Abstract—A fitness assignment process transforms the features (such as the objective value) of a candidate solution to a scalar fitness, which then is the basis for selection. Under frequency fitness assignment (FFA), the fitness corresponding to an objective value is its encounter frequency in selection steps and is subject to minimization. FFA creates algorithms that are not biased toward better solutions and are invariant under all injective transformations of the objective function value. We investigate the impact of FFA on the performance of two theory inspired, state-of-the-art evolutionary algorithms, the Greedy \((2 + 1)\) GA and the self-adjusting \((1 + \lambda, \lambda)\) GA. FFA improves their performance significantly on some problems that are hard for them. In our experiments, one FFA-based algorithm exhibited mean runtimes that appear to be polynomial on the theory-based benchmark problems in our study, including traps, jumps, and plateaus. We propose two hybrid approaches that use both direct and FFA-based optimization and find that they perform well. All FFA-based algorithms also perform better on satisfiability problems than any of the pure algorithm variants.

Index Terms—Evolutionary algorithm (EA), FEA, frequency fitness assignment (FFA), Ising problems, jump problems, linear harmonic functions, MaxSat problem, N-queens problem, OneMax, Plateau problems, satisfiability, Trap function, TwoMax, W-model benchmark.

I. INTRODUCTION

Fitness assignment is a component of many evolutionary algorithms (EAs). It transforms the features of candidate solutions, such as their objective value(s), to scalar values which are then the basis for selection. Usually, the goal is to maintain diversity in the population in order to avoid getting stuck at local optima. Under frequency fitness assignment (FFA), the fitness of a candidate solution is the absolute encounter frequency of its objective value so far during the optimization process [1]. Being subject to minimization, FFA drives the search away from already-discovered objective values and toward solutions with new qualities.

FFA breaks with one of the most fundamental concepts of heuristic optimization: FFA-based algorithms are not biased toward better solutions [2], i.e., they do not prefer better solutions over worse ones. They also are invariant under all injective transformations of the objective function value, which is the strongest invariance property of any nontrivial single-objective optimization algorithm [3]. Only random sampling, random walks, and exhaustive enumeration have similar properties and neither of them is considered to be an efficient optimization method.

One would expect that this comes at a significant performance penalty. Yet, FFA performed well in Genetic Programming tasks with their often rugged, deceptive, and highly epistatic landscapes [1], [4] and on a benchmark problem simulating such landscapes [5]. While the \((1+1)\) EA has exponential expected runtime on problems, such as Jump, TwoMax, and Trap, the \((1+1)\) FEA, the same algorithm but using FFA, exhibits mean runtimes that appear to be polynomial in experiments and also solves MaxSat problems much faster than the \((1+1)\) EA [3].

These interesting properties and results lead to the question whether FFA could also benefit state-of-the-art black-box metaheuristics. In this article, we investigate the behavior of FFA when plugged into two such algorithms, the Greedy \((2+1)\) GA (GGA) [6], [7] and the self-adjusting \((1+\lambda,\lambda)\) GA (SAGA) [8], [9]. GGA is the first GA with a provable guarantee to be significantly faster than any mutation-only GA and SAGA is the first GA with linear expected runtime, both on the well-known OneMax problem (see Section IV-A).

We conduct a series of experiments applying both EAs (in their efficient modified form from [10]) with and without FFA to a wide range on well-known benchmark problems from discrete optimization theory [11]. We observe that FFA leads to a performance decrease on problems that the algorithms already can solve well. On some of the problems that they cannot solve efficiently, FFA can provide a significant performance

1We stated in [3] that FFA makes algorithms invariant under bijections of the objective function value. Yet, surjectivity is not needed, injectivity is already sufficient.

Thomas Weise, Zhize Wu, Xinlu Li, Yan Chen, and Jörg Lässig
improvement, e.g., reducing exponential to empirical polynomial mean runtime. To investigate whether FFA can be combined with traditional optimization to reap the best performance in all scenarios, we develop two hybrid algorithms, the EAFEA, which executes the steps of the $(1+1)$ EA and $(1+1)$ FEA in an alternating fashion, and the SAHGA, a SAGA variant that can switch between direct and FFA-based optimization.

The eight core contributions of this article are as follows.

1) We develop FFA-based variants of two state-of-the-art EAs for discrete optimization.

2) We develop two hybrid algorithms that use both the objective function directly and FFA.

3) We investigate the performance of these algorithms in a very large-scale study on eleven theory-inspired benchmarks, some of which invoke exponential runtime of the base algorithms, and the satisfiability problem. We conduct more than 56 million runs, consuming more than 150 processor years, to ensure that our results are statistically sound and rigorous.

4) When plugged into a GGA or $(1+1)$ EA, FFA can provide runtime that appears to be polynomial in our experiments on Trap, TwoMax, and Jump, where these algorithms otherwise require exponential expected runtime. It does not improve their exponential expected runtime on Plateau problems.

5) The SAFGA, i.e., the SAGA with FFA, as well as the hybrid SAHGA have empirical polynomial runtime on all of the investigated theory-inspired benchmark problems, including Plateaus.

6) On the MaxSat problem, every FFA-based algorithm outperforms all of the pure methods by a huge margin.

7) We confirm that our hybrid algorithms exhibit good performance.

8) With this, we have substantiated the high attractiveness and also practical utility of FFA. FFA is the only approach that allows for efficient optimization without bias for better solutions. This makes it also highly interesting from a theoretical point of view.

All experimental results, all source codes, as well as all the code that creates the diagrams and tables from the results are available in an immutable archive at doi:10.5281/zenodo.5567725.

The remainder of this article is structured as follows: Section II discusses FFA and in Section III, we introduce the algorithms studied. The experimental setup and results are presented in Sections IV and V summarizes our findings and plans for future work.

II. FREQUENCY FITNESS ASSIGNMENT

The objective function $f : \mathcal{X} \rightarrow \mathcal{Y}$ maps each candidate solution $x \in \mathcal{X}$ to an objective value $y = f(x)$. Optimization algorithms normally use the objective values to decide which solution should be further explored. To improve the diversity in populations of EAs, fitness assignment processes, such as sharing and niching [12] combine the objective values and density information into a fitness value. The EA then uses the fitness instead of the objective value as a basis for selection.

FFA is a fitness assignment process suitable for problems where the number $|\mathcal{Y}|$ of possible objective values is not too high. This is true for many important hard problems, such as MaxSat, set covering, job shop scheduling, and vertex covering, for many well-studied benchmark problems in discrete optimization, as well as for several machine learning tasks, such as classification [1] or object detection [13].

Under FFA, the fitness of a candidate solution $x$ is the encounter frequency $H[y]$ of its objective value $y = f(x)$ so far during the search. The selection step of a metaheuristic chooses which candidate solutions to retain and which to remove from the population. Before selection, we increment the fitness corresponding to the objective value of each solution in the population by one. If one objective value occurs multiple times, its fitness is incremented multiple times as well. The fitness is subject to minimization and, as the sole criterion used for selection, replaces the objective value in all comparisons.

Whenever a solution $x_n$ with a previously unseen objective value $y_n = f(x_n)$ is encountered, it will receive the best possible fitness $H[y_n] = 1$ since its fitness is initially zero and then incremented to one before selection. Thus, any new best-so-far solution will also receive the best fitness under FFA when first encountered. FFA turns static problems into dynamic ones: Every time a solution with the same objective value $y_n$ is encountered or retained in the population, $H[y_n]$ is increased. This makes them less and less attractive and the search will eventually depart from them.

Counting the occurrences of the results of $f(x)$ or of $g(f(x))$ yields the same algorithm behavior as long as $g$ maps each unique value of $f$ to a unique value. Optimization algorithms basing their decisions on the frequency fitness alone therefore are invariant under any injective function $g$ applied to the objective values ($g$ of $f$) [3], including scaling, shifting, permutations, and even encryption. For the same reason, they are not biased toward better solutions. They will accept a new solution $x_n$ as long as its objective value has an encounter frequency not higher than the one of the current solution $x_c$, regardless whether it is better or worse. That such a scheme can solve problems efficiently is surprising, since preferring better solutions over worse ones (with a certain probability) is maybe the most central concept in optimization.

The related works on FFA are discussed in detail in our recent paper [3]. Invariance was one major design principle of Information-Geometric Optimization [14], which is maybe the closest related work in terms of invariance properties. Under FFA, an algorithm may oscillate between pursuing increasing and decreasing objective values. The idea of inverse tournament selection [15], where the loser is selected, is interesting in this context. FFA might also be considered as a Quality-Diversity approach [16]–[18], such as Novelty Search [19] or Surprise Search [20]. However, different from all of these methods, FFA itself is not an algorithm but a module that can be plugged into existing algorithms.

The search will, of course, remember the best-so-far solution based on the objective value in a special variable to determine its final result, but this variable is not used in any selection decision, see Fig. 1.
III. INVESTIGATED ALGORITHMS

Black-box metaheuristics working on bit strings $X = \{0, 1\}^s$ of a fixed length $s$ are a major concern for research. We integrate FFA into the basic $(1+1)$ EA and two state-of-the-art theory-inspired GAs for minimization problems. On the left-hand side of Fig. 1, we present the original algorithms as published in literature. On the right-hand side, their corresponding variants using FFA are defined. Without loss of generality, we limit our algorithm definitions to objective functions with a range $[0..UB]$ containing all integer numbers from 0 to $UB$.

We apply all algorithms in their modified efficient form defined in [10] and also use the following notation from this work: $\ell \leftarrow \text{Bin}(s, p)$ stands for sampling a number $\ell$ from the binomial distribution with $s$ trials and success probability $p$, i.e., $P[\ell = k] = \binom{s}{k} p^k (1-p)^{s-k}$. Setting $\ell \leftarrow \text{Bin}_{\geq 0}(s, p)$ means to repeat this sampling until an $\ell > 0$ is encountered. The operation $\text{mut}_s(x)$ creates a copy of $x$ with exactly $s$ bits flipped at different indexes chosen uniformly at random (u.a.r.). $\text{cross}_s(x_1, x_2)$ performs crossover by choosing, independently for every position, the value of the corresponding bit from $x_2$ with probability $c$ and from $x_1$ otherwise.

The first algorithm in our study is the $(1+1)$ EA, which starts with a random string $x_c$. In every step, it samples a new solution $x_n$ as a modified copy of $x_c$ by flipping each bit with probability $1/s$. This would correspond to drawing the number $\ell$ of bits to flip from the binomial distribution with $p = 1/s$. In the $(1+1)$ EA defined in Fig. 1(a), we instead use $\ell \leftarrow \text{Bin}_{\geq 0}(s, 1/s)$ to ensure $\ell > 0$. Thus, $x_n \neq x_c$ always holds and we avoid wasting function evaluations (FEs) [10]. If $x_n$ is at least as good as $x_c$, i.e., $f(x_n) \leq f(x_c)$, then it replaces $x_c$.

We plugged FFA into this algorithm in [3] and obtained the $(1+1)$ FEA [Fig. 1(b)]. The $(1+1)$ FEA bases all decisions influencing the course of its search on the frequency fitness table $H$. It also internally maintains a variable $y_B$ in which it stores the solution with the best-ever encountered objective value $y_B$ to be returned as end results. We have omitted this detail for the sake of brevity in all FFA-based algorithms.

The Greedy $(2+1)$ GA [6], [7] retains a population of the best two solutions $x_c$ and $x_d$ and without loss of generality, we assume that $f(x_c) \leq f(x_d)$. In each iteration, it randomly chooses two parents with the best fitness in the population with replacement to create an offspring $x_e$ via uniform crossover ($c = 0.5$). It then flips each bit in $x_e$ independently with probability $p = (1 + \sqrt{5})/(2s)$ to obtain the new solution $x_n$. If $f(x_n) \leq f(x_e)$, it will replace one of the parents: either $x_d$, if $x_d$ is worse than $x_e$, otherwise either of the two u.a.r.

The GGA [10], [21] [Fig. 1(c)] ensures that crossover is only done if $f(x_c) = f(x_d)$ and that the parents are chosen without replacement, while otherwise $x_e = x_c$. Furthermore, if $x_e \in \{x_c, x_d\}$, it is enforced that $\ell \geq 1$ bits are flipped in mutation.\(^3\)

\(^3\)We retain the mutation rate $p$ from the original Greedy $(2+1)$ GA algorithm [6], [7], which is not the optimal choice for our GGA algorithm variant on OneMaxGGA [10], [21].

The GGA was shown to be twice as fast as any mutation-only EA on OneMax [6], [7]. We plug FFA into this algorithm and obtain the GFGA specified in Fig. 1(d), which allows us to investigate the impact FFA on an algorithm where crossover—and hence, a (small) population—are provably efficient.

The $(1+\lambda)$ GA proposed in [8] maintains a single best individual $x_c$. In every step, it generates $\lambda$ offspring $x^{(i)}$ using mutation probability $p = k/s$ with $k > 1$. The best of them, $x'$, is then used as parent for another $\lambda$ offspring $x^{(i)}$ created via crossover using probability $c$. If $x_n$, the best among the offspring $x^{(i)}$, is at least as good as $x_c$, it replaces it. The self-adjusting $(1+\lambda, \lambda)$ GA [8] builds on this algorithm by adjusting the parameter $\lambda$: If $x_n$ is better than $x_c$, $\lambda$ is decreased to $\lambda/F$ and otherwise increased to $F^{\lambda/\lambda}$. $F = 1.5$ and $\lambda$ always remains in $[1..s]$. In [9] and [22], the parameters are set to $p = \lambda/s$ and $c = 1/\lambda$.

The SAGA [10] in Fig. 1(e) improves the efficiency by ensuring that mutation flips $\ell \geq 1$ bits, that crossover offspring equaling one of their parents are not evaluated, and by also letting $x'$ participate in the selection step after crossover.

The SAGA is a theory-inspired GA with linear expected runtime on OneMax [9]. In Fig. 1(f), we present the SAFGA, a variant of this algorithm applying FFA. We can thus investigate the impact of the different optimization paradigm offered by FFA on this highly efficient algorithm and its process of self-adjustment, which now is based on the frequency fitness.

As a side note: Not wasting FEs on repeatedly evaluating the same solutions saves runtime for the pure algorithm variants. This does not necessarily hold for the FFA-based variants, as evaluating the same solution again would influence the frequency fitness and we could increment the frequency counters without actually invoking the objective function. We do not investigate this approach here as it would require us to double the experimentation effort. Still, we want to at least mention that the modifications from [10] may not necessarily retain their impact under FFA.

Besides FFA-only and pure algorithm variants, we can also conceive hybrids. One could, for instance, assign the FEs to the $(1+1)$ EA and the $(1+1)$ FEA in a round robin fashion. In this case, the resulting runtime on any problem would be at most twice the runtime of the faster of the two, making it uninteresting for experiments. The EAFFA in Fig. 2(a) does this with a slight twist: If the $(1+1)$ FEA part discovers a solution equally good to the best-so-far solution of the $(1+1)$ EA part, it will overwrite it, thus “informing” the $(1+1)$ EA-part.

Another idea would be to toggle between FFA and pure optimization. This concept is implemented in our SAHGA in Fig. 2(b), which behaves exactly as the pure SAGA until one iteration after $\lambda$ reaches $s$, i.e., when basically all bits will be flipped during mutation. At this moment, the algorithm switches over to using FFA and keeps using it until the best-so-far solution improves, at which point it will toggle back to pure optimization. In SAHGA, we also added the minor improvement that only one single mutation is performed if $\lambda = s$, since then all bits are flipped and all mutation offspring are identical.

We thus cover two hybridization ideas, namely, a round-robin like and a back-and-forth switching method.
Fig. 1. Pseudo codes of six of the investigated algorithms for minimization. The left column contains the original variant, the right column the one with FFA, with differences highlighted. Note: The implementations of the FFA-based algorithm internally remember and return the candidate solution $x_B$ with the best encountered objective value $y_B$ (not fitness).

(a) (1+1) EA \[3\], \[10\], \[b\] (1 + 1) FEA \[3\]. (c) GGA \[10\], \[21\] with $p = (1 + \sqrt{5})/2s$. (d) GFGA with $p = (1 + \sqrt{5})/2s$.

(b) SAGA \[10\] with $F = 1.5$, $p = \lambda/s$, $c = 1/\lambda$.

(c) SAFGA with $F = 1.5$, $p = \lambda/s$, $c = 1/\lambda$.

IV. EXPERIMENTS

We now apply the eight algorithms to a wide selection of different benchmark problems. The objective functions are all subject to minimization. Their ranges are $Y \subseteq \{0, \ldots, UB\}$, where the upper bounds $UB$ are either linear functions of the problem scale $s$ or a small polynomial of it. On each problem instance and for each algorithm, except for the MaxSat instances, we conduct at least 71 independent runs. Except for OneMax, LeadingOnes, and $N$-Queens, we limit all runs to at most $10^{10}$ FEs in our experiments.

A. OneMax Problem

OneMax \[23\] is a unimodal optimization problem where the goal is to discover a bit string of all ones. Its minimization version of scale $s$ is defined in the following and illustrated in Fig. 3.

$$\text{OneMax}(x) = s - |x|_1$$

where $|x|_1 = \sum_{i=1}^{s} x[i]$. (1)

OneMax has a black-box complexity of $\Omega(s/\ln s)$ \[24\]. The (1 + 1) EA \[8\], \[23\] and the faster GGA \[6\]–\[8\] have
**Algorithm 1:** Hybrid algorithms using both FFA and direct optimization. (a) EAFEA. (b) SAHGA with $F = 1.5$, $p = \lambda/s$, and $c = 1/\lambda$.

Fig. 2. Hybrid algorithms using both FFA and direct optimization. (a) EAFEA. (b) SAHGA with $F = 1.5$, $p = \lambda/s$, and $c = 1/\lambda$.

Fig. 3. Illustrations of the OneMax, TwoMax, Trap, Jump, and Plateau problems for $s = 32$ and $w = 6$.

an expected runtime in $\Theta(s \ln s)$ FEs, while the SAGA achieves $O(s)$ [9], [22]. This is visible in the upper half of Fig. 4, which also shows that the FFA-based variants of the algorithms are consistently slower. The EAFEA needs about twice the time of the $(1+1)$ EA. The SAHGA performs identical to the SAGA, such that the curve of the former exactly covers the curve of the latter, meaning that the switch condition to FFA is never reached.

In [3], we speculated that the $(1+1)$ FEA is slower than the $(1+1)$ EA by a factor linear in the number of different possible objective values, which is $s+1$ in case of OneMax as $\mathcal{Y} = [0..s]$. Such a linear relationship seems to hold for all three algorithms, as the lower part of Fig. 4 reveals a slowdown approaching algorithm-specific constants $\zeta_i \in (0.05, 0.45)$ times $s+1$.

The OneMax problem is also suitable to check whether our implementations of the basic algorithms are correct. In [10, Fig. 3], the SAGA has a mean runtime right in the middle between 30 000 and 40 000 FEs to solve OneMax with $s = 5000$, while our implementation needs approximately 35 590 FEs. The GGA in [10] has a mean runtime of slightly above 38 000 FEs on this problem. It uses a different parameter setting for the mutation probability ($p = 0.773581/s$) than our implementation ($p = (1+\sqrt{5})/2s$). If we use the setting from [10], we obtain a mean runtime of 38 502 FEs. Both results replicate the performance observed in [10] and we can be confident that our implementations are correct.

**B. LeadingOnes Problem**

The LeadingOnes problem [25], [26] maximizes the length of a leading sequence containing only 1 bits. Its minimization

---

**Algorithm 1:** Hybrid algorithms using both FFA and direct optimization. (a) EAFEA. (b) SAHGA with $F = 1.5$, $p = \lambda/s$, and $c = 1/\lambda$.

Fig. 2. Hybrid algorithms using both FFA and direct optimization. (a) EAFEA. (b) SAHGA with $F = 1.5$, $p = \lambda/s$, and $c = 1/\lambda$.

Fig. 3. Illustrations of the OneMax, TwoMax, Trap, Jump, and Plateau problems for $s = 32$ and $w = 6$.

an expected runtime in $\Theta(s \ln s)$ FEs, while the SAGA achieves $O(s)$ [9], [22]. This is visible in the upper half of Fig. 4, which also shows that the FFA-based variants of the algorithms are consistently slower. The EAFEA needs about twice the time of the $(1+1)$ EA. The SAHGA performs identical to the SAGA, such that the curve of the former exactly covers the curve of the latter, meaning that the switch condition to FFA is never reached.

In [3], we speculated that the $(1+1)$ FEA is slower than the $(1+1)$ EA by a factor linear in the number of different possible objective values, which is $s+1$ in case of OneMax as $\mathcal{Y} = [0..s]$. Such a linear relationship seems to hold for all three algorithms, as the lower part of Fig. 4 reveals a slowdown approaching algorithm-specific constants $\zeta_i \in (0.05, 0.45)$ times $s+1$.

The OneMax problem is also suitable to check whether our implementations of the basic algorithms are correct. In [10, Fig. 3], the SAGA has a mean runtime right in the middle between 30 000 and 40 000 FEs to solve OneMax with $s = 5000$, while our implementation needs approximately 35 590 FEs. The GGA in [10] has a mean runtime of slightly above 38 000 FEs on this problem. It uses a different parameter setting for the mutation probability ($p = 0.773581/s$) than our implementation ($p = (1+\sqrt{5})/2s$). If we use the setting from [10], we obtain a mean runtime of 38 502 FEs. Both results replicate the performance observed in [10] and we can be confident that our implementations are correct.

**B. LeadingOnes Problem**

The LeadingOnes problem [25], [26] maximizes the length of a leading sequence containing only 1 bits. Its minimization
The black-box complexity of LeadingOnes is \( \Theta(s \ln \ln s) \) \cite{27,28}. The \((1 + 1)\) EA has a quadratic expected runtime on LeadingOnes \cite{29} and so does the \((1 + (\lambda, \lambda))\) GA regardless of the value of \( \lambda \) \cite{30,31}. From Fig. 5, we find that the SAGA performs worse on LeadingOnes than the GGA and \((1 + 1)\) EA, which behave similarly. The EAFEA performs almost exactly like \((1 + 1)\) FEA.

\((1 + 1)\) FEA, GFGA, and EAFEA can solve the TwoMax instances for all \( s \leq 333 \) within a mean runtime of below \( s^2 \ln s \) FE. SAFGA is faster than the above-mentioned algorithms, but slower than SAGA. Since SAGA needs more than \( s \ln s \) FE to solve the problem in average while SAFGA needs less than \( s^2 \), the experimentally observed slowdown seems to be less than linear this time.

From this moment on, we will often observe the following pattern regarding the FFA-based algorithm variants: If the basic algorithm cannot efficiently solve the problem, its FFA-based variant can. If the basic algorithm can solve the problem well, the FFA-based variant can still solve it but it is slower.

### D. Trap Function

The Trap function \cite{29,35} swaps the worst possible solution with the global optimum. The \((1 + 1)\) EA here has an expected runtime of \( \Theta(s^2) \) \cite{29}. The minimization version of the Trap function can be specified as follows:

\[
\text{Trap}(x) = \begin{cases} 
0, & \text{if } |x|_1 = 0 \\
1 + s - \max\{|x|_1, s - |x|_1\}, & \text{otherwise}
\end{cases}
\]  

The Trap function is bijective transformation of the OneMax problem. This means that all FFA-based algorithms behave exactly as on OneMax and can efficiently solve it. SAGA and SAHGA again perform the same, and for the same reason
as on TwoMax. Fig. 7 shows that all algorithms also behave similar as on the TwoMax problem.

E. Jump Problems

The Jump functions [29], [32], [36] introduce a deceptive region of width $w$ with very bad objective values right before the global optimum. The minimization version of the Jump function of scale $s$ and jump width $w$ is defined as follows:

$$\text{Jump}(x) = \begin{cases} s - |x|_1, & \text{if } (|x|_1 = s) \lor (|x|_1 \leq s - w) \\ w + |x|_1, & \text{otherwise}. \end{cases} \quad (5)$$

The expected runtime of the $(1+1)$ EA on such problems is in $\Theta(s^w + s \ln s)$ [29].

We conduct experiments with five different jump widths $w$, namely, $\lfloor \ln s \rfloor$, $\lfloor \ln s \rfloor + 1$, $\lfloor \sqrt{s} \rfloor$, $\lfloor \sqrt{s} \rfloor + 1$, and $\lfloor 0.5s \rfloor - 1$. From Fig. 8, we find that the performance of all pure algorithms quickly deteriorates with rising $w$. Since the Jump problem is another bijective transformation of the OneMax problem, all FFA-based algorithms behave the same as on OneMax. Indeed, Jump, Trap, and OneMax all are identical from the perspective of an algorithm that utilizes FFA, which is one of the properties making this concept interesting. SAHGA and EA/EFA are slightly slower, but, too, can solve the Jump problem efficiently regardless of the jump width.

F. Plateau Problems

The minimization version of the Plateau [38] function of scale $s$ with plateau width $w$ is defined as follows:

$$\text{Plateau}(x) = \begin{cases} s - |x|_1, & \text{if } (|x|_1 = s) \lor (|x|_1 \leq s - w) \\ w + |x|_1, & \text{otherwise}. \end{cases} \quad (6)$$

The expected runtime of the $(1+1)$ EA on such a problem is in $\Theta(s^w)$ [38]. Fig. 9 shows that, while the pure algorithms can solve the Plateau problem better than the Jump problem, they slow at doing so.

A plateau of the objective function is also a plateau under FFA. Indeed, we can observe that here, FFA provides no advantage for the $(1+1)$ FEA and GFGA, which are slower than the $(1+1)$ EA and GGA, respectively. The SAFGA and the SAHGA seemingly perform almost the same on all Plateau instances of a given scale $s$ in Fig. 9, regardless of the plateau width $w$, and solve them efficiently (and better than SAGA).

We repeat the experiment with both algorithms for larger scales up to 4096 in Fig. 10. We find that SAFGA and SAHGA solve the problem consistently in less than $s^4$ FEs. From the figure, it seems that the large plateaus with $w = \lfloor 0.5s \rfloor - 1$ are easier for both algorithms and the other four plateau widths lead to similar performance. When comparing the raw numbers, we find that problem tends to get easier for the algorithms the wider the plateaus are, i.e., the bigger $w$.\footnote{The difference between $\lfloor \ln s \rfloor$ and $\lfloor \sqrt{s} \rfloor + 1$ is just too small for the investigated scales for this to be visible in the diagram.}

This can be explained as follows: As shown in [3], FFA-based algorithms may optimize either toward improving solutions or toward worsening solution quality and from time to time change direction. If the algorithm switches toward worsening objective values after reaching the plateaus, it will eventually reach the worst-possible bit string $x_w$, composed of all $0$ s. There is only one such string and all strings with some $1$ s in them may have thus already received higher frequency values. Success in the FFA-based variants corresponds to finding a less-frequently encountered objective value and a failure in doing so increases $\lambda$. Once $x_w$ is reached,
WEISE et al.: FFA: OPTIMIZATION WITHOUT BIAS FOR GOOD SOLUTIONS CAN BE EFFICIENT

Fig. 9. Mean runtime measured on Plateau problems with scale $s$ and plateau width $w > 1$.

Fig. 10. More comprehensive experiment with SAFGA and SAHGA on the Plateau problem.

the algorithms may not immediately succeed to find a string with a lower objective value encounter frequency. The self-adjustment will thus increase $\lambda$, leading to more bits being flipped at once in mutation. This, in turn, increases the chance directly jump into the plateau (which may already have a high frequency assigned to it) and does so the more, the larger $w$. Once $\lambda = s$ is reached, all bits will be flipped, jumping from $x_w$ directly to the global optimum. Based on this explanation, we can also expect that a problem where plateaus surround both the optimum and $x_w$ would still be hard for the SAFGA and SAHGA.

G. N-Queens Problem

The $N$-Queens problem is defined for bit strings of length $s = N^2$ [11]. A bit string $x$ is mapped to a chess board and a queen is placed for any bit of value 1. The goal is to place $N$ queens such that they cannot attack each other. The total number of queens on the board be $Q(x) = |x|$, which might be more or less than $N$. We also count the number $Q_\xi(x)$ of queens in every single row, column, and diagonal $\xi$ of the chess board. The minimization version of the $N$-Queens problem is then

$$N\text{-Queens}(x) = N - Q(x) + N \sum_{\forall \xi} \max\{0, Q_\xi(x) - 1\}. \quad (7)$$

The $N$-Queens problems are attested moderate difficulty in [11]. They are interesting for investigating FFA because their number of possible objective values is larger than their scale $s$.

In this experiment, SAGA shows somewhat unstable performance. We therefore investigate all $N \in \{4,..12\}$ without runtime limit and 333 independent runs per algorithm and problem instance. For $N = 11$, one of these SAGA runs needed 11 936 590 163 FEs and for $N = 12$, SAGA needed 120 878 587 950 FEs in one run. This could indicate a bad, potentially exponential, worst-case runtime. The next-worst longest runtime over the 333 runs of any other algorithm is by the $(1 + 1)$ EA for $N = 12$ with 336 432 114 FEs.

Fig. 11 confirms that the SAGA performed worst. Its FFA-based variants perform much better (both need about 28 000 000 FEs for $N = 12$ in their worst runs) and similar to the $(1 + 1)$ FEA. The EAFEA performed best for larger scales, followed by GGA and $(1 + 1)$ EA.

H. Ising 1-D Problem

Ising problems define a graph $G(V = [0.s - 1], E \subseteq V^2)$ where each node stands for one bit index and the edges are undirected. If two indices $i$ and $j$ are connected, a penalty
of 1 is incurred in the objective value of a solution x if x[i] ≠ x[j], as shown in (8). In the Ising 1-D problem [11], [39], G describes a 1-D ring where each bit is connected to its predecessor and successor in x and E = E_1 is then defined in (9)

$$I(x, E) = \sum_{(i,j) \in E} |x[i] - x[j]|$$

(8)

$$E_1 = \{(i, j) | i, j \in [0..s-1] \land j = (i+1) \mod s\}. \quad (9)$$

From [39] we know that the (1 + 1) EA has an expected running time of $O(s^3)$ on this problem with small constant in the $O$-term, which is visible in Fig. 12, where its observed mean running time is between $s^2 \ln s$ and $s^3$. The simple (1 + 1) EA beats GGA, which, in turn, beats SAGA. We further find that all FFA-based algorithm variants are slower than their corresponding pure variants, but again by no more than a factor linear in s. All algorithms solved all instances in all runs within the prescribed budget.

I. Ising 2-D Problem

In the Ising 2-D problem [11], [40], the graph plugged into (8) describes a 2-D torus with $s = N^2$ and $E = E_2$, where each bit has four neighbors

$$E_2 = \left\{(\alpha + \beta N, \gamma + \delta N) \mid \alpha, \beta, \gamma, \delta \in [0..N-1] \wedge \left(\begin{array}{l} \gamma = (\alpha + 1) \mod N \land \delta = \beta \\ (\gamma = \alpha \land \delta = (\beta + 1) \mod N) \end{array}\right)\right\}. \quad (10)$$

We investigate the Ising 2-D problem up to $N = 10$, i.e., $s = 100$. Fig. 13 reveals that this problem is much harder than Ising 1-D for the pure algorithms, which can only reach a 100% success rate until $N = 6$ within the $10^{10}$ FEs budget. This fits to [40], which states that the (1 + 1) EA has exponential expected running times on a similar class of 2-D Ising models. All variants with FFA, however, can solve all Ising 2-D instances in all of their runs. Oddly, for them Ising 2-D seems to be easier than Ising 1-D for the (1+1) EA, which was the fastest algorithm on it! The EAFEA performed best by a margin. We confirmed the correct implementation of both problems with unit tests.

J. Linear Function With Harmonic Weights

The minimization version of the linear harmonic function [11] is given in (11). The (1 + 1) EA can solve this function in $\Theta(s \log s)$ FEs [11], [29]

$$\text{linHarm}(x) = 0.5s(s+1) - \sum_{i=1}^s ix_i. \quad (11)$$

This problem is particularly interesting for investigating FFA, because it is rather easy to solve, but its number of objective values grows quadratically with s. From Fig. 14, we see that the FFA-only variants require mean runtimes which could indeed be quadratically larger than the mean runtimes of the respective pure algorithms. This indicates that the cost of FFA on easy problems is likely proportional to the number of possible objective values.

K. W-Model Instances

The W-Model [41], [42] is a benchmark problem which exhibits different difficult fitness landscape features (ruggedness, deceptiveness, neutrality, and epistasis) in a tunable fashion. Nineteen diverse W-Model instances of different scales $s \in [16..256]$ have been selected based on a large experiment in [5], where they are described in detail. They are also used in [3], where the (1 + 1) FEA performed much better on them than the (1 + 1) EA.
Expanding on this prior finding, we can see from Fig. 15 that \((1 + 1)\) EA and GGA have a 100% success rate only on four W-Model instances. The SAGA can always solve the first five and has more than 80% success rate on the sixth instance. It can solve most of the W-Model instances more often than the other pure algorithms (with the exception of instances 9 and 11). In the top part of the figure, we plot the empirically estimated expected runtime (ERT) [43]. We find that SAGA tends to have a lower ERT on the harder instances while being slower on the easier ones compared to the other pure algorithms.

All algorithms with FFA can solve all W-Model instances in all runs within the budget. With the exception of instance 3, their mean runtimes are below or approximately equal to the third power of the instance scale \(s\). All of them also have similar ERT values which, again with the exception of instance 3, are always better than those of all pure algorithms.

L. MaxSat Problems

A MaxSat instance is a formula \(B : \{0, 1\}^s \rightarrow \{0, 1\}\) over \(s\) Boolean variables [44]. The variables appear as literals either directly or negated in \(c\) “or” clauses, which are all combined into one “and.” The objective function \(f(x)\), subject to minimization, computes the number of clauses which are false under the variable setting \(x\). If \(f(x) = 0\), all clauses are true, which solves the problem. The MaxSat problem exhibits low epistasis but deceptiveness [45]. In the so-called phase transition region with \(c/s \approx 4.26\), the average instance hardness for stochastic local search algorithms is maximal [46]–[48].

We apply our algorithms as incomplete solvers [49] on the ten sets of satisfiable uniform random 3-SAT instances from SATLib [46], which stem from this region. Here, the number of variables \(s\) is from \(\{20\} \cup \{25i : i \in [2..10]\}\), where 1000 instances are given for \(s \in \{20, 50, 75\}\) and 100 otherwise. For each scale level, we conduct 11,000 runs with our algorithms exactly uniformly distributed over all available instances. With the pure algorithms, we can only do this for \(s \in \{20, 50, 75\}\) due to the high runtime requirement resulting from many runs failing to solve the problem within \(10^{10}\) FEs.

From Table I, we can see that the highest number of failed runs at scale \(s = 250\) of any algorithm using FFA is lower than the lowest number of failed runs of any pure algorithm at \(s = 50\). From Table II, we find that no FFA-based algorithm has a higher ERT at scale \(s = 250\) than its pure variant on \(s = 50\). On the scales \(s \leq 75\), the FFA-based algorithms have a mean runtime which is between three and four orders of magnitude smaller that the ERT of the pure algorithms. Overall, the \((1 + 1)\) FEA performs best, closely followed by the EAFEA. These results provide striking evidence regarding the suitability of FFA to solve \(NP\)-hard and practically relevant problems.

M. Summary

In Table III, we summarize our experimental results. We try to give an impression on the worst-case time that the algorithms needed to solve the problems in relation to the problem scale \(s\), regardless of any other problem parameter. We computed \(t = \max\{\log_2 RT_i\}\), looking for values \(t\) for which \(s^t\)
marks an upper limit for the worst runtimes $R_T$, in FEs any run has consumed for solving any instance of the problem at a scale $s$.

On some problems, some of the runs were not successful or instances were skipped because it was clear that they would not finish within the budget. Both situations are signified with $\emptyset$. All columns with numerical values indicate that all runs on all explored scales in the range were successful.

From the table, we see again that SAFGA and SAHGA are the only algorithms that always succeed on all 11 problems on the listed scales. For them, the Ising 1-D is the hardest among the theory-inspired benchmarks. On the seven problem types that the SAGA can solve efficiently, its FFA-based variants only win on $N$-Queens.

The performance of the SAGA suffers when its population size $\lambda$ becomes unnecessarily large, e.g., on the LeadingOnes, $N$-Queens, Ising 1-D, and MaxSat problems. Limiting $\lambda$ to not exceed a constant like 5 or 10 or to a scale-dependent value such as $2 \ln s$ could be a remedy for this issue [50]. It would likely improve the performance of the three variants of the SAGA on problems with weak fitness distance correlation. In turn, it would probably remove the ability of the pure SAGA to efficiently solve the TwoMax and Trap problems and the ability of the SAFGA and SAHGA to efficiently solve Plateau problems. Alternative approaches would be applying self-adaptation with rollbacks [51], resetting $\lambda$ to 1 when it reaches $s$ and still cannot improve upon the best-so-far solution [52], or choosing $\lambda$ using a power-law distribution in each iteration [53].

The $(1 + 1)$ FEA, EAFEA, and GFGA can reliably solve all theory-inspired problems except Plateaus. The pure algorithms have problems with Jumps, Plateaus, Ising 2-D, and some $W$-Model instances. Interestingly, Ising 2-D appears to be easy for FFA-based methods. The FFA-based algorithms can reliably solve the $\mathcal{NP}$-hard MaxSat problem within the budget up to $s = 125$ and have their largest $t$-values on this problem. The good performance of the $(1 + 1)$ EA relative to the GGA and SAGA in Table III can be explained—at least on some problems—with the relatively low scales $s$ we investigated, with the fact that we use worst-case runtimes, and with our somewhat crude approach, which boils down performance to only a monomial, disregarding possible lower-order polynomials as well as constant or logarithmic factors.

V. Conclusion

FFA is an approach to prevent premature convergence of optimization processes. It has three very interesting and unique already known properties.

1) It creates optimization processes that are not biased toward better solutions [2].
2) It renders them invariant under all bijective transformations of the objective function value [3].
3) It can reduce the empirical mean runtime of the $(1 + 1)$ EA on several benchmark problems from exponential to polynomial in our experiments [3].

In this article, we significantly extended the knowledge along the lines of the third property above. We introduced FFA into two state-of-the-art EAs for discrete optimization, namely, the GGA and the SAGA. On problems that these algorithms can already solve efficiently, the addition of FFA leads to a slow-down which often seems to be linear in the number of distinct objective values.

On several of the investigated benchmark problems where GGA and $(1 + 1)$ EA need exponential expected runtime (TwoMax, Trap, and Jump), the GFGA, $(1 + 1)$ FEA, and the hybrid EAFEA seem to exhibit polynomial mean runtime. However, they retain the exponential mean runtime of their pure variants on the Plateau problems. On Jump and Plateau, where the SAGA also exhibits exponential expected runtime, the SAFGA with FFA and the hybrid SAHGA seem to require polynomial mean runtime only. They are also much faster on the $N$-Queens problem where the SAGA is slow. It is notable and surprising that FFA does not degenerate the self-adjusting method of SAGA. It even interacts with it positively, allowing it to solve all benchmark problems except MaxSat efficiently.

We furthermore investigated two ways to hybridize FFA with direct optimization (SAHGA and EAFEA) and found that both performed very well. For example, on $N$-Queens and Ising 2-D, the hybrid EAFEA was the fastest algorithm in our study.

With this article, we have confirmed that FFA is not an oddity that only, somehow, makes a basic and practically unimportant $(1 + 1)$ EA more efficient on hard problems. We found that it can also do so with two of the best EAs for discrete black-box optimization available.

The SAFGA and SAHGA were not necessarily the best algorithms in the lot, in particular on the MaxSat and Ising 1-D problems. Nevertheless, the existence of algorithms that may be able to solve all investigated benchmark problems from theory efficiently is very motivating. We will therefore study more variants [50], [51], [53] of these algorithms.

We believe that there is much potential in better hybrids of FFA and pure optimization. The good performance of the EAFEA on Ising 2-D is another indicator for this. We will also follow this research direction in the future.

Our first results for the $(1 + 1)$ FEA on the Job Shop Scheduling Problem [2] are very encouraging, too. We will therefore extend the scope of our experiments to more $\mathcal{NP}$-hard optimization tasks and other representations.

References

[1] T. Weise, M. Wan, K. Tang, P. Wang, A. Devert, and X. Yao, “Frequency fitness assignment,” IEEE Trans. Evol. Comput., vol. 18, no. 2, pp. 226–243, Apr. 2014.
[2] T. Weise, X. Li, Y. Chen, and Z. Wu, “Solving job shop scheduling problems without using a bias for good solutions,” in Proc. Genet. Evol. Comput. Conf. Companion (GECCO), 2021, pp. 1459–1466.
[3] T. Weise, Z. Wu, X. Li, and Y. Chen, “Frequency fitness assignment: Making optimization algorithms invariant under bijective transformations of the objective function value,” IEEE Trans. Evol. Comput., vol. 25, no. 2, pp. 307–319, Apr. 2021.
[4] T. Weise, M. Wan, K. Tang, and X. Yao, “Evolving exact integer algorithms with genetic programming,” in Proc. IEEE Congr. Evol. Comput. (CEC), 2014, pp. 1816–1823.
[5] T. Weise, Y. Chen, X. Li, and Z. Wu, “Selecting a diverse set of benchmark instances from a tunable model problem for black-box discrete optimization algorithms,” Appl. Soft Comput., vol. 92, Jul. 2019, Art. no. 106269.
[6] D. Sudholt, “Crossover speeds up building-block assembly,” in Proc. Genet. Evol. Comput. Conf. (GECCO), 2012, pp. 689–702.

[7] D. Sudholt, “How crossover speeds up building block assembly in genetic algorithms,” Evol. Comput., vol. 25, no. 2, pp. 237–274, 2017.

[8] B. Doerr, C. Doerr, and F. Ebela, “From black-box complexity to designing new genetic algorithms,” Theor. Comput. Sci., vol. 567, pp. 87–104, Feb. 2015.

[9] 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.

[10] E. C. Pinto and C. Doerr, “Towards a more practice-aware runtime analysis of evolutionary algorithms,” 2018, arXiv:1812.0493v1.

[11] C. Doerr, F. Ye, N. Hoares, H. Wang, O. M. Shire, and T. Bäck, “Benchmarking discrete optimization heuristics with IOPROFILER,” Appl. Soft Comput., vol. 88, Mar. 2020, Art. no. 106027.

[12] D. E. Goldberg and J. T. Richardson, “Genetic algorithms with sharing for multimodal function optimization,” in Proc. Int. Conf. Genet. Algorithms Appl. (ICGA), 1987, pp. 41–49.

[13] Z.-Z. Wu, X.-F. Wang, L. Zou, L.-X. Xu, X.-L. Li, and T. Weise, “Hierarchical object detection for very high-resolution satellite images,” Appl. Soft Comput., vol. 113, Dec. 2021, Art. no. 107855.

[14] Y. Olivier, L. Arnold, A. Auger, and N. Hansen, “Information-geometric optimization algorithms: A unifying view via invariance principles,” J. Mach. Learn. Res., vol. 18, no. 18, pp. 1–65, 2017.

[15] D. Corus, A. Lissovoi, P. S. Oliveto, and C. Witt, “On steady-state evolutionary algorithms and selective pressure: Why inverse rank-based allocation of reproductive trials is best,” ACM Trans. Evol. Learn. Optim., vol. 1, no. 2, p. 21, 2021.

[16] D. Sudholt, “How crossover speeds up building-block assembly in genetic algorithms,” Evol. Comput., vol. 25, no. 2, pp. 237–274, 2017.

[17] T. Friedrich, F. Quinzan, and M. Wagner, “Escaping large deceptive basins of attraction with heavy-tailed mutation operators,” in Proc. Genet. Evol. Comput. Conf. (GECCO), 2018, pp. 293–300.

[18] C. V. Huygens, D. E. Goldberg, and B. Naudts, “From TwoMax to the Ising model: Easy and hard symmetrical problems,” in Proc. Genet. Evol. Comput. Conf. (GECCO), 2002, pp. 626–633.

[19] T. Friedrich, P. S. Oliveto, D. Sudholt, and C. Witt, “Analysis of diversity-preserving mechanisms for global exploration,” Evol. Comput., vol. 17, no. 4, pp. 455–476, 2009.

[20] S. Nijssen and T. Bäck, “An analysis of the behavior of simplified evolutionary algorithms on trap functions,” IEEE Trans. Evol. Comput., vol. 7, no. 1, pp. 11–22, Feb. 2003.

[21] F. Quinzan, A. Göbel, M. Wagner, and T. Friedrich, “Evolutionary algorithms and submodular functions: Benefits of heavy-tailed mutations,” Nat. Comput., vol. 20, no. 3, pp. 561–575, 2021.

[22] B. Doerr, C. Doerr, and T. Kötzing, “Unbiased black-box complexities of jump functions,” Evol. Comput., vol. 23, no. 4, pp. 641–670, 2015.

[23] D. Antipov and B. Doerr, “Precise runtime analysis for plateau functions,” ACM Trans. Evol. Learn. Optim., vol. 1, no. 4, p. 13, 2021.

[24] S. Fischer and I. Wegener, “The one-dimensional Ising model: Mutation versus recombination,” Theor. Comput. Sci., vol. 544, nos. 2–3, pp. 755–775, 2005.

[25] S. Fischer, “A polynomial upper bound for a mutation-based algorithm on the two-dimensional Ising model,” in Proc. Genet. Evol. Comput. Conf. (GECCO), 2004, pp. 1100–1112.

[26] T. Weise and Z. Wu, “Difficult features of combinatorial optimization problems and the tunable W-model benchmark problem for simulating them,” in Proc. Compan. Mat. Genet. Evol. Comput. Conf. (GECCO), 2018, pp. 1769–1776.

[27] T. Weise, S. Niemann, H. Skubek, R. Rechle, and K. Gehls, “A tunable model for multi-objective, epistatic, rugged, and neutral fitness landscapes,” in Proc. Ann. Conf. Genet. Evol. Comput. (GECCO), 2008, pp. 795–802.

[28] N. Hansen, A. Auger, S. Finck, and R. Ros, “Real-parameter black-box optimization benchmarking: Experimental setup,” Université Paris Sud, INRIA Futurs, Equipe TAO, Orsay, France, Rep., Mar. 2012. [Online]. Available: http://coco.lri.fr/BBOB-downloads/download1.05.0 bbobdocexperiment.pdf

[29] H. H. Hoos and T. Stützle, Stochastic Local Search: Foundations and Applications. San Francisco, CA, USA: Morgan Kaufmann Publ., 2005.

[30] S. B. Rana and L. D. Whitley, “Genetic algorithm behavior in the MAXSAT domain,” in Parallel Problem Solving from Nature (PPSN V), vol. 1784, Part II. Cham, Switzerland: Springer, 1999, pp. 29–41.

[31] B. Doerr and C. Doerr, “Optimal parameter choices through self-adjustment: Applying the 1/5-th rule in discrete settings,” in Proc. Genet. Evol. Comput. Conf. (GECCO), 2015, pp. 1335–1342.

[32] H. Mühlenbein, “How genetic algorithms really work: Mutation and hillclimbing,” in Parallel Problem Solving from Nature 2 (PPSN-II), Cham, Switzerland: Springer, 1992, pp. 15–26.

[33] S. Droste, T. Jansen, and I. Wegener, “Upper and lower bounds for randomized search heuristics in black-box optimization,” Theory Comput. Syst., vol. 39, no. 4, pp. 525–544, 2006.

[34] L. D. Whitley, “The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best,” in Proc. Intl. Conf. Genet. Algorithms (ICGA), 1989, pp. 116–121.

[35] G. Rudolph, Convergence Properties of Evolutionary Algorithms. Hamburg, Germany: Verlag Dr. Kracht, 1997.

[36] P. Afshani, M. Agrawal, B. Doerr, K. G. Larsen, K. Mehlhorn, and C. Winzen, “The query complexity of finding a hidden permutation,” in Space-Efficient Data Structures, Streams, and Algorithms, Berlin, Germany: Springer, 2013, ch. 1, pp. 1–11.

[37] P. Afshani, M. Agrawal, B. Doerr, C. Doerr, K. G. Larsen, and K. Mehlhorn, “The query complexity of a permutation-based variant of mastermind,” Discret. Appl. Math., vol. 260, pp. 28–50, May 2019.

[38] P. Droste, T. Jansen, and I. Wegener, “On the analysis of the \((1+1)\) evolutionary algorithm,” Theor. Comput. Sci., vol. 276, nos. 1–2, pp. 51–81, 2002.

[39] D. Antipov, B. Doerr, and V. Karavaya, “A tight runtime analysis for the \((1+\lambda,\lambda)\) GA on leadingones problem,” in Proc. 15th ACM/SIGEVO Conf. Found. Genet. Algorithms (FOGA), 2019, pp. 169–182.
Thomas Weise received the Ph.D. degree in computer science from the University of Kassel, Kassel, Germany, in 2009. He then joined the University of Science and Technology of China, Hefei, Anhui, China, first as a Postdoctoral Fellow, later as an Associate Professor. In 2016, he moved to Hefei University, Hefei, China, as a Full Professor to found the Institute of Applied Optimization, School of Artificial Intelligence and Big Data. He has authored or coauthored 115 peer-reviewed academic publications, including over 40 articles and 65 conference papers. His research interests include metaheuristic optimization and algorithm benchmarking. Dr. Weise was awarded the title Hefei Specially Recruited Foreign Expert, Hefei, in 2019, and the Hefei City Friendship Award in 2020.

Zhize Wu received the bachelor’s degree from the Anhui Normal University, Wuhu, Anhui, China, in 2012, and the Ph.D. degree in computer science from the University of Science and Technology of China, Hefei, Anhui, China, in 2017. He then joined the Institute of Applied Optimization, School of Artificial Intelligence and Big Data, Hefei University, Hefei, as Lecturer. His research interests include image processing, neural networks, deep learning, machine learning in general, and algorithm benchmarking.

Xinlu Li received the bachelor’s degree from Fuyang Normal University, Fuyang, Anhui, China, in 2002, the M.Sc. degree in computer science from Anhui University, Hefei, Anhui, China, in 2009, and the Ph.D. degree in computer science from TU Dublin, Dublin, Ireland, in 2019. His career with Hefei University started as a Lecturer in 2009, a Senior Lecturer in 2012, in 2018 he joined the Institute of Applied Optimization, and in 2019 he became an Associate Researcher. From 2014 to 2018, he was a Teaching Assistant with the School of Computing of TU Dublin. His research interests include swarm intelligence and optimization, algorithms which he applies to energy-efficient routing protocol design for large-scale wireless sensor networks.

Yan Chen received the bachelor’s degree from the Anhui University of Science and Technology, Huainan, Anhui, China, in 2011, the M.Sc. degree from the China University of Mining and Technology, Xuzhou, Jiangsu, China, in 2014, and the Doctoral degree from TU Dortmund, Dortmund, Germany, in 2019. There, he studied spatial information management and modeling as a member of the Spatial Information Management and Modeling Department, School of Spatial Planning. He joined the Institute of Applied Optimization, School of Artificial Intelligence and Big Data, Hefei University as a Lecturer in 2019. His research interests include hydrological modeling, geographic information system, remote sensing, and sponge city/smart city applications, as well as deep learning and optimization algorithms.

Jörg Lässig received the Ph.D. degree in efficient algorithms and models for cooperation generation and control from the Chemnitz University of Technology, Chemnitz, Germany, in 2009. He is a Full Professor for enterprise application development with the University of Applied Sciences Zittau/Görlitz, Görlitz, Germany, and also a Research Manager with the Fraunhofer Society. This was followed by postdoctoral stays with the International Computer Science Institute, Berkeley, CA, USA, and with the Università della Svizzera italiana, Lugano, Switzerland. His research includes metaheuristics for optimization, quantum algorithmics, and cybersecurity for critical infrastructures.