PRIFYSGOL ABERYSTWYTH UNIVERSITY

Aberystwyth University

The (1+(λ,λ)) Genetic Algorithm on the Vertex Cover Problem

Buzdalov, Maxim

Published in:
Proceedings of IEEE Congress on Evolutionary Computation

DOI:
10.1109/CEC55065.2022.9870224

Publication date:
2022

Citation for published version (APA):
Buzdalov, M. (2022). The (1+(λ,λ)) Genetic Algorithm on the Vertex Cover Problem: Crossover Helps Leaving Plateaus. In *Proceedings of IEEE Congress on Evolutionary Computation* (2022 IEEE Congress on Evolutionary Computation, CEC 2022 - Conference Proceedings). IEEE Press.
https://doi.org/10.1109/CEC55065.2022.9870224

General rights
Copyright and moral rights for the publications made accessible in the Aberystwyth Research Portal (the Institutional Repository) are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the Aberystwyth Research Portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the Aberystwyth Research Portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

tel: +44 1970 62 2400
email: is@aber.ac.uk

Download date: 30. Sep. 2023

The $(1 + (\lambda, \lambda))$ Genetic Algorithm on the Vertex Cover Problem: Crossover Helps Leaving Plateaus

Maxim Buzdalov
ITMO University
Saint Petersburg, Russia
mbuzdalov@gmail.com

***Abstract*—Many discrete optimization problems feature plateaus, which are hard to evolutionary algorithms due to the lack of fitness guidance. While higher mutation rates may assist in making a jump from the plateau to some better search point, an algorithm typically performs random walks on a plateau, possibly with some assistance from diversity mechanisms.**

The vertex cover problem is one of the important NP-hard problems. We found that the recently proposed $(1 + (\lambda, \lambda))$ genetic algorithm solves certain instances of this problem, including those that are hard to heuristic solvers, much faster than simpler mutation-only evolutionary algorithms.

Our theoretical analysis shows that there exists an intricate interplay between the problem structure and the way crossovers are used. It results in a drift towards the points where finding the next improvement is much easier. While this condition is formally proven only on one class of instances and for a subset of search points, experiments show that it is responsible for performance improvements in a much larger range of cases.

I. Introduction

Theoretical understanding of discrete evolutionary algorithms [1], [2] is nowadays not limited to universal statements, like convergence to the global optimum with probability 1, and to analysis on artificially constructed problems with simple definitions that highlight particular properties of an algorithm. A lot of work has been recently done for discrete combinatorial optimization problems, including various problems on graphs, such as minimal spanning trees [3]–[5], the traveling salesperson problem both by conventional [6]–[8] and gray-box algorithms [9], [10], shortest path problems [11]–[13]. and many other problems.

One of such problems is the (minimum) vertex cover problem. Given a graph, one needs to select a subset of its vertices of the minimum size, such that for each edge at least one of its endpoints is selected. It is NP-hard [14], and quite hard to solve in practice compared to some other NP-hard problems. Hence, to obtain satisfactory results on real-world instances, various approximation algorithms are used [15], with a typical guarantee being an approximation factor of $2 - o(1)$. Evolutionary algorithms have also been reported to solve the vertex cover problem quite well [16], [17] either on their own or as a refinement procedure applied to the results of other approximation algorithms [18].

However, the vertex cover problem appears to be an interesting benchmark problem for studying how evolutionary algorithms can leave local optima in the favor of better search points including the global optimum. The latter has been a research topic for the recently developed algorithms, such as the $(1+(\lambda, \lambda))$ genetic algorithm, or the $(1+(\lambda, \lambda))$ GA. This algorithm, proposed in [19], is designed in such a way that it has notable theoretical guarantees of improved performance compared to the algorithms using only mutation operators. Such guarantees have been initially proven only for very simple unimodal problems [20], however, later similar advantages have also been proven for simply-defined multimodal problems [21] with relatively small parameter changes.

In this respect, the $(1+(\lambda, \lambda))$ GA has the fate similar to the "fast genetic algorithms" using heavy-tailed distributions [22]. An interesting fact is that all existing works on both algorithmic families regarding multimodal functions only consider the problems where the only way to leave a local optimum is to jump directly to the global optimum, possibly with some guidance from inferior solutions.

Currently, not much is known how the $(1+(\lambda, \lambda))$ GA copes with the vertex cover problems. We make first steps towards this direction, already with some insights that we find striking and inspiring. Namely, we have identified a regime when the $(1 + (\lambda, \lambda))$ GA performs a random walk on a fitness plateau, like the mutation-only algorithms, however, unlike them, the crossover biases this walk towards the exit from this plateau.

The rest of the paper is organized as follows. **Section II** introduces the vertex cover problem, definitions of the algorithms and recalls the necessary results from the literature. The next sections consider different classes of the vertex cover problem. **Section III** considers the bipartite graph, where the $(1 + (\lambda, \lambda))$ GA outperforms the $(1 + 1)$ EA by employing a kind of divide-and-conquer scheme to jump from the local optimum to the global one. Here, the heavy-tailed versions of the algorithms solve the problem much easier due to the properties of the underlying power-law distributions. **Section IV** considers another instance, the so-called Papadimitrou-Steiglitz graph, where the $(1+(\lambda, \lambda))$ GA essentially *uses crossover to introduce drift* towards the switchpoint to the global optimum on a plateau of local optima. Note that in this case using heavy-tailed distributions alone does not improve the performance to the same extent as using the crossover in the way the $(1 + (\lambda, \lambda))$ GA does. **Section V** evaluates all the algorithms on randomly generated instances of the vertex cover problem. The results suggest that both of the scenarios above appear "in the wild" with similar frequencies. Finally, **Section VI** gives the concluding remarks.

II. Definitions and Related Work

A. The $(1+1)$ EA with Binomial and Power-Law Distributions

The $(1 + 1)$ EA is the simplest evolutionary algorithm, which received a lot of attention in theoretical analysis [23]. It has only one parent, and on every iteration it generates one offspring via a mutation operator, which then replaces the parent if it is not worse than the parent by the fitness value.

The pseudocode of the $(1+1)$ EA is given in Algorithm 1. Note that it is defined using an arbitrary mutation strength distribution $\mathcal{M}$, since lots of work has been recently done on how these distributions may, or should, look like. The standard bit mutation is represented as the binomial distribution over the number of flipped bits, that is, $\mathcal{M} \leftarrow \mathcal{B}(n, 1/n)$ for the problem size n. However, recent research proposed the use of the heavy-tailed distributions, such as the power-law distribution [22]. The evolutionary algorithms with heavy-tailed distributions are commonly known as "*fast*" evolutionary algorithms. In power-law distributions, the probability of sampling i bits is proportional to $i^{-\beta}$, where β is the control parameter that should be greater than 1. The paper [22] used a slightly different approach, namely, it sampled a number i from such a heavy-tailed distribution and then flipped each bit with probability i/n. From this point of view, $\mathcal{M}$ is a tricky composition of a heavy-tailed distribution and of a binomial distribution.

Some of the recent research also advocated for use of truncated distributions that never produce zero values [24], since flipping zero bits never helps in noise-free optimization. (Note that in population-based algorithms flipping zero bits may introduce a clone of an existing individual to the population, which may serve good purposes; however, reformulating such algorithms to avoid the necessity of creating a clone is arguably a better approach). This saves a constant factor in many analyses, which even helped in the past to see the effects that have been otherwise hindered [25].

In this work, we consider two mutation distributions for $\mathcal{M}$.

- The *resampling* mutation $x \sim \mathcal{B}(n, 1/n) \mid x > 0$, proposed in [24].
- The power-law mutation with $\Pr[x] = x^{-\beta}/(\sum_{i=1}^{n} i^{-\beta})$. Note that this is somewhat different than [22]: first, by directly using the power law without having a binomial distribution as a proxy, and second, by allowing to flip up to n bits instead of only up to $n/2$ bits. This is the approach adopted in the Nevergrad optimization platform [26]. Due to the binomial distribution being well-concentrated for $n \cdot p \gg 1$, getting rid of it as a proxy distribution leaves most of the properties of the algorithm intact, and allowing to flip up to n bits was shown to improve the performance in certain scenarios [27]. This mutation operator also flips a positive number of bits.

Slighly abusing the notation, we call the $(1+1)$ EA with the former mutation operator simply the $(1+1)$ EA, and with the latter operator, with β chosen to be equal to 1.1 by preliminary experiments, the $(1 + 1)$ FEA where "F" means *fast*.

Algorithm 1 The $(1 + 1)$ EA to maximize $f : \{0, 1\}^n \to \mathbb{R}$

Require: mutation strength distribution $\mathcal{M}$
  $x \leftarrow$ sample uniformly from $\{0, 1\}^n$
  query $f(x)$
  while true do
    $\ell \leftarrow$ sample from $\mathcal{M}$
    $y \leftarrow$ flip ℓ pairwise different, uniformly chosen bits in x
    query $f(y)$
    if $f(y) \geq f(x)$ **then**
      $x \leftarrow y$
    end if
  end while

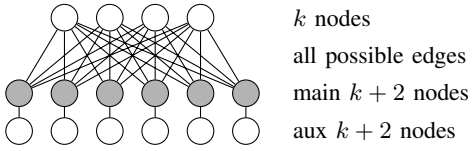

Fig. 1. The Papadimitrou-Steiglitz graph and its minimum cover in gray

B. The Vertex Cover Problem

Given an undirected graph (V, E) with $v = |V|$ vertices and $e = |E|$ edges, a *vertex cover* is a subset of vertices $V_C \subseteq V$ such that for any edge $(v_1, v_2) \in E$ either of its endpoints belongs to this subset: $(v_1 \in V_C) \vee (v_2 \in V_C)$. The *minimum vertex cover problem* is to find a vertex cover of the given graph such that it has the minimum possible size. Its decision version, the *vertex cover problem*, decides whether there exists a vertex cover of at most the given size, and it is NP-complete [14], hence the minimization problem is NP-hard. In this paper we consider the minimization problem, and call it simply the *vertex cover problem* for brevity.

The vertex cover problem is quite hard even for approximation; it is known that its polynomial worst-case approximation cannot be better than a factor of 2 if the unique game conjecture is true [28]. Hence it is subject to much research on approximation algorithms. In particular, in the book [15, Chapter 15, Section 6], a heuristic local search algorithm called *Vercov* has been proposed and analyzed. The analysis includes a family of problem instances where *Vercov* produces the worst approximation, one of which has been used in many subsequent papers and was called the Papadimitrou-Steiglitz graph, or the PS-graph. This is a complete bipartite graph on $k + 2$ and k vertices, where each of the $k + 2$ vertices has an additional vertex attached by a single edge (Figure 1). This graph is a subject of study in one of the sections of this paper.

To be solved by an evolutionary algorithm, we need to determine the representation and the fitness function. Our representation is a bit string of size v, where the bit value of 1 means the corresponding vertex is selected, and 0 means it is not. The fitness function is the same as in the single-objective works, say [29]: let x be an individual, and $e(x)$ be the number of edges uncovered by the vertices selected by x, then the fitness function is $f(x) = e(x) \cdot v + |x|$. This way,

if two individuals cover a different number of edges, the one that covers more is always better. This definition is equivalent to minimizing the tuple $\langle e(x), |x| \rangle$ lexicographically.

C. Vertex Cover and Theory of Evolutionary Algorithms

In probably the first work on theoretical analysis of the behavior of evolutionary algorithms on the vertex cover problem [30], He et al. analyzed the simplest evolutionary algorithm called the $(1+1)$ EA, as well as its two modifications heuristically tailored to the vertex cover problem. These algorithms use the straightforward representation, namely the vector of bits corresponding to whether to take a vertex into the cover, and the fitness function which aims at minimizing the number of uncovered edges first, and only then at minimizing the number of chosen vertices. They proved that the $(1+1)$ EA reaches the feasible vertex cover in time $O(v \cdot e)$, where v is the number of vertices and e is the number of edges, and finds the minimum vertex cover in finite but potentially very large time. They also performed experiments with two classes of instances, one that is too hard for all the considered algorithms and another one that is much easier.

In [29], a number of important results have been proven regarding the behavior of simple evolutionary optimizers, such as the $(1+1)$ EA, on certain instances of the vertex cover problem, including the PS graph, the bipartite graph, two instance classes that are hard to solve with overwhelming probability, and one easy instance class with vertex having degrees at most 2.

In particular, [29, Lemma 4] proves that RLS finds the optimum on a certain instance of the vertex cover problem in time $O(v \log v)$ with at least constant probability close to $1/2$. This is also the characteristic of many other evolutionary algorithms that do not employ restarts, including the $(1+1)$ EA [29, Lemma 8], and of many other instances of the vertex cover problem. With the remaining probability, the search is likely to hit a local optimum, from which it is difficult to get to the global one. In particular, in [30] such a bimodal behavior was spotted for the $(1+1)$ EA on another problem instance. That paper, however, did not provide any explanation or insight regarding this behavior.

For certain problem instances the local optima with the same fitness have a structure that allows the optimizer to walk between them freely. In particular, [29, Lemma 10], for the studied problem instance, proves that even if the optimizer is in the local optimum that is just $O(1)$ bit flips apart from the attractor of the global optimum, it is still quite likely to go away to a distance that would require exponential time to reach the global optimum.

In [31] similar results have been proven for simple population-based algorithms, the $(1+\lambda)$ EA and the $(\mu+1)$ EA. Most of these results are really the same as the ones for the $(1+1)$ EA, either with or without restarts. However, using crossover in an accurate way may improve the efficiency of the $(\mu+1)$ EA, but only if this crossover is well-aligned with the graph structure, which one cannot expect in the wild.

The work [18], unlike the previous works (which considered initializing the algorithms uniformly, by all ones or all zeros), used existing approximation algorithms for initialization. The employed approximation algorithms were the greedy algorithm, that takes a random uncovered vertex with the maximum degree until all vertices are covered, and the maximum matching algorithm. This way, an elitist evolutionary algorithm would produce results which are not worse than the approximation algorithm. Some cases were found where the initial approximation is quickly and significantly improved by an evolutionary algorithm, in other cases no significant improvement is possible in polynomial time.

Recent work considered multiobjective problem settings, which on many occasions allow to converge faster and to a better approximation [32], weighted vertex cover problems [33], dynamic vertex cover problems [34], made use of the dual linear programming formulation [35] and proved upper bounds that are exponential not in the problem size, but in the size of the problem's *core*, which is the hardest part, thus making evolutionary algorithms the randomized fixed-parameter tractable algorithms for the vertex cover problem [36].

D. The $(1+(\lambda,\lambda))$ Genetic Algorithm and Its Modifications

The $(1+(\lambda,\lambda))$ genetic algorithm is proposed in [19] as an example of a full-scale evolutionary algorithm that is designed with inspiration by the black-box complexity theory and, in particular, can solve a simple benchmark problem called ONEMAX asymptotically faster than any mutation-only algorithm. It is also a showcase of the online parameter control [20], as the version with the one-fifth rule that controls its parameter λ is asymptotically better than any fixed choice. This algorithm was also shown, and sometimes proven, to be better than many other algorithms on more practical problems [37]–[39]. Recently, instead of the active stateful parameter control, sampling the parameters randomly from heavy-tailed distributions has been proposed in [21], [40], [41] with different degrees of freedom and similar performance advantages.

The generic form of the $(1+(\lambda,\lambda))$ GA is given in Algorithm 2. It features two phases in an iteration. The *mutation phase* samples a number of offspring at the same distance ℓ from the parent, which is usually large. The best of these offspring is then crossed over, again multiple times, with the parent in the *crossover phase* in such a way that much more bits come from the parent than from the offspring. The result of the crossover competes with the parent, replacing it if it is not worse than the parent.

The flavors of the $(1+(\lambda,\lambda))$ GA differ in how they treat their parameters λ, p and c, which control the population size, mutation rate and crossover rate respectively. We consider the following ones:

- The self-adjusting version with the one-fifth rule. In this version, λ is subject to self-adjustment, and other parameters are set as $p \leftarrow \lambda/n$ and $c \leftarrow 1/\lambda$. Initially, $\lambda \leftarrow 1$, and every time the best fitness improves at the end of the iteration, $\lambda \leftarrow \lambda/C$, where $C \approx 1.5$ is a constant.

Algorithm 2 The $(1+(\lambda,\lambda))$ GA to max. $f : \{0,1\}^n \to \mathbb{R}$

$n \leftarrow$ the problem size
$x \leftarrow$ uniformly at random (u.a.r.) from $\{0,1\}^n$
for $t \leftarrow 1, 2, 3, \ldots$ **do**
  Choose λ, p, c, $\lambda' \leftarrow \lfloor \lambda \rfloor$, $\ell \sim \mathcal{B}(n,p)$
  for $i \in \{1, 2, \ldots, \lambda'\}$ **do** ▷ Phase 1: Mutation
    $x^{(i)} \leftarrow$ flip ℓ uniformly chosen bits in x
  end for
  $x' \leftarrow$ u.a.r. from $\{x^{(j)} \mid f(x^{(j)}) = \max\{f(x^{(i)})\}\}$
  for $i \in \{1, 2, \ldots, \lambda'\}$ **do** ▷ Phase 2: Crossover
    for $j \in [n]$ **do**
      $y_j^{(i)} \leftarrow x'_j$ with probability c, otherwise x_j
    end for
  end for
  $y \leftarrow$ u.a.r. from $\{y^{(j)} \mid f(y^{(j)}) = \max\{f(y^{(i)})\}\}$
  if $f(y) \geq f(x)$ **then** ▷ Selection
    $x \leftarrow y$
  end if
end for

Otherwise, $\lambda \leftarrow \lambda \cdot C^{1/4}$. Then λ is truncated to the safe region of $[1; \overline{\lambda}]$ where $\overline{\lambda} \geq n$ is the upper bound. We consider two versions: $\overline{\lambda} = 2\ln(n+1)$ for the reasons detailed in [39], and a more aggressive $\overline{\lambda} = n/4$ based on preliminary experiments.

- The single-parameter heavy-tailed version. Here, λ is chosen from the power-law distribution with $\beta_\lambda \in (2;3)$, while $p = \lambda/n$ and $c = 1/\lambda$. The smaller β, the more often large λ values are chosen, and since we are dealing with a hard optimization problem, we chose $\beta_\lambda = 2.1$, although its impact on the performance is not very strong.
- The three-parameter heavy-tailed version. Here, not only λ, but also other parameters are chosen from the power-law distributions: $p \leftarrow p'/n$ and $c \leftarrow c'/n$, where p' and c' are sampled from distributions with $\beta_p > 1$ and $\beta_c > 1$. This scheme was proposed and motivated in [41] on the example of the so-called jump benchmark functions. We chose $\beta_\lambda = 2.1$ for λ and $\beta_p = \beta_c = 1.1$ for the same reasons as above.

We also adjusted our implementations of the $(1+(\lambda,\lambda))$ GA variants so that they do not waste fitness evaluations. Namely, all the binomial distributions are also resampled when they produce zeros, the crossover phase is skipped when the Hamming distance of mutants to the parent is one, the crossover offspring identical to the best mutant are not evaluated, and the best mutation offspring also competes with the parent in the final selection stage.

III. The Bipartite Graph

The first part of our study considers the bipartite graph of size $v = 2k+1$ as an instance class. The two halves have the size of k and $k+1$ respectively. The minimum vertex cover obviously has the size of k by choosing all the vertices from the small half, whereas the second-best local optimum chooses all the opposite vertices, and all other individuals have a worse fitness value.

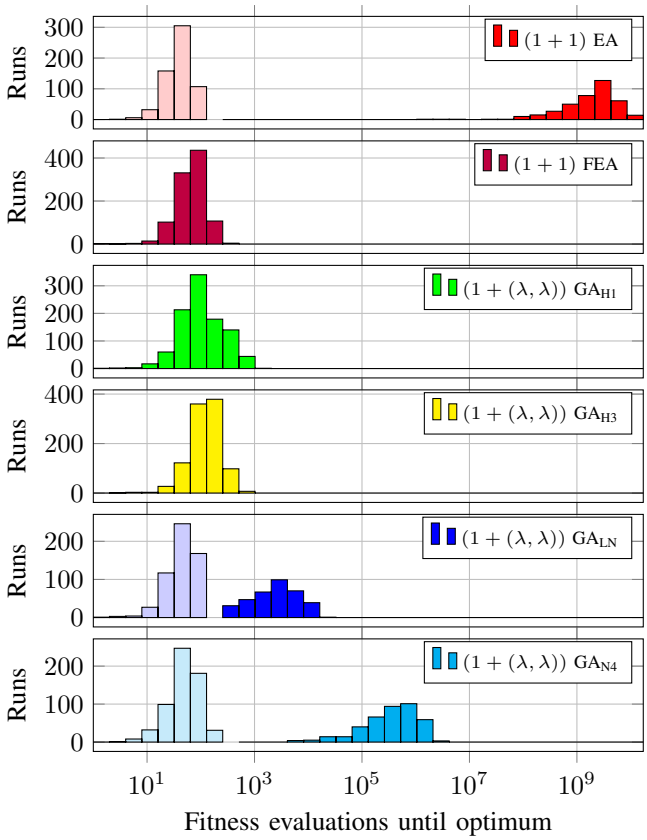

Fig. 2. Distribution histograms: bipartite graphs, $v = 11$, 1000 runs

We perform experiments first. We consider the following six algorithms, whose definitions were given above:

- **(1+1) EA**: the $(1+1)$ EA with a (resampling) binomial mutation operator;
- **(1+1) FEA**: the $(1+1)$ EA with a heavy-tailed mutation operator;
- **$(1+(\lambda,\lambda))$ GA$_{\text{LN}}$**: the $(1+(\lambda,\lambda))$ GA with self-adjusted $\lambda \leq 2\ln(v+1)$;
- **$(1+(\lambda,\lambda))$ GA$_{\text{N4}}$**: the same with $\lambda \leq v/4$;
- **$(1+(\lambda,\lambda))$ GA$_{\text{H1}}$**: the heavy-tailed $(1+(\lambda,\lambda))$ GA with only λ sampled randomly;
- **$(1+(\lambda,\lambda))$ GA$_{\text{H3}}$**: the same with all three parameters sampled randomly.

Each of these algorithms is initialized with uniform sampling of each bit in the individual. We consider small half sizes $k \in [2;30]$, hence $v \in \{5, 7, \ldots, 61\}$. Since non-heavy-tailed algorithms show runtimes exponential in n, the maximum n for the $(1+1)$ EA is 11, for the $(1+(\lambda,\lambda))$ GA$_{\text{N4}}$ it is 15, and for the $(1+(\lambda,\lambda))$ GA$_{\text{LN}}$ it is 19. For each problem size and each algorithm, 1000 independent runs have been performed.

Before presenting the results of these experiments, we shall discuss the shapes of the runtime distrubutions. For $v = 11$, their histograms are given in Fig. 2. One can see that non-heavy-tailed algorithms show clear bimodal distributions with

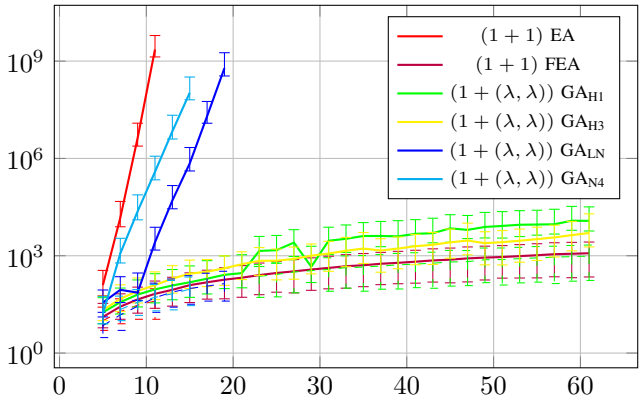

Fig. 3. Peak-median runtimes: bipartite graphs, 1000 runs. Solid lines correspond to the peak with larger runtimes, dashed lines indicate the first peak when present. Medians are plotted, error bars show interquartile ranges.

vastly differing running times. From theoretical works we know [29] that the $(1+1)$ EA converges in time $O(v \log v)$ to either the local or the global optimum, and in the former case it needs at least exponential time to reach the global optimum, as it needs to flip all the bits at once. This explains both the presence of two peaks and the fact that they have roughly equal probability to happen, as the probability to hit either optimum is bound from below by a constant.

For this reason, we cannot simply present the running times for various sizes with commonly used statistics, such as means or medians, because these would not show the real picture. Instead we perform the following procedure beforehands:

- The experimental runs are distributed into the bins based on their binary logarithm: the i-th bin, $i \geq 0$, contains the runs with the running time between 2^i, inclusively, and 2^{i+1}, exclusively. Let C_i be the number of runs in the i-th bin.
- The first bin p is looked such that $C_{p+1} < C_p$. This is the top of the first peak.
- The first bin $q > p$ is looked, such that either $C_q < C_{q+1}$ or $C_q = 0$. This is the end of the first peak. In the first case, or in the second case when some of the higher bins additionally have nonzero counts, all the bins beginning with the q-th are considered to be the second peak.
- If two peaks are present, they are processed separately in the subsequent analysis.

In particular, the light and dark colors in Fig. 2, when present, illustrate the first and the second peak found by the procedure above. Note that, in general, the correctness of such a procedure is at least questionable, however, given the nature of the peaks (small polynomial vs exponential runtimes) and the fact that the first peak is quite well-concentrated, this simple procedure yields good enough results.

With these precautions, we now present the summary of the experimental results for the bipartite graph in Fig. 3. The plots indeed show some clear trends[1]: while the "hard" peaks of the non-heavy-tailed algorithms, show the behavior that is at least exponential in problem size, their "easy" peaks, and all heavy-tailed algorithms are clearly polynomial.

The good behavior of the heavy-tailed algorithms can be formulated and proven as follows.

Theorem 1. *The expected time that the heavy-tailed algorithms require to move from the local optimum to the global one on a bipartite graph with v vertices, v is odd, having halves of $\lfloor v/2 \rfloor$ and $\lceil v/2 \rceil$ vertices, is*

- $O(v^\beta)$ *for the* $(1+1)$ *FEA;*
- $O(v^{\beta_p})$ *for the* $(1+(\lambda,\lambda))$ *GA$_{H3}$ with* $p_\lambda > 2$*;*
- $O(v^{\beta_\lambda})$ *for the* $(1+(\lambda,\lambda))$ *GA$_{H1}$;*

Proof: We pessimistically assume that these algorithms stay in the local optimum until they jump directly to the global one. The $(1+1)$ FEA flips all the bits on each iteration with the probability $\Theta(v^{-\beta})$, and as its iteration requires only one fitness evaluation, the expected time until the global optimum is hit is $O(v^\beta)$.

The $(1+(\lambda,\lambda))$ GA$_{\text{H3}}$, similarly, flips all the bits in the mutants on each iteration with the probability $\Theta(v^{-\beta_p})$. As for $\beta_\lambda > 2$ the expected population size is $\Theta(1)$, the global optimum is hit in expected $O(v^{\beta_p})$ fitness evaluations.

Finally, the $(1+(\lambda,\lambda))$ GA$_{\text{H1}}$ samples $\lambda = v$ with probability $\Theta(v^{-\beta_\lambda})$. In this case, $p \leftarrow 1$, and all the mutants are obtained by flipping all the bits. Despite the fact that the population size is large at this moment, it has been $\Theta(1)$ in expectation in preceding iterations. The last iteration has $\Theta(v)$ fitness evaluations, which is subsumed into the expected time of $O(v^{\beta_\lambda})$. ∎

For the chosen parameter values, the $(1+1)$ FEA and the $(1+(\lambda,\lambda))$ GA$_{\text{H3}}$ complete the transition in $O(v^{1.1})$ expected fitness evaluations, and the $(1+(\lambda,\lambda))$ GA$_{\text{H1}}$ in $O(v^{2.1})$ expected fitness evaluations. Since the time of reaching the local optimum is smaller, but the problem size is not large enough to distinguish $O(v^{1.1})$ from $\Theta(v \log v)$, only the $(1+(\lambda,\lambda))$ GA$_{\text{H1}}$ eventually shows a bimodal behavior starting from $v = 23$.

Among the non-heavy-tailed algorithms, the transition time of the $(1+1)$ EA seems to be $\Theta(v^v)$ as it needs to invert each bit. The employed resampling version of the $(1+1)$ EA flips all the bits with probability $v^v \cdot (1-(1-1/v)^v)$, hence for $v = 11$ the corresponding time should be roughly $1.86 \cdot 10^{11}$. However, our experiments show times of roughly 10^9 to 10^{10}, so it seems that this algorithm in fact does something better. We formulate and prove a bound which is more precise.

Theorem 2. *The expected time for the* $(1+1)$ *EA configured for the probabilities of:*

- p_1 *of flipping exactly one bit;*
- p_v *of flipping all v bits;*
- $p_w = o(v \cdot p_1)$ *of flipping exactly* $v-1$ *bits;*

[1]Two apparent glitches in the plots are due to the (rare) events when the multiple peaks were not discovered correctly, in which case the median moves to the easy region and the interquartile ranges visually shrink.

that is required to move from the local optimum to the global one on a bipartite graph with v vertices, v is odd, having halves of $\lfloor v/2 \rfloor$ and $\lceil v/2 \rceil$ vertices, is

$$\frac{1}{p_v + \frac{v+1}{2v} p_w} \cdot (1 \pm o(1)).$$

Proof: Note that the fitness of the cover that is different from a global optimum by adding exactly one arbitrary vertex from the larger half is the same as the fitness of the local optimum. Let us call this cover a *transitional* state.

The following moves are possible from the local optimum: either (i) all the bits are flipped to reach the global optimum, or (ii) all but one bits are flipped, such that the remaining bit is in the large half, to move to a transitional state. Similarly, the following moves are possible from a transitional state: either (i) the one missing bit is flipped to reach the global optimum, (ii) all bits, except for the one missing bit, are flipped to return to the local optimum, (iii) some pairs of bits may be flipped to move to another transitional state. One can see that no other moves are possible, since they decrease the fitness.

Let T_v be the expected time required to get to the global optimum from the local one, and T_{v-1} be the time from any transitional state. The cases above can be written as follows:

$$T_v = 1 + p_v \cdot 0 + p_w \frac{v+1}{2v} T_{v-1} + \left(1 - p_v - p_w \frac{v+1}{2v}\right) T_v$$

$$T_{v-1} = 1 + \frac{p_1}{v} \cdot 0 + \frac{p_w}{v} T_v + \left(1 - \frac{p_1}{v} - \frac{p_w}{v}\right) T_{v-1}$$

This is solved as a system of linear equations to

$$T_v = \frac{v\left(2 + \frac{p_w(v+1)}{p_1+p_w}\right)}{2vp_v + p_w(v+1) - \frac{p_w^2(v+1)}{p_1+p_w}}$$

and simplified by applying the asymptotical relations from the theorem statement to get the desired form. ∎

By setting $p_v = 1/11^{11}/(1-(1-1/11)^{11})$ and $p_w = 11 \cdot (1-1/11)/11^{10}/(1-(1-1/11)^{11})$ we get the expected transition time of $3.04 \cdot 10^9$ evaluations for the $(1+1)$ EA with resampling on $n = 11$, which agrees well with the experimental results.

Finally we analyse the non-heavy-tailed versions of the $(1+(\lambda,\lambda))$ GA. We shall note that the only way to eventually obtain a global optimum when residing at the local optimum is to flip all the bits (except for maybe one, as the previous analysis shows) in the mutation phase. The probability of getting all the bits flipped in the mutation phase times the probability of accepting all these bits in the crossover phase is again of order n^{-n}, that is, the canonical version of the $(1+(\lambda,\lambda))$ GA may get the global optimum as a mutant, but then is very likely to discard it. While this fact does not actually harm with the commonly used notion of first hitting time, it may drastically decrease the running time in many situations, including the one occurring in the previous analysis. Hence considering a good mutant offspring seems to be a good idea in the general setting.

Due to larger mutation rates in the mutation phase, the $(1+(\lambda,\lambda))$ GA may flip all the bits, as well as all the bits except one of the bit in the larger half, much sooner than the $(1+1)$ EA. We formalize and prove this statement in the following theorem.

Theorem 3. *The expected time for the $(1+(\lambda,\lambda))$ GA configured as follows:*

- *$1 < \lambda < v/3$, $p \leftarrow \lambda/v$, $c \leftarrow 1/\lambda$;*
- *λ is rounded down to determine the population size;*
- *the number of bits to flip in mutants is resampled until positive;*
- *the best mutant offspring is taken to the selection stage;*

that is required to move from the local optimum to the global one on a bipartite graph with v vertices, v is odd, having halves of $\lfloor v/2 \rfloor$ and $\lceil v/2 \rceil$ vertices, is

$$2\lfloor\lambda\rfloor \cdot \frac{\left(\frac{v}{\lambda}\right)^v - \left(\frac{v}{\lambda} - 1\right)^v}{1 + \frac{v+1}{2\lambda}(v-\lambda)} \cdot (1 \pm o(1)).$$

Proof: We use the same idea as in Theorem 2, taking into account that

$$p_v = \frac{\left(\frac{\lambda}{v}\right)^v}{1 - \left(1 - \frac{\lambda}{v}\right)^v}, \quad p_w = \frac{\left(1 - \frac{\lambda}{v}\right)\left(\frac{\lambda}{v}\right)^{v-1} \cdot v}{1 - \left(1 - \frac{\lambda}{v}\right)^v}$$

and that the cost of one iteration is 2λ fitness evaluations. By substituting these values into the result of Theorem 2, one gets the desired result.

The only remaining thing is to prove that the probability of getting from the transitional state, in terms of Theorem 2, to the global optimum is indeed high enough. In fact, a very pessimistic way to estimate this probability already gives a satisfying result. The probability of generating the global optimum in a single application of mutation is

$$p_1' = \frac{\left(1 - \frac{\lambda}{v}\right)^{v-1} \cdot \frac{\lambda}{v}}{1 - \left(1 - \frac{\lambda}{v}\right)^v},$$

so we can estimate how large it is compared to p_w:

$$\frac{p_w \cdot v}{p_1'} = \frac{\left(1 - \frac{\lambda}{v}\right)\left(\frac{\lambda}{v}\right)^{v-1} \cdot v^2}{\left(1 - \frac{\lambda}{v}\right)^{v-1} \cdot \frac{\lambda}{v}} = v^2 \cdot \left(\frac{\lambda}{v-\lambda}\right)^{v-2} = o(1)$$

for $\lambda < v/3$, hence the global optimum is reached before rolling back to the local optimum with high probability. ∎

Note that the self-adjusting versions of the $(1+(\lambda,\lambda))$ GA hit the threshold value for λ when residing in the local optimum for just $O(\log v)$ iterations, hence Theorem 3 is applicable to them assuming the maximum λ. Applying the formula from Theorem 3 to $v = 15$ and $\lambda = v/4 = 3.75$, we obtain the expected transition time of $\approx 2.54 \cdot 10^8$; to $v = 15$ and $\lambda = 2\ln(v+1) \approx 5.545$ we obtain the time of $\approx 2.07 \cdot 10^6$; to $v = 19$ and $\lambda = 2\ln(v+1) \approx 5.99$ we obtain the time of $\approx 1.47 \cdot 10^9$. These values agree pretty well with the experimental data from Fig. 3.

Note that larger λ are better, and the only reason why the $(1+(\lambda,\lambda))$ GA_{N4} seems worse than the $(1+(\lambda,\lambda))$ GA_{LN} is that $v/4$ is smaller than $2\ln(v+1)$ for small v. The plot slopes suggest that this relation changes for large enough v.

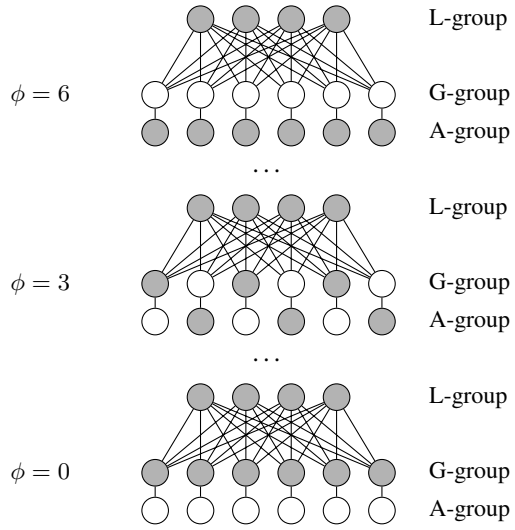

Fig. 4. Local optima network in the PS-graph. Covers are shown in gray.

IV. The Papadimitrou-Steiglitz Graph

Now we switch to another instance class, the Papadimitrou-Steiglitz graphs, or PS-graphs. The example of such a graph is given in Fig. 1. Let $v = 3k + 4$ be the number of vertices, and k, $k + 2$ and $k + 2$ be the sizes of the groups of nodes. For brevity, we call the group of size k the L-group (for "local optimum"), the group of size $k+2$ connected to it the G-group (for "global optimim"), and the remaining group the A-group (for auxiliary vertices).

This instance class features the unique global optimum and a family of 2^{k+2} local optima that correspond to choosing the whole L-group and one of the endpoint nodes for each of the edge connecting the G-group and the A-group (see Fig. 4 for example). These local optima are connected into a single plateau by two-bit flips. We denote as ϕ the number of chosen vertices in the A-group. If $\phi = 0$, a highly likely event is to remove one of the vertices from the L-group, which results in a subsequent fast convergence to the global optimum by quickly deselecting the whole L-group: it is known that the $(1 + 1)$ EA does that in $O(v \log v)$ time [29].

In general, evolutionary algorithm tend to discover either the global optimum or one of the local optima, both choices with constant probability. In the former case, the optimization is fast. When being in the network of local optima, the $(1 + 1)$ EA typically proceeds by flipping pairs of bits at the edges connecting the G-group and the A-group, which happens at random, so the whole process is the random walk where value ϕ is typically in the vicinity of $(k + 2)/2$. Only with a very small probability, the bits are flipped in such a way that the G-group is selected, the L-group loses at lease one vertex, and the A-group becomes almost empty. Such an event is more likely as ϕ approaches zero, however, due to the strong drift towards the center, ϕ can be small very infrequently and for comparatively short periods of time. For these reasons, the $(1+1)$ EA has a $2^{\Omega(\sqrt[3]{n})}$ lower bound on its expected runtime [29, Theorem 3].

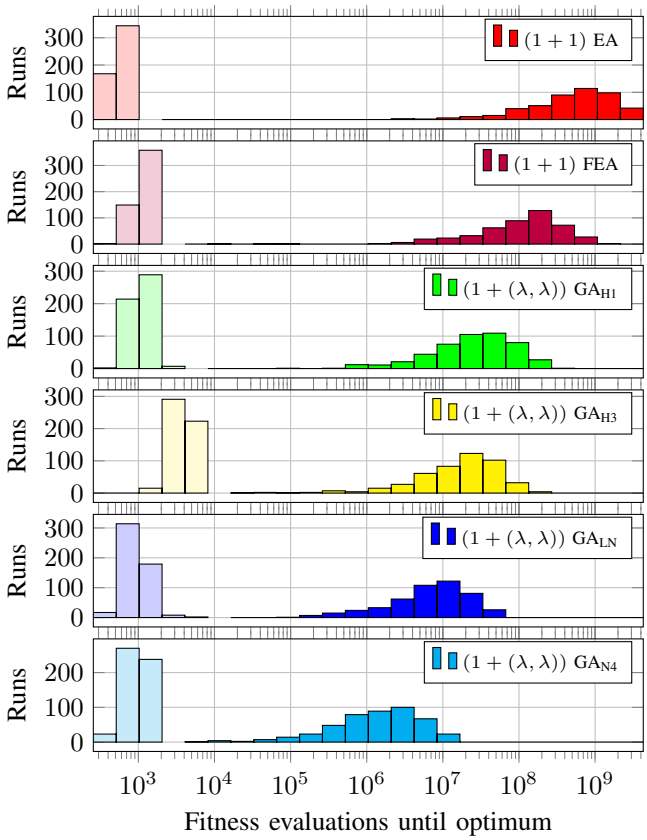

Fig. 5. Distribution histograms: PS-graphs, $v = 73$, 1000 runs

Clearly, the $(1 + 1)$ FEA cannot do radically better, as although it can flip $\Theta(n)$ bits with inversely polynomial probabilities, it still has to guess which exactly bits to flip. Otherwise, it is also doomed to random walks in the local optima network. Technically, our version of the $(1 + 1)$ FEA can flip all bits and reach the global optimum from the state of $\phi = k + 2$, however, this state is just as unlikely to visit as the state of $\phi = 0$.

Our experiments, with the same algorithmic setting and the same approach to partition the outcomes into hard and easy ones as in the previous section, reveal a pretty interesting picture though. Fig. 5 shows the histograms, which are this time bimodal for all the employed algorithms. They suggest that the $(1 + 1)$ FEA performs somewhat better than the $(1+1)$ EA, however, the $(1+(\lambda, \lambda))$ GA family does a much better job. The runtime plots are presented in Fig. 6.

Here we can see that the $(1+(\lambda, \lambda))$ GA, while still being exponential in the number of vertices in the bad case, tend to be exponentially faster than both the $(1+1)$ EA and the $(1+1)$ FEA. With regards to λ, the trends seem to be, similar to the previous section, that the greater λ the better the performance. However, this time such small runtimes as roughly $2 \cdot 10^6$ for $v = 73$ and hence $k = 23$ cannot be simply attributed to high mutation rates in the mutation phase and acceptance of the best mutant, as even with $\lambda = n/4$ flipping this many bits

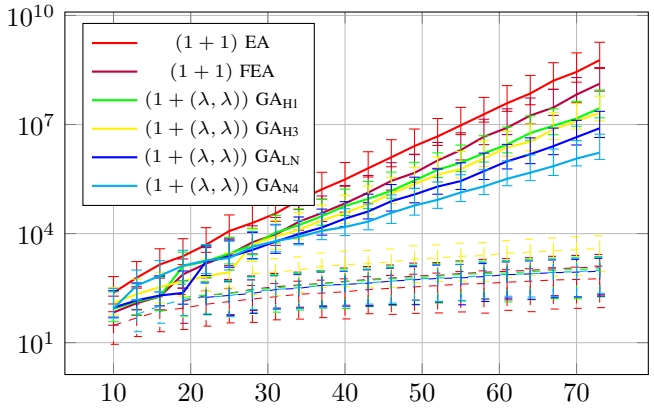

Fig. 6. Peak-median runtimes: PS-graphs, 1000 runs. Solid lines correspond to the peak with larger runtimes, dashed lines indicate the first peak when present. Medians are plotted, error bars show interquartile ranges.

correctly is still unlikely.

Instead, we conjecture that the crossover mechanism that follows selection at the mutation phase is the key for success. Unfortunately, due to the complexity of the interaction between the algorithm and the problem, this time we cannot yet be as rigorous as even in the previous section. Instead, we formulate our conjecture in not very rigorous terms, and augment it with a theorem that shows why this conjecture can be true and points at the way to prove it rigorously.

Conjecture 1. *On the Papadimitrou-Steiglitz instance class of the vertex cover problem, the $(1+(\lambda,\lambda))$ GA selects mutants in such a way that bit flips exchanging a vertex from the A-group with a matching vertex from the G-group are more likely compared to the opposite bit flips than it would happen at random.*

This happens because, with high probability, a number of vertices from the L-group is deselected in the mutants, which penalizes mutants with small number of vertices in the G-group due to the number of uncovered edges dominated by the product of numbers of missing vertices in these two groups.

Together, this phenomenon introduces an additional drift on the local optima network towards decreasing ϕ, which makes it more likely to occasionally jump towards the vicinity of the global optimum in the mutation phase.

We prove this conjecture in a specific condition of $\phi = \frac{k+2}{2}$. Under random walks, this state has a zero drift, which means that the move to either direction is equally likely. However, the $(1+(\lambda,\lambda))$ GA has a pronounced drift towards smaller values of ϕ.

Theorem 4. *On the Papadimitrou-Steiglitz instance class of the vertex cover problem, in a state defined by $\phi = \frac{k+2}{2}$, the $(1+(\lambda,\lambda))$ GA with $\lambda = \omega(\log v)$ transitions to a state with $\phi' < \frac{k+2}{2}$ with probability strictly greater than $1/2$ conditioned on the transition happening.*

Proof: The state with $\phi = \frac{k+2}{2}$ has an equal number of zero and one bits in the G-group, hence the event E_0 of flipping more zero bits than one bits in this group has the same probability of the event E_1 of flipping more one bits than zero bits. Obviously, $P[E_0] = P[E_1] > 0$.

Consider a random variable X_u that corresponds to the mutant having the best fitness conditioned on that u bits in the L-group are flipped in that offspring. All mutants with the same u have fitness $uv \cdot |g| + O(\lambda v)$, where g is the number of zero bits in the G-group, where the first addend corresponds to uncovered edges between the L-group and the G-group, and the second one corresponds to uncovered edges between the G-group and the A-group, as well as to the change in the number of selected vertices. Since, due to Chernoff bounds, $u = \frac{\lambda k}{v} \pm c_1\sqrt{\frac{\lambda k}{v}\log\frac{\lambda k}{v}}$ and $g = \frac{k+2}{2} \pm c_2\sqrt{\frac{\lambda k}{v}\log\frac{\lambda k}{v}}$ with probability $1 - O(v^{-c_3})$ for suitable constants c_1, c_2, c_3, the first addend is asymptotically larger than the second. Hence, whenever two or more mutants are generated with the same u, they are effectively compared by g, where smaller g have an advantage.

For this reason, $\Pr[X_u$ flipped more zero than one bits$] > \frac{1}{2}$ conditioned on that there are two or more mutants with the same u. Given that mutants are created independently, for each u in the range above the probability of having two or more mutants is strictly nonzero. Despite mutants with different u may compare in arbitrary way depending on how their u and g relate, for each u the better mutant is selected with nonzero probability, and with the remaining probability the g value of that mutant is distributed as if no selection happens. Hence, the selection towards smaller $\phi' = g$ happens with nonzero probability, and otherwise no preference is made.

We finish by noting that the transition happens only if pairs of matching bits in the G-group and the A-group are flipped, while the selection process happens only on the G-group, and some of the bits flipped in the right direction may not have the matching bit flipped. However, bit flipping in the A-group does not have any negative effect on the selection process[2]. Hence, we may consider it as an independent random process without worsening the result. In this case, each changed bit in the G-group has a probability of λ/v to be augmented with a matching bit flip in the A-group. The crossover phase translates an increased probability of flipping zero bits in the G-group to an increased probability of constructing a crossover offspring representing a valid cover with fitness $2k+4$ and smaller ϕ'. Since all crossover offspring with fitness $2k+4$ have the same chance to become the next parent, the result also has the increased probability to have the smaller ϕ', which concludes the proof. ∎

We may also note that, due to the effective range for u induced by Chernoff bounds, which has the size $O(\sqrt{\frac{\lambda k}{v}\log\frac{\lambda k}{v}})$ with overwhelming probability, and recalling that $v = 3k+4$, the population of λ mutants hits at least one of the possible u value at least twice due to the pigeonhole principle. This means that selection actually happens in the mutant population

[2]In fact, the impact of the bits from the A-group biases slightly towards deselection of these bits, which makes moves to larger ϕ even more complicated.

TABLE I
SUMMARY OF EXPERIMENTS ON RANDOM GRAPHS

$(1+1)$ EA vs $(1+1)$ FEA	$(1+1)$ EA vs $(1+(\lambda,\lambda))$ GA_{LN}	$(1+1)$ FEA vs $(1+(\lambda,\lambda))$ GA_{LN}	Count
$<$	$<$	$<$	23
$<$	$<$	$\approx$	159
$<$	$<$	$>$	119
$<$	$\approx$	$\approx$	69
$<$	$\approx$	$>$	88
$<$	$>$	$>$	21
$\approx$	$<$	$<$	1
$\approx$	$\approx$	$\approx$	12
$\approx$	$\approx$	$>$	2
$\approx$	$>$	$>$	31
$>$	$<$	$<$	2
$>$	$\approx$	$<$	4
$>$	$\approx$	$\approx$	1
$>$	$>$	$<$	9
$>$	$>$	$\approx$	88
$>$	$>$	$>$	71
$< / \approx / >$ 479 / 46 / 175	$< / \approx / >$ 304 / 176 / 220	$< / \approx / >$ 39 / 329 / 332	

for at least one of the values of u with the same overwhelming probability.

V. Random Graphs

To investigate the picture outside of the particular instance classes, we considered generating random graphs, finding their minimum vertex cover and running the algorithms mentioned above until they find the optimum. Namely, we consider the number of vertices $v \in [10..16]$, the number of edges is $e = \min\{v(v+1)/2, 5 \cdot v\}$, and the edges are chosen uniformly at random. For each problem size v, we generated 100 instances, and for each instance each algorithm was run 1000 times.

In fact, such a small problem size was chosen, as the $(1+1)$ EA found some of the larger instances so complicated that it would apparently take it several days to finish. This aligns well with the worst-case bounds of order $n^{\text{poly}(n)}$. Other algorithms, however, did not show such performance degradation, although exponential runtimes were still possible.

We chose the $(1+1)$ EA, the $(1+1)$ FEA and the $(1+(\lambda,\lambda))$ GA_{LN} as the representative algorithms to compare. The whole comparison is performed on the hard peaks, extracted as mentioned in Section III, taking the whole set of runs if the peaks could not be separated. First, we validated that the $(1+1)$ FEA and the $(1+(\lambda,\lambda))$ GA_{LN} are never significantly worse than the $(1+1)$ EA: our results do not contain a single occasion when the worst runtime of the former algorithms exceeds the worst runtime of the $(1+1)$ EA by five times or more. This essentially means that other algorithms, including even the $(1+(\lambda,\lambda))$ GA_{LN}, do not stuck in the cases $(1+1)$ EA finds easy.

Next, for each instance we performed the Wilcoxon rank sum test [42] for each pair of these three algorithms. Note that, based on the insights from the previous sections, this median-based comparison makes sense for the extracted peaks, but it does not make sense to be run on the unprocessed data, since the median is typically in the easy range, and may spuriously oscillate. We say that algorithm A is more efficient than algorithm B (which maps to $A < B$) if the corresponding one-sided rank sum test produces a p-value less than 10^{-3}. We don't perform any post-hoc correction since our aim is to roughly classify the instances by how the results of the algorithms are related, and not to find an outlier to be declared statistically significant. Hence, based on the results of our experiments, each instance is annotated by three comparison outcomes, each having three possible values ("significantly smaller", "significantly greater" or "similar").

Table I presents the results of running this statistical comparison. Based on pairwise comparisons, one can see that the $(1+1)$ EA is not worse than others in 67.5% of cases, whereas the $(1+(\lambda,\lambda))$ GA_{LN} is second (54.7%) and the $(1+1)$ FEA is the last (16.7%). The $(1+1)$ EA wins the $(1+1)$ FEA 2.7 times more often than it loses, which suggests that many instances can be solved easily with small mutation rates. When comparing the $(1+1)$ FEA with the $(1+(\lambda,\lambda))$ GA_{LN}, both of which are capable of long jumps, one can see that the latter wins much more often (47%) and loses very rarely (6%).

With some caution, based on these results we may conjecture that the effects from the crossover, as opposite to just employing higher mutation rates with heavy-tailed mutation, have a positive effect quite frequently, even on such small problem sizes. A stronger conjecture would be that the just-discovered effect of adding beneficial drift to the plateaus of local optima, apparently abundant in the vertex cover problem, is responsible for a fair share of these benefits — however, more investigation is definitely needed.

VI. Conclusion

Taking the vertex cover problem as an example, we have shown, though not as rigorously as one may desire, that the crossover mechanism employed by the $(1+(\lambda,\lambda))$ genetic algorithm may give additional benefit by introducing drift onto the fitness-flat local optima networks towards regions where improvement is more likely. This is not the only mechanism that helps in such conditions, as increased mutation rates coupled with possible acceptance of the best mutant also play a significant role in the good performance of this algorithm, however, it is probably the most surprising one.

As of now, the size of this effect is not quantified theoretically — only with experiments, which, however, indicate that it may be large enough to be worth investigating in detail. This is complicated by the necessity of tracking the probability to reach the global optimum from each of the local optima forming a network with great precision. We hope that suitable theoretical tools, such as drift theorems [43], [44] and plateau analysis [45] will eventually make it possible. In turn, this will give more recipes for how one may improve evolutionary algorithms by a proper introduction of crossover.

Acknowledgments

This work was supported by the Analytical Center for the Government of the Russian Federation (IGK 000000D730321P5Q0002), agreement No. 70-2021-00141.

REFERENCES

[1] A. Auger and B. Doerr, *Theory of Randomized Search Heuristics: Foundations and Recent Developments*. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 2011.

[2] B. Doerr and F. Neumann, Eds., *Theory of Evolutionary Computation—Recent Developments in Discrete Optimization*. Springer, 2020.

[3] F. Neumann and I. Wegener, "Minimum spanning trees made easier via multi-objective optimization," *Natural Computing*, vol. 5, no. 3, pp. 305–319, 2006.

[4] ——, "Randomized local search, evolutionary algorithms, and the minimum spanning tree problem," *Theoretical Computer Science*, vol. 378, no. 1, pp. 32–40, 2007.

[5] V. Roostapour, J. Bossek, and F. Neumann, "Runtime analysis of evolutionary algorithms with biased mutation for the multi-objective minimum spanning tree problem," in *Proceedings of Genetic and Evolutionary Computation Conference*, 2020, pp. 551–559.

[6] S. Nallaperuma, F. Neumann, and D. Sudholt, "A fixed budget analysis of randomized search heuristics for the traveling salesperson problem," in *Proceedings of Genetic and Evolutionary Computation Conference*, 2014, pp. 807–814.

[7] S. Nallaperuma, M. Wagner, F. Neumann, B. Bischl, O. Mersmann, and H. Trautmann, "A feature-based comparison of local search and the Christofides algorithm for the travelling salesperson problem," in *Proceedings of Foundations of Genetic Algorithms XII*, 2013, pp. 147–160.

[8] O. Mersmann, B. Bischl, J. Bossek, H. Trautmann, M. Wagner, and F. Neumann, *Local Search and the Traveling Salesman Problem: A Feature-Based Characterization of Problem Hardness*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 115–129.

[9] D. S. Sanches, D. Whitley, and R. Tinós, "Improving an exact solver for the traveling salesman problem using partition crossover," in *Proceedings of Genetic and Evolutionary Computation Conference*, 2017, pp. 337–344.

[10] L. D. Whitley, F. Chicano, G. Ochoa, A. M. Sutton, and R. Tinós, "Next generation genetic algorithms," in *Proceedings of Genetic and Evolutionary Computation Conference Companion*, 2019, pp. 1113–1136.

[11] B. Doerr, D. Johannsen, T. Kötzing, F. Neumann, and M. Theile, "More effective crossover operators for the all-pairs shortest path problem," *Theoretical Computer Science*, vol. 471, pp. 12–26, 2013.

[12] B. Doerr, E. Happ, and C. Klein, "Tight analysis of the (1+1)-EA for the single source shortest path problem," *Evolutionary Computation*, vol. 19, no. 4, pp. 673–691, 2011.

[13] B. Doerr, A. R. Hota, and T. Kötzing, "Ants easily solve stochastic shortest path problems," in *Proceedings of Genetic and Evolutionary Computation Conference*, 2012, pp. 17–24.

[14] R. Karp, "Reducibility among combinatorial problems," in *50 Years of Integer Programming 1958–2008*, 2010, pp. 219–241.

[15] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. New York: Dover, 1998.

[16] S. Khuri and T. Bäck, "An evolutionary heuristic for the minimum vertex cover problem," in *Proceedings of KI-94 Workshop on Genetic Algorithms Within Framework of Evolutionary Computation*, 1994, pp. 86–90.

[17] I. K. Evans, "Evolutionary algorithms for vertex cover," in *Proceedings of International Conference on Evolutionary Programming*, ser. Lecture Notes in Computer Science, 1998, no. 1447, pp. 377–386.

[18] T. Friedrich, J. He, N. Hebbinghaus, F. Neumann, and C. Witt, "Analyses of simple hybrid algorithms for the vertex cover problem," *Evolutionary Computation*, vol. 17, no. 1, pp. 3–19, 2009.

[19] B. Doerr, C. Doerr, and F. Ebel, "From black-box complexity to designing new genetic algorithms," *Theoretical Computer Science*, vol. 567, pp. 87–104, 2015.

[20] B. Doerr and C. Doerr, "Optimal static and self-adjusting parameter choices for the $(1 + (\lambda, \lambda))$ genetic algorithm," *Algorithmica*, vol. 80, no. 5, pp. 1658–1709, 2018.

[21] D. Antipov and B. Doerr, "Runtime analysis of a heavy-tailed $(1 + (\lambda, \lambda))$ genetic algorithm on jump functions," in *Parallel Problem Solving from Nature – PPSN XVI*, ser. Lecture Notes in Computer Science, 2020, no. 12270, pp. 545–559.

[22] B. Doerr, H. P. Le, R. Makhmara, and T. D. Nguyen, "Fast genetic algorithms," in *Proceedings of Genetic and Evolutionary Computation Conference*, 2017, pp. 777–784.

[23] S. Droste, T. Jansen, and I. Wegener, "On the analysis of the (1+1) evolutionary algorithm," *Theoretical Computer Science*, vol. 276, no. 1-2, pp. 51–81, 2002.

[24] E. Carvalho Pinto and C. Doerr. (2018) Towards a more practice-aware runtime analysis of evolutionary algorithms. [Online]. Available: https://arxiv.org/abs/1812.00493

[25] ——, "A simple proof for the usefulness of crossover in black-box optimization," in *Parallel Problem Solving from Nature – PPSN XV, Vol. 2*, ser. Lecture Notes in Computer Science, 2018, no. 11102, pp. 29–41.

[26] J. Rapin and O. Teytaud. (2018) Nevergrad - A gradient-free optimization platform. [Online]. Available: https://GitHub.com/FacebookResearch/Nevergrad

[27] D. Corus, P. S. Oliveto, and D. Yazdani, "Fast immune system-inspired hypermutation operators for combinatorial optimization," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 5, pp. 956–970, 2021.

[28] S. Khot, "On the power of unique 2-prover 1-round games," in *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, 2002, pp. 767–775.

[29] P. S. Oliveto, J. He, and X. Yao, "Analysis of the (1+1)-EA for finding approximate solutions to vertex cover problems," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 1006–1029, 2009.

[30] J. He, X. Yao, and J. Li, "A comparative study of three evolutionary algorithms incorporating different amounts of domain knowledge for node covering problem," *IEEE Transactions on Systems, Man and Cybernetics Part C*, vol. 35, no. 2, pp. 266–271, 2005.

[31] P. S. Oliveto, J. He, and X. Yao, "Analysis of population-based evolutionary algorithms for the vertex cover problem," in *IEEE Congress on Evolutionary Computation*, 2008, pp. 1563–1570.

[32] T. Friedrich, J. He, N. Hebbinghaus, F. Neumann, and C. Witt, "Approximating covering problems by randomized search heuristics using multi-objective models," *Evolutionary Computation*, vol. 18, no. 4, pp. 617–633, 2010.

[33] M. Pourhassan, F. Shi, and F. Neumann, "Parameterized analysis of multiobjective evolutionary algorithms and the weighted vertex cover problem," *Evolutionary Computation*, vol. 27, no. 4, pp. 559–575, 2019.

[34] M. Pourhassan, W. Gao, and F. Neumann, "Maintaining 2-approximations for the dynamic vertex cover problem using evolutionary algorithms," in *Proceedings of Genetic and Evolutionary Computation Conference*, 2015, pp. 903–910.

[35] M. Pourhassan, T. Friedrich, and F. Neumann, "On the use of the dual formulation for minimum weighted vertex cover in evolutionary algorithms," in *Foundation of Genetic Algorithms*, 2017, pp. 37–44.

[36] S. Kratsch and F. Neumann, "Fixed-parameter evolutionary algorithms and the vertex cover problem," *Algorithmica*, vol. 65, no. 4, pp. 754–771, 2013.

[37] B. Goldman and W. Punch, "Parameter-less population pyramid," in *Proceedings of Genetic and Evolutionary Computation Conference*, 2014, pp. 785–792.

[38] A. Gandomi and B. Goldman, "Parameter-less population pyramid for large-scale tower optimization," *Expert Systems with Applications*, vol. 96, pp. 175–184, 2018.

[39] M. Buzdalov and B. Doerr, "Runtime analysis of the $(1+(\lambda, \lambda))$ genetic algorithm on random satisfiable 3-CNF formulas," in *Proceedings of Genetic and Evolutionary Computation Conference*, 2017, pp. 1343–1350.

[40] D. Antipov, M. Buzdalov, and B. Doerr, "Fast mutation in crossover-based algorithms," in *Proceedings of Genetic and Evolutionary Computation Conference*. ACM, 2020, pp. 1268–1276.

[41] ——, "Lazy parameter tuning and control: choosing all parameters randomly from a power-law distribution," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2021, pp. 1115–1123.

[42] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[43] B. Doerr, D. Johannsen, and C. Winzen, "Multiplicative drift analysis," *Algorithmica*, vol. 64, no. 4, pp. 673–697, 2012.

[44] T. Kötzing and M. Krejca, "First-hitting times under drift," *Theoretical Computer Science*, vol. 796, pp. 51–69, 2019.

[45] D. Antipov and B. Doerr, "Precise runtime analysis for plateaus," in *Parallel Problem Solving from Nature – PPSN XV*, ser. Lecture Notes in Computer Science, 2018, no. 11102, pp. 117–128.