called the *mutation strength*. Then it creates λ offspring, each by flipping exactly ℓ bits in x chosen uniformly at random. Then it chooses the *mutation winner* x' as the offspring with the best fitness value. If there are several challengers, the winner is chosen from them uniformly at random. The main idea of such mutation is to perform a standard bit mutation λ times, but conditional on that all offspring must have the same distance from x. This is supposed to help to detect the beneficial bit flips via fitness and to choose an offspring with such flips as the mutation winner.

In the crossover phase we again create λ offspring, but this time we apply a crossover to the current individual x and the mutation winner x'. The crossover operator takes each bit from x' with probability c and from x with probability $1-c$. This mechanism is supposed to create an offspring which preserves the beneficial mutation made in x' while it repairs all destructive mutations by taking the corresponding bits from x. The pseudocode of the $(1+(\lambda,\lambda))$ GA is shown in Algorithm 1.

The recommendations for the relation between the three parameters of the algorithm were given in [7]. They suggest to set $p = \frac{\lambda}{n}$ and $c = \frac{1}{\lambda}$. This choice was shown to be optimal for the OneMax problem, but it also worked well on other problems such as LeadingOnes [2] and MAX-3SAT [5]. For this reason we rely on this recommendation in this paper.

2.2 The Minimum Spanning Tree Problem

In the MST problem we are given an undirected connected graph $G=(V,E)$ and a weight function $\omega : E \to \mathbb{R}^+$ and our aim is to find a connected subgraph $G'=(V,E')$, where $E' \subset E$ has the minimum weight. That is, we want to minimize the function $f(G') = \sum_{e\in E'}\omega(e)$ on the set of connected subgraphs of G. This is a relatively simple problem, and there exist algorithms which can solve it precisely in $O(|E|\log|E|)$ or $O(|V|\log|V|+|E|)$ time. However, in the context of the theory of evolutionary computation this problem lets us analyze the behavior of the EAs on graph problems [9, 16]. For this reason we chose this problem to make the first steps towards the runtime analysis of the $(1+(\lambda,\lambda))$ GA on graph problems.

Algorithm 1: The $(1+(\lambda,\lambda))$ GA minimizing a pseudo-Boolean function $f : \{0,1\}^n \to \mathbb{R}$.

```
x ← random bit string of length n;
while not terminated do
    Mutation phase:
    Choose ℓ ~ Bin(n, p);
    for i ∈ [1..λ] do
        x^(i) ← a copy of x;
        Flip ℓ bits in x^(i) chosen uniformly at random;
    end
    x' ← arg min_{z ∈ {x^(1),...,x^(λ)}} f(z);
    Crossover phase:
    for i ∈ [1..λ] do
        Create y^(i) by taking each bit from x' with
          probability c and from x with probability (1 − c);
    end
    y ← arg min_{z ∈ {y^(1),...,y^(λ)}} f(z);
    if f(y) ≤ f(x) then
        x ← y;
    end
end
```

To run the $(1+(\lambda,\lambda))$ GA on the MST problem we use the same representation of potential solutions and the same fitness function as in [16]. We represent graph $G'=(V,E')$ as a bit string of length $m := |E|$ in which each bit is equal to one if the corresponding edge from E is present in E' and is equal to zero otherwise. As a fitness function we use

$$f(G') = A^2 cc(G') + A|E'| + \sum_{e\in E'}\omega(e),$$

where $cc(G')$ is the number of connected components in G' and $A = \max\{\sum_{e\in E}\omega(e), |E|\}+1$. With this fitness function any connected graph has a better (smaller) fitness than any unconnected graph, any spanning tree has a better fitness than any connected graph with extra edges and among the spanning trees the best fitness belongs to the tree of minimum weight.

It was shown in [16] that when the $(1+1)$ EA optimizes this fitness function, it finds a connected graph in $O(m\log(n))$ iterations (where $n := |V|$), then it finds a spanning tree in another $O(m\log(n))$ iterations and then it needs $O(m^2\log(n\omega_{\max}))$ more iterations to transform this tree into the minimum one, where $\omega_{\max}$ is the maximal weight of an edge. An simple proof for the third phase of the optimization was also shown in [9].

3 THE RUNTIME ANALYSIS

In this section we show that the expected runtime of the $(1 + (\lambda, \lambda))$ GA on all graphs with a unique MST is $\Omega(\frac{m^2}{\lambda^2})$ iterations, if it starts with a non-minimum spanning tree. Since in each iteration we perform 2λ fitness evaluations, the runtime in terms of fitness evaluations is $\Omega(\frac{m^2}{\lambda})$.

3.1 Notation

First we introduce some definitions to help us prove the lower bound on the runtime. For all definitions we assume a fixed search point x representing a subtree of the considered graph in the start of iteration t.

- n — total number of verticies.
- m — total number of edges.
- $X_t = s$ — a random variable, which equals to the number of edges from the MST absent from the particular subtree described by the current individual x.
- $r = \{i^1 ... i^s, i^{s+1} ... i^{2s}\}$ — a set of flips required to complete the MST starting from the search point x. The first s indecies correspond to edges from the MST absent from x. The last s indecies correspond to edges from x which are not present in the MST. This set is fixed for a particular x.
- $e \in \{i^1 ... i^{\ell-2s}\}$ — a set of error flips after the mutation of the current individual x. Each i^j is a position in the bit string which is not the $2s$ required bits from r.
- p_ℓ — the probability to choose the particular mutation strength ℓ from $\mathrm{Bin}(n, p)$.
- x_e — a search point with r required bit flips and additional e error bit flips.
- $p_{m,e}$ — the probability of x_e winning the mutation stage.
- $p'_{m,e}$ — the probability of x_e appearing as one of λ mutants in the mutation stage.
- $p_{c,e}$ — the probability of crossover phase flipping back bits from e but leaving the necessary r bits untouched thus completing the MST.
- p_s — the probability to obtain the MST from the current individual x.

3.2 The Lower Bound

Since in every iteration the $(1 + (\lambda, \lambda))$ GA performs 2λ fitness evaluations, the transition between the runtime in terms of iterations and the runtime in terms of fitness iterations is trivial. Hence we formulate our result as the following theorem.

Theorem 3.1. *The expected number of fitness evaluations which the $(1 + (\lambda, \lambda))$ GA makes before it finds the optimum of the MST problem with a unique solution is $E[T] = \Omega(\frac{m^2}{\lambda})$, if it starts with a non-minimum spanning tree.*

To prove this claim we first introduce two auxiliary lemmas. Our aim is to analyze the last jump from non-optimal spanning tree with $X_t = s$ to the MST with $X_{t+1} = 0$. This implies that s edges of the MST yet absent from x replace s edges present in x but not in the MST.

We fix the number of bits ℓ which we flip in the mutation phase. Then for the success probability of the last optimisation step we have $p_s = \sum_{\ell=2s}^{m} p_\ell \sum_e p_{m,e} p_{c,e}$.

Lemma 3.2. *For all $l \in [2s, m]$ we have $\sum_e p_{m,e} \le \lambda \frac{(\ell-2s+1)\cdot\ldots\cdot\ell}{(m-2s+1)\cdot\ldots\cdot m}$.*

Proof. First, we determine the probability to generate x_e in the mutation stage $p'_{m,e}$. Note that $p_{m,e} \le p'_{m,e}$, since x_e may not be the mutant with the best fitness in a sample. Therefore, since for all $m \ge 2$ we have $\frac{1}{e} \ge (1 - \frac{1}{m})^m$, we obtain

$$\begin{aligned}\sum_e p_{m,e} &\le \sum_e p'_{m,e} \\ &= \sum_e \left[1 - \left(1 - \frac{(\ell-2s+1)\cdot\ldots\cdot\ell}{(m-2s+1)\cdot\ldots\cdot m}\binom{m-2s}{\ell-2s}^{-1}\right)^{\lambda}\right] \\ &\le \sum_e \left[1 - \left(1 - \lambda\frac{(\ell-2s+1)\cdot\ldots\cdot\ell}{(m-2s+1)\cdot\ldots\cdot m}\binom{m-2s}{\ell-2s}^{-1}\right)\right] \\ &= \sum_e \lambda\frac{(\ell-2s+1)\cdot\ldots\cdot\ell}{(m-2s+1)\cdot\ldots\cdot m}\binom{m-2s}{\ell-2s}^{-1}.\end{aligned}$$

Note that there exist $\binom{m-2s}{\ell-2s}$ different error sets, since we have $\ell - 2s$ additional edges to flip and $m - 2s$ options.

$$\sum_e \lambda\frac{(\ell-2s+1)\cdot\ldots\cdot\ell}{(m-2s+1)\cdot\ldots\cdot m}\binom{m-2s}{\ell-2s}^{-1} = \lambda\frac{(\ell-2s+1)\cdot\ldots\cdot\ell}{(m-2s+1)\cdot\ldots\cdot m}.$$

□

Lemma 3.3. *For all $l \in [2s, m]$ we have $p_{c,e}(\ell) \le \frac{1}{\lambda^{2s-1}}\left(\frac{1}{e}\right)^{\frac{\ell-2s}{\lambda}}$.*

Proof. The value of $p_{c,e}$ does not depend on particular e but only on the value of $\ell - 2s$. Therefore,

$$\begin{aligned}p_{c,e} &= 1 - \left(1 - \frac{1}{\lambda^{2s}}\left(1 - \frac{1}{\lambda}\right)^{\ell-2s}\right)^{\lambda} \le 1 - \left(1 - \frac{1}{\lambda^{2s}}\left(\frac{1}{e}\right)^{\frac{\ell-2s}{\lambda}}\right)^{\lambda} \\ &\le 1 - \left(1 - \frac{1}{\lambda^{2s-1}}\left(\frac{1}{e}\right)^{\frac{\ell-2s}{\lambda}}\right) = \frac{1}{\lambda^{2s-1}}\left(\frac{1}{e}\right)^{\frac{\ell-2s}{\lambda}}.\end{aligned}$$

In this computations we use that for all $m \ge 2$ we have $\frac{1}{e} \ge (1 - \frac{1}{m})^m$ and Bernoulli's inequality $(1 + x)^r \ge 1 + rx$. □

We are now in position to prove Theorem 3.1.

Proof of Theorem 3.1. By Lemmas 3.3 and 3.2, we estimate the upper bound on the probability to find the optimum in one iteration $p_s = \sum_{\ell=2s}^{m} p_\ell \sum_e p_{m,e} p_{c,e}$:

$$\begin{aligned}\sum_{\ell=2s}^{m} p_\ell \sum_e p_{m,e} p_{c,e} &\le \sum_{\ell=2s}^{m} p_\ell \frac{1}{\lambda^{2s-1}}\left(\frac{1}{e}\right)^{\frac{\ell-2s}{\lambda}} \sum_e p_{m,e} \\ &\le \sum_{\ell=2s}^{m} p_\ell \frac{1}{\lambda^{2s-2}}\left(\frac{1}{e}\right)^{\frac{\ell-2s}{\lambda}} \frac{(\ell-2s+1)\cdot\ldots\cdot\ell}{(m-2s+1)\cdot\ldots\cdot m}.\end{aligned}$$

Recalling that $p_\ell = \binom{m}{\ell}\left(\frac{\lambda}{m}\right)^{\ell}\left(1 - \frac{\lambda}{m}\right)^{m-\ell}$ we have

$$\sum_{\ell=2s}^{m} p_\ell \frac{1}{\lambda^{2s-2}}\left(\frac{1}{e}\right)^{\frac{\ell-2s}{\lambda}} \frac{(\ell-2s+1)\cdot\ldots\cdot\ell}{(m-2s+1)\cdot\ldots\cdot m}$$

$$= \frac{1}{\lambda^{2s-2}} \sum_{\ell=2s}^{m} \binom{m}{\ell} \left(\frac{\lambda}{m}\right)^{\ell} \left(1-\frac{\lambda}{m}\right)^{m-\ell}$$
$$\cdot \left(\frac{1}{e}\right)^{\frac{\ell-2s}{\lambda}} \frac{(\ell-2s+1)\cdot\ldots\cdot\ell}{(m-2s+1)\cdot\ldots\cdot m}$$
$$\leq \frac{1}{\lambda^{2s-2}} \sum_{\ell=2s}^{m} \frac{(m-2s+1)\cdot\ldots\cdot m}{(\ell-2s+1)\cdot\ldots\cdot\ell} \binom{m-2s}{\ell-2s}$$
$$\cdot \left(\frac{\lambda}{m}\right)^{\ell} \left(1-\frac{\lambda}{m}\right)^{m-\ell} \left(\frac{1}{e}\right)^{\frac{\ell-2s}{\lambda}} \frac{(\ell-2s+1)\cdot\ldots\cdot\ell}{(m-2s+1)\cdot\ldots\cdot m}$$
$$\leq \left(\frac{\lambda}{m}\right)^{2s} \frac{1}{\lambda^{2s-2}} \sum_{\ell=2s}^{m} \binom{m-2s}{\ell-2s} \left(e^{-\frac{1}{\lambda}}\frac{\lambda}{m}\right)^{\ell-2s} \left(1-\frac{\lambda}{m}\right)^{m-\ell}$$
$$= \frac{\lambda^2}{m^{2s}} \left(1-\frac{\lambda}{m}+e^{-\frac{1}{\lambda}}\frac{\lambda}{m}\right)^{m-2s}.$$

To simplify the last clause we use the Taylor formula for the exponential function $e^{-\frac{1}{\lambda}} = 1 - \frac{1}{\lambda} + \frac{1}{2\lambda^2} + o(\frac{1}{\lambda^2})$, where the $o(\frac{1}{\lambda^2})$ term is negative. Hence, we determine

$$\frac{\lambda^2}{m^{2s}} \left(1-\frac{\lambda}{m}+e^{-\frac{1}{\lambda}}\frac{\lambda}{m}\right)^{m-2s}$$
$$= \frac{\lambda^2}{m^{2s}} \left(1-\frac{\lambda}{m}+\left(1-\frac{1}{\lambda}+\frac{1}{2\lambda^2}+o\left(\frac{1}{\lambda^2}\right)\right)\frac{\lambda}{m}\right)^{m-2s}$$
$$\leq \frac{\lambda^2}{m^{2s}} \left(1-\frac{1}{m}+\frac{1}{2\lambda m}\right)^{m-2s} = \frac{\lambda^2}{m^{2s}} \left(1-\frac{1}{m}\left(1-\frac{1}{2\lambda}\right)\right)^{m-2s}$$
$$\leq \frac{\lambda^2}{m^{2s}} \left(1-\frac{1}{2m}\right)^{m-2s} \leq \frac{\lambda^2}{m^{2s}} \left(\frac{1}{e}\right)^{\frac{m-2s}{2m}} \leq \frac{\lambda^2}{m^{2s}}.$$

Now we estimate $E[T_{last}]$, the expected runtime on the last step. For this we note that the last step is the series of trials until obtaining the MST. Then the amount of iterations until obtaining the MST can be described by the geometric distribution with parameter $p \leq \frac{\lambda^2}{m^{2s}}$, which reaches its maximum in $s = 1$. Recalling that the expectation on the number of trials until the first success is $\frac{1}{p}$ we have $E[T_{last}] = \frac{1}{p} = \Omega(\frac{m^2}{\lambda^2})$. Finally, using $E[T_{last}] \leq E[T]$ we obtain the result of the theorem. □

4 CONCLUSION

In this paper we analysed the $(1+(\lambda,\lambda))$ GA on MST problem with unique solution and proved that its runtime is $\Omega(\frac{m^2}{\lambda})$ fitness evaluations. This means that for $\lambda = \Theta(1)$ we do not have much advantage over the $O(m^2 \log(m\omega_{\max}))$ runtime of the $(1+1)$ EA. Although for $\lambda = \Omega(1)$ our lower bound is more optimistic, we conjecture that with high values of λ it becomes hard to detect good mutations in the mutation offspring, since with an aggressive mutation the better individual is not the one with the right bit flip, but the one with less connected components. This high priority of the number of connected components in the fitness function on the one hand helps us to find a connected graph faster, but on the other hand it shadows the beneficial mutations in the late optimisation stages.

For the further research of the $(1+(\lambda,\lambda))$ GA on this problem we suggest to modify the algorithm so that it was more tailored for the problem. Among the most natural modifications we see using a different fitness-function or using a more specific mutation operator, which maintains an invariant that any mutation offspring is still a spanning tree.

ACKNOWLEDGMENTS

The reported study was funded by RFBR and CNRS, project number 20-51-15009.

REFERENCES

[1] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. 2020. Fast mutation in crossover-based algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2020*. ACM, 1268–1276.
[2] Denis Antipov, Benjamin Doerr, and Vitalii Karavaev. 2019. A tight runtime analysis for the $(1+(\lambda,\lambda))$ GA on LeadingOnes. In *Foundations of Genetic Algorithms, FOGA 2019*. ACM, 169–182.
[3] Denis Antipov, Benjamin Doerr, and Vitalii Karavaev. 2020. The $(1+(\lambda,\lambda))$ GA is even faster on multimodal problems. In *Genetic and Evolutionary Computation Conference, GECCO 2020*. ACM. To appear.
[4] Anton Bassin and Maxim Buzdalov. 2020. The $(1+(\lambda,\lambda))$ genetic algorithm for permutations. In *Genetic and Evolutionary Computation Conference, GECCO 2020, Companion Material*. ACM, 1669–1677.
[5] Maxim Buzdalov and Benjamin Doerr. 2017. Runtime analysis of the $(1+(\lambda,\lambda))$ genetic algorithm on random satisfiable 3-CNF formulas. In *Genetic and Evolutionary Computation Conference, GECCO 2017*. ACM, 1343–1350.
[6] Benjamin Doerr and Carola Doerr. 2018. Optimal static and self-adjusting parameter choices for the $(1+(\lambda,\lambda))$ genetic algorithm. *Algorithmica* 80 (2018), 1658–1709.
[7] Benjamin Doerr, Carola Doerr, and Franziska Ebel. 2015. From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science* 567 (2015), 87–104.
[8] Benjamin Doerr, Edda Happ, and Christian Klein. 2007. A tight bound for the (1 + 1)-EA for the single source shortest path problem. In *Congress on Evolutionary Computation, CEC 2007*. IEEE, 1890–1895.
[9] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. 2010. Multiplicative drift analysis. In *Genetic and Evolutionary Computation Conference, GECCO 2010*. ACM, 1449–1456.
[10] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. 2012. Multiplicative drift analysis. *Algorithmica* 64 (2012), 673–697.
[11] Benjamin Doerr, Frank Neumann, and Andrew M. Sutton. 2015. Improved runtime bounds for the $(1+1)$ EA on random 3-CNF formulas based on fitness-distance correlation. In *Genetic and Evolutionary Computation Conference, GECCO 2015*. ACM, 1415–1422.
[12] Stefan Droste, Thomas Jansen, and Ingo Wegener. 1998. A rigorous complexity analysis of the 1 + 1 evolutionary algorithm for separable functions with boolean inputs. *Evolutionary Computation* 6 (1998), 185–196.
[13] Brian W. Goldman and William F. Punch. 2014. Parameter-less population pyramid. In *Genetic and Evolutionary Computation Conference, GECCO 2014*. ACM, 785–792.
[14] Vladimir Mironovich and Maxim Buzdalov. 2015. Hard test generation for maximum flow algorithms with the fast crossover-based evolutionary algorithm. In *Genetic and Evolutionary Computation Conference, GECCO 2015, Companion Material*. 1229–1232.
[15] Frank Neumann and Andrew M. Sutton. 2019. Runtime analysis of the (1 + 1) evolutionary algorithm for the chance-constrained knapsack problem. In *Foundations of Genetic Algorithms, FOGA 2019*. ACM, 147–153.
[16] Frank Neumann and Ingo Wegener. 2007. Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. *Theoretical Computer Science* 378 (2007), 32–40.
[17] Pietro Simone Oliveto, Jun He, and Xin Yao. 2009. Analysis of the (1+1)-EA for finding approximate solutions to vertex cover problems. *IEEE Transactions on Evolutionary Computation* 13 (2009), 1006–1029.
[18] Feng Shi, Frank Neumann, and Jianxin Wang. 2019. Runtime analysis of evolutionary algorithms for the depth restricted (1, 2)-minimum spanning tree problem. In *Foundations of Genetic Algorithms, FOGA 2019*. ACM, 133–146.
[19] Carsten Witt. 2005. Worst-case and average-case approximations by simple randomized search heuristics. In *Symposium on Theoretical Aspects of Computer Science, STACS 2005*. Springer, 44–56.
[20] Carsten Witt. 2014. Revised analysis of the (1+1) EA for the minimum spanning tree problem. In *Genetic and Evolutionary Computation Conference, GECCO 2014*. ACM, 509–516.