Frequency Fitness Assignment: Making Optimization Algorithms Invariant under Bijective Transformations of the Objective Function Value

Thomas Weise Member, IEEE, Zhize Wu, Xinlu Li, and Yan Chen

Abstract—Under Frequency Fitness Assignment (FFA), the fitness corresponding to an objective value is its encounter frequency in fitness assignment steps and is subject to minimization. FFA renders optimization processes invariant under bijective transformations of objective function values. On TwoMax, Jump, and Trap functions of dimension $s$, the classical $(1+1)$-EA with standard mutation at rate $1/s$ can have expected runtimes exponential in $s$. In our experiments, a $(1+1)$-FEA, the same algorithm but using FFA, exhibits mean runtimes that seem to scale as $s^2 \ln s$. Since Jump and Trap are bijective transformations of OneMax, it behaves identical on all three. On OneMax, LeadingOnes, and Plateau problems, it seems to be slower than the $(1+1)$-EA by a factor linear in $s$. The $(1+1)$-FEA performs much better than the $(1+1)$-EA on W-Model and MaxSat instances. We further verify the bijection invariance by applying the Md5 checksum computation as transformation to some of the above problems and yield the same behaviors. Finally, we show that FFA can improve the performance of a memetic algorithm for job shop scheduling.

Index Terms—Frequency Fitness Assignment, Evolutionary Algorithm, OneMax, TwoMax, Jump problems, Trap function, Plateau problems, W-Model benchmark, MaxSat problem, Job Shop Scheduling Problem, $(1+1)$-EA, memetic algorithm

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. Frequency Fitness Assignment (FFA) [1, 2] was developed to enable algorithms to escape from local optima. In FFA, the fitness corresponding to an objective value is its encounter frequency so far in fitness assignment steps and is subject to minimization. As we discuss in detail in Section II, FFA turns a static optimization problem into a dynamic one where objective values that are often encountered will receive worse and worse fitness.

In this article, we uncover a so-far unexplored property of FFA: It is invariant under any bijective transformation of the objective function values. This is the strongest invariance known to us and encompasses all order-preserving mappings. Other examples for bijective transformations include the negation, permutation, or even encryption of the objective values. According to [3], invariance extends performance observed on a single function to an entire associated invariance class, that is, it generalizes from a single problem to a class of problems. Thus it hopefully provides better robustness w.r.t. changes in the presentation of a problem. FFA generalizes the performance of an algorithm on OneMax to all problems which are bijections thereof, including Jump and Trap.

While invariances are generally beneficial for optimization algorithms [4, 5], such strong invariance comes at a cost: The idea that solutions of better objective values should be preferred to those with worse ones can no longer be applied, since many bijections are not order-preserving. FFA only considers whether objective values are equal or not. One would expect that this should lead to a loss of performance. We find that the opposite is the case on several benchmarks evaluated in our study. On those where FFA increases the number of function evaluations (FEs) to find the optimum, i.e., the runtime, it seems to do so only linearly with the number of different objective values or the problem dimension $s$, as both cases are indistinguishable in the investigated problems.

We plug FFA into the most basic EA [6], the $(1+1)$-EA with standard mutation at rate $1/s$, and obtain the $(1+1)$-FEA. We investigate its performance on several well-known problems, namely the OneMax, LeadingOnes, TwoMax, Jump, Trap, and Plateau functions, the W-Model, and MaxSat, all defined over bit strings of length $s$. We find that the resulting $(1+1)$-FEA is slower on OneMax, LeadingOnes, and on the Plateau functions, while it very significantly reduces the runtime needed to solve the other problems. Most notably, in our experiments, it has runtime requirements in the scale of $s^2 \ln s$ on the TwoMax, Trap and Jump problems, for which the expected runtime needed by the $(1+1)$-EA to find the global optimum is in $\Omega(s^w)$, $\Theta(s^w)$, and $\Theta(s^{w+\ln s})$ (for jump width $w$) FEs, respectively. We confirm the invariance under bijections of the objective value by solving several benchmark problems with the $(1+1)$-FEA by optimizing the Md5 checksums, i.e., cryptographic hashes, of their objective values and observing no change in algorithm behavior. We also explore plugging FFA into a well-performing algorithm for a job shop scheduling problem, where it can improve the result quality under budget constraints.

In Section II, we discuss the invariance property of FFA and how FFA can be plugged into the $(1+1)$-EA. Related works are discussed in Section III. Our comprehensive experimental
study is given in Section IV. We conclude our article and give pointers to future work in Section V.

II. FREQUENCY FITNESS ASSIGNMENT

This study investigates the impact of FFA when plugged into the maybe most basic EA, the (1+1)-EA. The (1+1)-EA starts with a random bit string \( x_0 \) of length \( s \). Until the termination criterion is met, in each step, it applies the standard mutation operator, where each of the \( s \) bits of \( x_c \) is flipped independently with probability \( 1/s \) and the result is a new string \( x_n \). If \( x_n \) is at least as good as \( x_c \), it replaces \( x_c \). The expected runtime of the (1+1)-EA for an arbitrary objective function is at most \( s^8 \) [7]. Some of the benchmark problems we investigate invoke this boundary.

We apply a slight modification of the (1+1)-EA, called the (1+1)-FEA\(_{>0} \) [8]: The standard mutation in each iteration is repeated until at least one bit is flipped [9]. No FE is wasted by evaluating a candidate solution identical to the current one.

The probability of this in the (1+1)-EA is \( 1 - (1 - s^{-1})^s \), which approaches \( 1/e \approx 0.368 \) for \( s \to \infty \). This small change thus saves more than one third of the FEs while not changing any other characteristic of the algorithm [8, 10]. In the following text, expected runtimes for the (1+1)-EA will therefore be corrected by factor \( 1 - (1 - s^{-1})^s \) to hold for the (1+1)-FEA\(_{>0} \) where necessary.

In Figure 1, we put the pseudo code of the (1+1)-FEA\(_{>0} \) next to a simplified version of the (1+1)-FEA\(_{\geq 0} \). We assume that

1) the objective function \( f \) is subject to minimization, that
2) its upper bound \( UB \) is known, that
3) all objective values are integers greater or equal to 0, and that
4) the solution space is \( \{0, 1\}^s \), the bit strings of length \( s \).

This can be established for many well-known benchmark problems on which the (1+1)-EA is usually investigated, as well as for many practical optimization problems like MaxSat.

Under these assumptions, only minimal changes to the (1+1)-FEA\(_{>0} \) are necessary to introduce FFA: An array \( H \) of integers of length \( UB + 1 \) is used to hold the frequency of each objective value in \([0..UB]\). Before selecting one of the two candidate solutions with objective values \( f_c \) and \( f_n \), the frequencies \( H[f_c] \) and \( H[f_n] \) of these objective values are increased. The results of these increments are compared. Note: Both frequencies are increased, because if \( H[f_c] \) was not incremented, solutions with unique objective values could become traps for the optimization process.

In order to conduct an efficient search under FFA, the set \( \mathbb{Y} \) of possible objective values for the problem to be solved should not be too big. FFA must maintain a frequency table \( H \), which has the same size as \( \mathbb{Y} \). Also, FFA attempts to distribute the search effort evenly over all objective values. In the extreme case where each distinct solution has a different objective value, FFA almost degenerates the search to a random walk.

Most often, the (1+1)-EA is analyzed as maximization algorithm. Since the (1+1)-FEA minimizes the objective value frequencies, we also present the (1+1)-EA for minimization and define the benchmark problems in Section IV accordingly.

The (1+1)-FEA implementation given in Figure 1b can easily be extended towards a \( (\mu+\lambda) \)-EA. It can also be modified to handle problems with unknown upper and lower bounds of the objective function (or objective functions that return real numbers but can still be discretized) by implementing \( H \) as hash table [11] (see Section IV-G). FFA can be introduced into arbitrary metaheuristics.

**Theorem 1:** The sequence of candidate solutions \( x \in \mathbb{X} \) generated by an optimization process applying FFA is invariant under any bijection \( g : \mathbb{Y} \to \mathbb{Z} \) of the objective function \( f : \mathbb{X} \to \mathbb{Y} \), where \( \mathbb{X} \) is the solution space, \( \mathbb{Y} \) is a finite subset of \( \mathbb{R} \), and \( \mathbb{Z} \) is a set of the same size.

**Proof:** The bijection \( g \) maps each value from \( \mathbb{Y} \) to one value in \( \mathbb{Z} \) and vice versa. Therefore, if two objective values identify the same (or a different) entry in \( H \), so will their bijective transformations. Under FFA, only the entries in \( H \) are modified and compared to make selection decisions.

We can also prove this inductively: Assume that two runs of the (1+1)-FEA\(_{>0} \) which minimize \( f \) and \( g \circ f \), respectively, are identical until iteration \( t \): They have the same random seed, same \( x_c \), and \( H[y] = H'[g(y)] \forall y \in \mathbb{Y} \) holds for their respective FFA tables. Both will sample the same next point \( x_n \), \( H[f(x_n)] = H'[g(f(x_n))] \) and \( H'[f(x_n)] = H'[f(x_n)] \) will still hold after incrementing the entries. Hence, both will make the same decision regarding the update of \( x_c \) and begin iteration \( t + 1 \) in the same state. In Sections IV-D, IV-E, and IV-G, we provide experimental evidence that this invariance indeed holds.

III. RELATED WORK

FFA was designed as an approach to prevent the premature convergence to a local optimum. In the context of EAs, it is therefore related to fitness sharing, niching, and clearing [12, 13]. Several such diversity-preserving mechanisms have been plugged into a \( (\mu+1) \)-EA and studied theoretically in [14] on TwoMax, where the original algorithm requires \( O(s^8) \) FEs to find the optimum. It is found that avoiding fitness or genotype duplicates does not help, whereas with deterministic
crowding and sufficiently large $\mu$, the problem can be solved efficiently with high probability. With fitness sharing and $\mu \geq 2$, the $(\mu+1)$-EA can solve TwoMax in $O(\mu s \log s)$. Different from the above methods, which only consider the current population, FFA tries to guide the search away from objective values that have been encountered often during the whole course of the optimization process. In Section IV-C, we will show that FFA can help solving TwoMax efficiently already at $\mu = 1$.

Another related idea is Tabu Search (TS) [15], which improves local search by declaring solutions (or solution traits) which have already been visited as tabu, preventing them from being sampled again. Like FFA, it utilizes the search history, but usually in form of a list of tabu solutions or solution traits. Different from FFA, the TS relies on the order of objective values when deciding which solutions to accept.

The Fitness Uniform Selection Scheme (FUSS) [16] selects solutions in such a way that their corresponding objective values are approximately uniformly distributed within the range of the minimum and maximum objective value in the population. The Fitness Uniform Deletion Scheme (FUDS) [17] works similarly, but instead of selecting individuals, it deletes them when slots in the population are required to integrate the offspring. Both methods need populations, only consider the individuals in the current population, and are only invariant under translation and scaling of the objective function.

The ageing operator in Artificial Immune Systems (AIS) deletes individuals either after they have survived a certain number of iterations, with a certain probability, or both [18–20]. Ageing has also been applied in EAs [21]. Like FFA, ageing makes solutions less attractive if they remain in the population for a long time. Different from FFA, the information about these solutions disappears from the optimization process once they “die.”

Methods which try to balance between solution quality and population diversity are today grouped under the term Quality-Diversity (QD) algorithms [22–24].

Novelty Search (NS) [25] is a QD algorithm. Instead of an objective function $f$, NS uses a (dynamic) novelty metric $\rho$. This metric is computed, e.g., as mean behavior difference to the $k$ nearest neighbors in an archive of past solutions. FFA works on the original objective function and just transforms it to a dynamic fitness measure. It does not require an archive of solutions but uses a table $H$ counting the frequency of the objective values.

While NS was aimed to abandon the objective function $f$, using it as behavior definition was also tested [25]. Then, $\rho$ is the mean distance to $k$ neighbors (or all solutions ever found) in the objective space. Unlike FFA, this uses the assumption that differences between objective values are useful or correlate with diversity. Novelty Search with Local Competition (NSLC) [26] combines the search for finding diverse solutions with a local competition objective rewarding solutions which can outperform those most similar to them.

The MAP-Elites algorithm [27] combines a performance objective $f$ and a user-defined space of features that describe candidate solutions, which is not required by FFA. MAP-Elites searches for highest-performing solution in each cell of the discretized feature space.

Surprise Search (SS) [28] uses the concept of surprise as an alternative to novelty. A solution is scored by the difference of its observed behavior from the expected behavior. A history of discovered solution behaviors is maintained and used to predict the behavior of the new solutions. SS has also been combined with NSLC in a multi-objective fashion [22].

All of the above algorithms are conceptually different from FFA. They either are complete optimization methods (NS, QD, TS) or modules for EAs (FUSS/FUDS), while FFA can be plugged into many different optimization algorithms. Unlike FFA, none of the above methods exhibits an invariance under bijective transformations of the metrics they try to optimize.

From the perspective of invariances, FFA is related to Information-Geometric Optimization (IGO) [3]. IGO also replaces the objective function $f$ with an adaptive transformation of it. This transformation indicates how good or bad an objective value is relative to other observed objective values, i.e., is different from our method which simply compares encounter frequencies. IGO is invariant under all strictly increasing transformations of $f$, whereas FFA creates invariance under all bijective transformations. IGO is a complete family of optimization methods which can also exhibit invariance under several transformations of the search space. Since FFA only works on $f$, it cannot provide such invariances. IGO can optimize continuous objective functions, which is not possible with FFA.

### IV. Experiments

We now apply the $(1+1)$-EA$_{\geq 0}$ and the $(1+1)$-FEA$_{\geq 0}$ to minimization versions of different classical optimization problems. We initialize the $(1+1)$-EA$_{\geq 0}$ and the $(1+1)$-FEA$_{\geq 0}$ with the same random seeds for each run, i.e., we always have pairs of runs starting at the same random initial solution and sampling the same first offspring solutions for both algorithms.

The runs are terminated when they discover the optimum. In some experiments, we additionally limit the computational budget to $10^{10} = 10^6 000 000 000$ FEs. This should be enough to converge on problems that the algorithms can solve well, as can be seen in Section IV-B. Leading to several hours to more than a day for a single run on the corresponding problems, this was also the maximum budget we could feasibly allow.

Whenever all runs on an instance succeed to find the optimum, we can compare the mean runtime ‘mean(RT)’ they need to do so in terms of the consumed FEs (often called the first hitting time). When some runs fail in the budget-limited settings, we follow the approach from [29] and use the empirically estimated expected runtime (ERT) instead. The ERT for a problem instance is estimated as the ratio of the sum of all FEs that all the runs consumed until they either have discovered a global optimum or exhausted their budget, divided by the number of runs that discovered a global optimum [29]. The ERT is the mean expected runtime under the assumption of independent restarts after failed runs, which then may either succeed (consuming the mean runtime of the successful runs) or fail again (with the observed failure probability, after consuming $10^{10}$ FEs).
In order to guarantee the reproducibility of our work, we provide the complete data used in this paper, including the result log files, the scripts used to generate all the figures and tables, as well as the source code of all algorithms and all benchmark problems in [30].

A. OneMax Problems

OneMax [9] is a unimodal optimization problem where the goal is to discover a bit string of all ones. Its minimization version of dimension $s$ is defined below and illustrated in Figure 2:

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

It has a black-box complexity of $\Omega(s/\ln s)$ [31, 32]. Here, an (1+1)-EA has an expected runtime of $\Theta(s\ln s)$ FEs [9]. A very exact formula [33] with our correction factor for the (1+1)-EA is given in Equation (2), where $C_1 \approx 1.89254$ and $C_2 \approx 0.5978975$.

$$[1 - (1-s^{-1})^s] \left[ \varepsilon s \ln s - C_1 s + 0.5 \varepsilon \ln s + C_2 + O(\ln s)/s \right]$$

We conduct 3333 runs with both the (1+1)-EA$_{>0}$ and (1+1)-FEA$_{>0}$ on this problem for each $s \in [3..333]$ and 71 runs for 26 selected larger values of $s$ up to 4096, all without budget constraint. In Figure 3, we illustrate the mean runtime to solve the instances with the range of the 15.9% to the 84.1% quantiles in the background. In the top-most sub-figure, we illustrate all results for $s \in [3..52]$. The middle figure is a log-log plot based on the complete data, but with marks only placed at $s \in \{2^i, \text{round}(2^i/3)\}$ to not clutter the plot. In both diagrams, we illustrate the results of Equation (2) without the $O(\ln s)/s$ term. They exactly match the results of the (1+1)-EA$_{>0}$.

The mean runtime of the (1+1)-FEA$_{>0}$ seems to be in the scale of $s^2 \ln s$ for the investigated range of $s$. The illustrated model was obtained using linear regression on the complete set of 1’105’069 runs with the inverse variances of the measured runtimes per distinct $s$ value used as weights. All regression models in the rest of this article are obtained in the same way. The curve of the model visually fits to the mean runtimes and the adjusted $R^2$ value of 0.8 indicates that it can explain most of the variance in the data.

The observed distribution of the runtime is skewed and the median is lower than the mean on all dimensions. This is illustrated exemplarily in the histogram for dimension $s = 32$ in the lower part of Figure 3. Its shape resembles a log-normal distribution.

---

1These quantiles are wider than the inter-quartile range and would represent exactly the range mean-stdev to mean+stdev under a normal distribution.
distribution or a sum of parameterized geometric distributions [34]. For \( s \leq 8 \), the histograms look like exponential distributions, caused by the high chance of randomly guessing the optimum.

Figure 4 illustrates nine typical runs of the \((1+1)-\text{FEA}_>0\) on the OneMax problem with \( s = 64 \). Initially, some of the runs progress towards better solutions, others to worse. They change the search direction from time to time. This oscillation is repeated until the global optimum is discovered.

**B. LeadingOnes Problems**

The LeadingOnes problem [4, 35] maximizes the length of a leading sequence containing only 1 bits. Its minimization version of dimension \( s \) is defined as follows:

\[
\text{LeadingOnes}(x) = s - \sum_{i=1}^{s} \prod_{j=1}^{i} x[j]
\]  

The problem exhibits epistasis, as the bit at index 2 can change the search direction from time to time. This oscillation results from the slightly different shapes of the two LeadingOnes problems for odd and even values of \( s \). Since the basins of attraction have the same size, a \((1+1)-\text{EA}\) can solve the problem in \( \Theta(s \ln \ln s) \) [36]. The \((1+1)-\text{EA}\) has a quadratic expected runtime on LeadingOnes [7]. The exact formula [37, 38] is presented with our correction factor in Equation (4):

\[
[1 - (1 - s^{-1})^s] \left[ 0.5s^2 \left((1 - 1/s)^{-s} - 1 + 1/s \right) \right]
\]

Figure 5 has the same structure as Figure 3 and is based on an experiment with the same parameters, only using the LeadingOnes instead of the OneMax problem. The \((1+1)-\text{EA}_>0\) behaves as predicted in Equation (4).

The runtime of the \((1+1)-\text{FEA}_>0\) fits to the illustrated regression model for the investigated range of \( s \) and can explain almost all of the variance of the data. Due to the approximately cubic runtime, the mean time to solve LeadingOnes at \( s = 4096 \) is 9.9 \times 10^9 FEs. The histogram of the observed runtimes for \( s = 32 \) in the lower part of Figure 5 is slightly skewed.

**C. TwoMax Problems**

The minimization version of the TwoMax [39, 40] problem of dimension \( s \) can be defined as follows:

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

The TwoMax problem introduces deceptiveness in the objective function by having a local and a global optimum. Since their basins of attraction have the same size, a \((1+1)-\text{EA}\) can solve the problem in \( \Theta(s \ln s) \) steps with probability 0.5 while otherwise needing exponential runtime. The resulting overall expected runtime is in \( \Omega(s^2) \) [14, 39].

On each instance of TwoMax with \( s \in [3...32] \), we conduct 71 runs with \((1+1)-\text{EA}_>0\) and 3333 with \((1+1)-\text{FEA}_>0\). For all experiments from here on except in Section IV-J, we use a budget of \( 10^{10} \) FEs. The \((1+1)-\text{EA}_>0\) succeeds in solving the problem only in about half of the runs for \( s > 10 \) within the budget, which was the reason for the 71-run limit. We illustrate its performance in Figure 6 only for dimensions \( s < 10 \) where it always succeeded.

**D. Jump Problems**

The Jump functions as defined in [7, 39] 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 dimension \( s \) and jump width \( w \) is defined as follows:

\[
\text{Jump}(x) = \begin{cases} 
|s - |x|_1| & \text{if } (|x|_1 = s) \lor (|x|_1 \leq s - w) \\
w + |x|_1 & \text{otherwise}
\end{cases}
\]

Researchers have formulated different types of Jump functions. The one in [41], e.g., is similar to our Plateau function but differs in the plateau objective value.
Fig. 6: The runtime measured for the (1+1)-EA and (1+1)-FEA on the TwoMax problem.

Fig. 7: The runtime measured for the (1+1)-EA and (1+1)-FEA on Jump problems with dimension \( s \) and jump width \( w > 1 \).

The expected runtime of the (1+1)-EA on such problems is in \( \Theta(s^w + s \ln s) \) [7]. The Jump problem is a bijective transformation of the OneMax problem.\(^3\) The (1+1)-FEA will exhibit the same behavior and runtime requirement on any jump problem instance as on a OneMax instance of the same dimension \( s \), regardless of the jump width \( w \).

We conduct experiments with five different jump widths \( w \), namely \( \lceil \ln s \rceil, \lfloor \ln s \rfloor + 1, \lfloor \sqrt{s} \rfloor, \lceil \sqrt{s} \rceil + 1, \) and \( 0.5s - 1 \). We illustrate the results in Figure 7 only for those setups where a success rate of 100% within the \( 10^{10} \) FEs were achieved in 71 runs. (1+1)-FEA finds the optimum in all runs and all the observed mean runtimes fall on the function fitted to the results on OneMax (see Figure 3), confirming that the two problems are indeed identical from the perspective of an algorithm using FFA. As expected, the runtime of the (1+1)-EA steeply increases with the jump width \( w \) and it is outperformed by the (1+1)-FEA.

Depending on its configuration, the AIS Opt-IA [18] needs runtimes of at least \( O(s^2 \ln s) \) and \( O(s^3) \) FEs on OneMax and LeadingOnes, respectively. It seems that our (1+1)-FEA has similar requirements, i.e., compared to the (1+1)-EA with standard bit mutation, a linear slowdown is incurred on these problems. However, on the Jump problems, Opt-IA needs, again depending on its configuration, at least \( O(s^w + 1 \cdot e^w) \) FEs.

The Trap function [7, 45] is very similar to the OneMax problem, except that it replaces the worst possible solution there with the global optimum. Following a path of improving objective values will always lead the optimization algorithm away from the global optimum. The (1+1)-EA here has an expected runtime of \( \Theta(s^w) \) [7]. The minimization version of the Trap function can be specified as follows:

\[
\text{Trap}(x) = \begin{cases} 
0 & \text{if } |x| = 0 \\
|s - |x|| + 1 & \text{otherwise}
\end{cases} \tag{7}
\]

E. Trap Function

The Trap function [7, 45] is very similar to the OneMax problem, except that it replaces the worst possible solution there with the global optimum. Following a path of improving objective values will always lead the optimization algorithm away from the global optimum. The (1+1)-EA here has an expected runtime of \( \Theta(s^w) \) [7]. The minimization version of the Trap function can be specified as follows:

\[
\text{Trap}(x) = \begin{cases} 
0 & \text{if } |x| = 0 \\
|s - |x|| + 1 & \text{otherwise}
\end{cases} \tag{7}
\]

The Trap function is another bijective transformation of the OneMax problem. When we plot the results from 3333 runs of the (1+1)-FEA on the Trap function in Figure 8, we find that the results are almost exactly identical to those obtained on OneMax and illustrated in Figure 3. The function fitted to the mean runtime on OneMax, again plotted in Figure 8, passes through the points measured on the Trap function.
Fig. 9: The runtime measured for the (1+1)-EA_{>0} and (1+1)-FEA_{>0} on Plateau problems with dimension $s$ and plateau width $w > 1$.

F. Plateau Problems

The minimization version of the Plateau [46] function of dimension $s$ with plateau width $w$ is defined as follows:

$$\text{Plateau}(x) = \begin{cases} s - |x| & \text{if } (|x| = s) \lor (|x| \leq s - w) \\ w & \text{otherwise} \end{cases}$$  \hspace{1cm} (8)

The expected runtime of the (1+1)-EA on such a problem is in $\Theta(s^w)$ [46]. The Plateau problems are non-bijective transformation of OneMax. Instead, they reduce the number of possible objective values ($|\mathbb{Z}| < |\mathbb{N}|$). We can expect that the fitness of the solutions on the plateau will get worse quickly under FFA. We conduct the same experiment as for the Jump function with the Plateau function and plot the results in the same manner in Figure 9. This time, the (1+1)-FEA_{>0} performs worse than the (1+1)-EA. Interestingly, if we divide the observed mean runtimes of (1+1)-FEA_{>0} by the problem dimension $s$, we approximately obtain those observed with (1+1)-EA_{>0} (see the gray marks in Figure 9). This might be a coincidence and more research is necessary.

G. Bijection Invariance: Md5 Checksum of Objective Values

We now repeat our experiments with the (1+1)-FEA_{>0} on the OneMax, TwoMax, LeadingOnes, and Trap problems with $s \in [3,32]$, but use a transformation of the objective functions: Instead of working on the objective values directly, we optimize their Md5 checksums. We therefore implement $H$ as a hash table where their encounter frequencies are stored. The Md5 checksum is a 128 bit message digest published in [47], where it is conjectured that it is computationally infeasible to produce two messages having the same message digest. Although Md5 checksums are not an encryption method, they do allow us to further test the invariance under such “extreme” transformations and the idea of implementing $H$ as hash table without further assumptions.4

We use the same random seeds as in the original runs working on the objective values. We find that all 3333 runs on all the instances have the same (FE, objective value)-traces as their counterparts, which also follows from Theorem 1. Illustrating these results here has no merit, as the figures would be identical to those already shown. We include the full log files as well as the algorithm implementation in our dataset [30].

H. W-Model Instances

The W-Model [48–50] is a benchmark problem which exhibits different difficult fitness landscape features in a tunable fashion.5 These include the base size (via parameter $n$), neutrality (via parameter $m$), epistasis (via parameter $\nu$), and ruggedness (via parameter $\gamma$), from which instances of dimension $s = mn$ result. The W-Model base problem is equivalent to OneMax but searches for a string of alternating 0 and 1 bits. Different transformations are applied to it. While the ruggedness transformation is a bijective transformation of objective function, the mappings introducing neutrality and epistasis transform the search space itself. 19 diverse W-Model instances have been selected in [51] based on a large-scale experiment. No theoretical bounds for the runtimes on these instances are known, but they exhibit different degrees of empirical hardness for different algorithms.

We conduct 71 runs for both algorithms on each of these 19 W-Model instances. In Table I, we presented the fraction $fs$ of runs that found the global optimum and the ERT for (1+1)-EA_{>0}. While it can always solve the four easiest instances, its success rate within the $10^{10}$ FEIs then drops, which leads to very high ERT values. The (1+1)-FEA_{>0} is always faster than (1+1)-EA_{>0} and all of its runs discovered the global optima of their respective W-Model instances. In

---

4It can be assumed that applying (1+1)-EA to this problem would yield the worst-case complexity and we thus omit doing it.

5There was a mistake in [48]: at line 19 of Algorithm 1, “start” should be replaced with “n”. This was corrected in [51, 52] and was always correct in the W-Model implementation [49].
this case, mean(RT) = ERT and we list it alongside the median runtime med(RT), which, like on the previously investigated problems, is always smaller than the mean.

Of special interest here is instance 6, which could not be solved by (1+1)-EA_{>0} at all. Here, \( s = mn = 32 \) and only a ruggedness transformation with \( \gamma = 397 \) is performed, while no additional epistasis (\( \nu = 2 \) or neutrality (\( m = 1 \)) are introduced in the landscape. In other words, here, the objective function is equivalent to a (bijective) permutation of the objective values produced by a OneMax instance (with a different but equivalent base problem).

This permutation leads to a long deceptive slope in the mid-range of the original objective values and three extremely rugged spikes near the global optimum, i.e., we can expect it to have a hardness similar to the Jump or Trap functions for the (1+1)-EA, which the experiment confirms. Only for this instance, we conduct 3333 runs with (1+1)-FEA_{>0} and find that the mean 1602 and median 1355 of the runtime are very close to those on the OneMax (1620, 1375) and Trap functions (1620, 1390), which again confirms the invariance of FFA towards bijective transformations of the objective function.

### I. MaxSat Problems

The Satisfiability Problem is one of the most prominent problems in artificial intelligence. An instance is a formula \( B : \{0,1\}^s \rightarrow \{0,1\} \) over \( s \) Boolean variables. The variables appear as literals either directly or negated in so-called \( \textit{or} \) clauses, which are all combined into one \( \textit{and} \). Solving a Satisfiability Problem means finding a setting \( x \) for the variables so that \( B(x) \) becomes \textit{true} (or whether such a setting exists). This \( \mathcal{NP} \)-hard [53] decision problem is transformed to an optimization version, the MaxSat problem [54], where the objective function \( f(x) \), subject to minimization, computes the number of clauses which are \textit{false} under \( x \). If \( f(x) = 0 \), all clauses are \textit{true}, which solves the Satisfiability Problem.

The worst possible value \( UB \) that \( f \) can take on is \( c \).

The MaxSat problem exhibits low epistasis but deceptive character [55]. In the so-called phase transition region with \( c/s \approx 4.26 \), the average instance hardness for stochastic local search algorithms is maximal [56–58]. We apply our algorithms as incomplete solvers [59] on the ten sets of satisfiable uniform random 3-SAT instances from SATLib [56], 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 \( s \in \{20,50,100\} \) and 100 otherwise. With the (1+1)-EA_{>0}, we can only conduct 11 runs for each \( s \in \{20,50,75\} \) due to the high runtime requirement resulting from many runs failing to solve the problem within \( 10^{10} \) FEs. With the (1+1)-FEA_{>0}, we conduct 11 runs for \( s \in \{20,50,100\} \) and 110 runs for each dimension other than these, i.e., have \( 110 * 100 = 11 * 1000 = 11'000 \) runs for each instance dimension \( s \) in SATLib.

The overall performance of the algorithms aggregated over the instance sets is given in Table II. We find that the (1+1)-FEA_{>0} performs much better than the (1+1)-EA_{>0}. While the former can reliably solve instances of all dimensions, the latter already fails in almost half of the runs for \( s = 75 \). The overall ERT of the (1+1)-FEA_{>0} for dimension \( s = 250 \) is only about 7% of the ERT that the (1+1)-EA_{>0} needs over all instances of \( s = 50 \).

TABLE II: The fraction \( fs \) of successful runs, the ERT, and the mean end objective value \( mean(f_B) \) for (1+1)-EA_{>0} and (1+1)-FEA_{>0} on the satisfiable MaxSat instances from SATLib.

<table>
<thead>
<tr>
<th>instance set</th>
<th>(1+1)-EA_{&gt;0}</th>
<th>(1+1)-FEA_{&gt;0}</th>
</tr>
</thead>
<tbody>
<tr>
<td>( uf_{20} )</td>
<td>0.985 ( 1.91 \times 10^8 )</td>
<td>0.0154 ( 3.091 )</td>
</tr>
<tr>
<td>( uf_{50} )</td>
<td>0.748 ( 3.50 \times 10^9 )</td>
<td>0.299 ( 93.459 )</td>
</tr>
<tr>
<td>( uf_{75} )</td>
<td>0.583 ( 7.41 \times 10^9 )</td>
<td>0.528 ( 490.166 )</td>
</tr>
<tr>
<td>( uf_{100} )</td>
<td>- - -</td>
<td>1.214 ( 10^6 )</td>
</tr>
<tr>
<td>( uf_{125} )</td>
<td>- - -</td>
<td>1.527 ( 10^6 )</td>
</tr>
<tr>
<td>( uf_{150} )</td>
<td>- - -</td>
<td>1.20 ( 10^7 )</td>
</tr>
<tr>
<td>( uf_{175} )</td>
<td>- - -</td>
<td>1.257 ( 10^7 )</td>
</tr>
<tr>
<td>( uf_{200} )</td>
<td>- - -</td>
<td>0.991 ( 2.44 \times 10^8 )</td>
</tr>
<tr>
<td>( uf_{225} )</td>
<td>- - -</td>
<td>0.994 ( 2.43 \times 10^8 )</td>
</tr>
<tr>
<td>( uf_{250} )</td>
<td>- - -</td>
<td>0.992 ( 2.43 \times 10^8 )</td>
</tr>
</tbody>
</table>

![Fig. 10: The ERT-ECDF curves for the SATLib instances: the fraction of instances of a given dimension \( s \) solved over their empirically determined expected runtime.](image)
(1+1)-FFA₀ algorithm we applied is not competitive to the state-of-the-art even two decades ago [60]. We now want to investigate if FFA can be helpful when the base algorithm is already performing well and we will do so on an entirely different domain.

In a job shop scheduling problem (JSSP) [61] without preemption, there are $M$ machines and $N$ jobs. Each job must be processed by all machines in a job-specific sequence and has, for each machine, a specific processing time. The goal is to find assignments of jobs to machines that result in an overall shortest makespan, i.e., the schedule which can complete all the jobs the fastest. The JSSP is $\mathcal{NP}$-hard [61]. The objective values are positive integers since the processing times are integers. We obtain an upper bound $U_B$ needed for FFA as the sum of all processing times of all sub-jobs. We use the JSSP as an educational example in [62], where we discuss all of the following components (except FFA) in great detail.

A solution for the JSSP is encoded as permutation with repetition, as integer strings where each of the $N$ job IDs occurs exactly $M$ times [63]. Such an integer string $x$ is processed from front to end. When encountering job $i$, we know to which machine $j$ it needs to go next based on the job-specific machine sequence and on how often we already saw $i$ in $x$ before. We can start it on $j$ at a time which is the maximum of $1$ when the previous sub-job assigned to $j$ will finish and 2) when the previous sub-job of $j$ completes on its corresponding machine.

We develop a memetic algorithm [64] which retains the $\mu = 16$ best candidate solutions in its population and generates $\lambda = 16$ new strings in each step via recombination. Recombination proceeds similar to the solution decoding, but reads unprocessed sub-jobs iteratively from two parent strings (between which it randomly switches) and writes them to an offspring, while marking each processed sub-job in both parents as processed [62]. The $\lambda$ new strings each are refined with ten different local searches which, in each step, scans the single-swap neighborhood of the string in random order until it finds a makespan-improving move and applies it (or stops if none can be found).

The two algorithms we investigate differ only in what they do once this step is completed: The first, MA, now applies selection based on the objective values. In the FMA, on the other hand, the FFA table $H$ is updated by increasing the frequency counter of the corresponding objective value of each of the $\mu + \lambda$ solutions in the joint parent-offspring population. Selection chooses the $\mu$ solutions with the lowest frequency fitness value. FMA still uses the objective function $f$ in the local search and to break ties in FFA. It is therefore not invariant under bijective variations of $f$.

Our goal this time is to achieve the best possible result within five minutes of runtime on an Intel Core i7 8700 CPU with 3.2 GHz and 16 GiB RAM under Java OpenJDK 13 on Ubuntu 19.04. This is very different from the previous goals of solving the well-known JSSP instances from the OR-Library [65, 66], namely the sets abz*, ft*, lb*, orb*, and ynn*, where all processing times are integers.

From Table III, we can find that MA can already discover
the best known solution (BKS) on 36 instances at least once and always on 27. FMA, however, can do so 46 and 32 times, respectively. FMA has better best, median, and mean results 37, 45, and 51 times, respectively, while the same is true for the MA only 8, 3, and 4 times. In other words, on 93% of the instances that are not already always solved to optimality by MA, FMA has a better mean result. The mean (median) result of FMA is better than the best result of MA in 17 (13) instances, while the opposite is never true. FMA has a smaller standard deviation in 48 cases, MA only in 4. We apply the two-sided Welch’s t-test to the results on the 49 instances where the algorithms have different mean results with non-zero standard deviations. FMA performs significantly better than MA on 25 of them at a significance level of $\alpha = 0$. Such high significance is a very strong result at only 11 runs. The opposite is true only on swv11, even if we set $\alpha = 0.1$.

Neither MA nor FMA can outperform the state-of-the-art on the JSSP, but they are not very far off, at least if we consider result quality only: The basic MA obtains better best (mean) results than the GWO proposed in [79] (2018) in 16 (21) of the 39 instances for which results are provided, while the opposite is never (once) true. While the HPSAQ [80] (2016) has better mean (best) result quality in 23 (12) of 48 comparable cases, FMA scores even in the rest, while having better mean solution quality on 4 instances. It also 13 times achieves better mean makespans (28 times worse ones) on the 63 common instances compared to the HIMGA [81] (2015), while its best solution is never better. On instances swv16 to swv20, which can be solved to the BKS by both MA and FMA, budgets of more than 16 min were used in [68] to find said BKSes. Still, the FMA is worse than, e.g., the algorithms in [82] (2016) and [75] (2015) on every common instance where it does not find the BKS.

In summary, we find that even in a more complicated setup based on an algorithm that already does not perform badly in comparison to recent publications, FFA can lead to a significant performance improvement. This does not mean that other diversity improvement strategies, e.g., those from Section III, could not have improved the performance of the MA as well or even better. Still, together with the results on the MaxSat problems in Section IV-I and those in our earlier papers on FFA on domains such as Genetic Programming [1, 2], this adds evidence to the idea that FFA may not just be of purely academic interest.

V. CONCLUSIONS

In this paper, we plugged Frequency Fitness Assignment (FFA) into the most basic evolutionary algorithm, the (1+1)-EA, and applied the resulting (1+1)-FEA to several problems defined over bit strings of dimension $s$. On the one hand, we found that the (1+1)-FEA is slower than the (1+1)-EA on the OneMax, LeadingOnes, and Plateau functions. In our experiments with these problems, it seems to increase the mean runtime needed to discover the global optimum by a factor no worse than linear in the number of objective values or in $s$. On the other hand, FFA can seemingly decrease the mean runtime on the Trap, Jump, and TwoMax problems from exponential to the scale of $s^2 \ln s$. On the MaxSat problem and on the W-Model benchmark, the (1+1)-FEA very significantly outperforms the (1+1)-EA.

These results are surprising when considering the nature of FFA – being invariant under bijective transformations of the objective function, i.e., possessing the strongest invariance property known to us. FFA never compares objective values directly. An algorithm applying only FFA would exhibit the same performance on the objective function $f$ as on $g \circ f$, where $g$ could be an arbitrary encryption method (which we simulate by setting $g$ to the Md5 checksum routine in Section IV-G).

This realization is baffling. Two central assumptions of black-box optimization are that following a trail of improving objective values tends to be a good idea and that “nice” optimization problems should exhibit causality, i.e., small changes to a solution should lead to small changes in its objective value. Under FFA, neither assumption is used. As a result, properties such as causality, ruggedness, or deceptiveness of a fitness landscape may have little impact on the algorithm performance. Interestingly, this does seemingly not necessarily come at a high cost in terms of runtime. Instead of the cost of the invariance, the limitation of the method seems to be that it requires objective functions that can be discretized and do not take on too many different values.

We finally showed that FFA can be combined with “normal” optimization and plugged into more complex algorithms. We inserted it into the selection step of a memetic algorithm whose local search proceeds without FFA and works directly on the objective values. Here, an FFA variant purely works as population diversity enhancement mechanism and can improve the result quality that the algorithm produces on the JSSP within a budget of five minutes. Notably, while this algorithm does not belong to the state-of-the-art on the JSSP, it seems to be relatively close to it. Together with our results on the MaxSat problem, this means that FFA might even be helpful in cases bordering to practical relevance.

There are several interesting avenues for future work. First, we want to also plug FFA into other EAs, such as those in [8]. Second, a theoretical analysis of the properties of FFA could be both interesting and challenging, also from the perspective of black-box complexity. Third, using FFA is the only approach known to us that can solve encrypted optimization problems. This could open new types of applications in operations research, machine learning, and artificial intelligence. Fourth, on problems FFA leads to a slowdown. The question whether this slowdown is proportional to the problem dimension or to the number of possible different objective values deserves an investigation.

Finally, it may be possible to adapt ideas from the research on multi-armed bandits to implement an FFA-like approach: We envisage an Upper Confidence Bound [83]-like algorithm, where one solution per encountered objective value is preserved and treated as bandit arm. Playing an arm would mean to use the solution as input to mutation and the reward could be 1 if the offspring has a new objective value.

\*Of course on older hardware, but our Java implementation is not optimized.
ACKNOWLEDGMENTS

We acknowledge support by the National Natural Science Foundation of China under Grant 61673359, the Hefei Specially Recruited Foreign Expert Program, the Youth Project of the Anhui Natural Science Foundation 1908085QF285, the University Natural Science Research Project of Anhui KJ2019A0835, the Key Research Plan of Anhui 201904d07020002, the Major Project of the Research and Development Fund of Hefei University 19ZIR05ZDA, the Talent Research Fund of Hefei University 18-19RC26, and the DIM-RSFI N 2019-01 C19/1526 project. The first author especially wants to thank Dr. Carola Doerr for the chance to participate in the Dagstuhl Seminar 19431 with the topic Theory of Randomized Optimization Heuristics, which gave us the inspiration to investigate FFA on some basic scenarios where it may even be theoretically tractable.

REFERENCES


Thomas Weise obtained his Ph.D. 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 PostDoc, later as Associate Professor. In 2016, he moved to Hefei University (in Hefei) as Full Professor to found the Institute of Applied Optimization (IAO) at the School of Artificial Intelligence and Big Data. His research interests include metaheuristic optimization and algorithm benchmarking. He was awarded the title Hefei Specially Recruited Foreign Expert of the city Hefei in Anhui, China in 2019.

Zhize Wu obtained his Bachelor from the Anhui Normal University (Wuhu, Anhui, China) in 2012 and his Ph.D. 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 (IAO) of the School of Artificial Intelligence and Big Data at Hefei University as Lecturer. His research interest include image processing, neural networks, deep learning, machine learning in general, and algorithm benchmarking.

Xinlu Li received his Bachelor from the Fuyang Normal University (Fuyang, Anhui, China) in 2002, his M.Sc. degree in Computer Science from Anhui University (Hefei, Anhui, China) in 2009, and his Ph.D. degree in Computer Science from the TU Dublin (Dublin, Ireland) in 2019. His career at Hefei University started as Lecturer in 2009, Senior Lecturer in 2012, in 2018 he joined the Institute of Applied Optimization (IAO) and in 2019 he became Associate Researcher. From 2014 to 2018, he was Teaching Assistant in the School of Computing of TU Dublin. His research interest include Swarm Intelligence and optimization, algorithms which he applies to energy efficient routing protocol design for large-scale Wireless Sensor Networks.

Yan Chen received his Bachelor from the Anhui University of Science and Technology (Huainan, Anhui, China) in 2011, his M.Sc. degree from the China University of Mining and Technology (Xuzhou, Jiangsu, China) in 2014, and his doctorate from the TU Dortmund (Dortmund, Germany) in 2019. There, he studied spatial information management and modeling as a member of the Spatial Information Management and Modelling Department of the School of Spatial Planning. He joined the Institute of Applied Optimization (IAO) of the School of Artificial Intelligence and Big Data at Hefei University as 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.

© 2020 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.