Bet-based Evolutionary Algorithms: Self-improving Dynamics in Offspring Generation

Simon Reichhuber[a] and Sven Tomforde[b]
Intelligent Systems, University of Kiel, Hermann-Rodewald-Str. 3, Kiel, Germany

Keywords: Evolutionary Computation, Genetic Algorithms, Optimisation, Fitness Landscapes, Diversity, Bet-based Approach.

Abstract: Evolutionary Algorithms (EA) are a well-studied field in nature-inspired optimisation. Their success over the last decades has led to a large number of extensions, which are particularly suitable for certain characteristics of specific problems. Alternatively, variants of the basic approach have been proposed, for example to increase efficiency. In this paper, we focus on the latter: We propose to enrich the evolutionary problem with a self-controlling betting strategy to optimise the evolution of individuals over successive generations. For this purpose, each individual is given a betting parameter to be co-optimised, which allows him to improve his chances of "survival" by betting. We analyse the behaviour of our approach compared to standard procedures by using a reference set of complex functional problems.

1 INTRODUCTION

The field of Evolutionary Computing (EC) comprises four basic directions: evolutionary programming (EP) (Fogel et al., 1966), evolution strategies (ES) (Schwefel, 1965; Rechenberg, 1978), genetic algorithms (GA) (Holland, 1975), and genetic programming (GP) (Koza and Koza, 1992). Based on this original work, an integrated field of research has developed. The underlying processes have been successively developed and solutions for a wide range of problems have been proposed. Alternatively, novel approaches and improvements in the algorithmic logic were presented, which aim at a higher robustness (see, e.g., (Eiben and Smit, 2011)), the handling of noise and uncertainty (see, e.g., (Jin and Branke, 2005)), the consideration of security concerns (see, e.g., (Prothmann et al., 2009)) or the consideration of different objective functions (see, e.g., (Zelinka, 2015)).

In this paper, we look at an alternative approach to generating the next population. Combined with an elitist strategy and based on the objective of maintaining diversity, especially in dynamic problems, we extend the basic procedure by a possibility for each individual to temporarily increase their selection despite their currently limited fitness. The individuals are given an additional parameter with which they can invest a so-called 'bet' with regard to their fitness. This parameter should then also be the focus of the optimisation. Additionally, we suggest an approach in which another external population is evolved to optimally bet on another population.

The remainder of this paper is organised as follows: Section 2 introduces the basic terms and foundations of this paper. Since the field of EAs is very broad and heterogeneous, we determine a reference approach in Section 3. Afterwards, Section 4 introduces our novel bet-based approach. Section 5 analysis the behaviour of the approach in comparison to other approaches (especially the reference) by considering optimisation problems defined by a set of standard mathematical functions. Finally, Section 6 summarises the paper and gives an outlook to future work.

2 BACKGROUND

Holland et al. invented an optimisation technique that is nowadays known as the canonical version of Genetic Algorithms (GA) in 1975. Inspired by the natural evolutionary process, this stochastic optimisation technique uses selection, recombination, and mutation to evolve a set of binary solution strings denoted as the population. The members of the population are called chromosomes and are represented as vec-

[a] https://orcid.org/0000-0001-8951-8962
[b] https://orcid.org/0000-0002-5825-8915

Reichhuber, S. and Tomforde, S.
Bet-based Evolutionary Algorithms: Self-improving Dynamics in Offspring Generation.
DOI: 10.5220/0010345611921199
In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART 2021) - Volume 2*, pages 1192-1199
ISBN: 978-989-758-484-8

tors, where each entry is called a gene. Algorithm 1 represents one GA run: First, the initial population is randomly drawn from a uniform random distribution. Second, the fitness value of each chromosome in the population is calculated and based on this fitness value a subset of the population, namely the parents, are separated from the rest. These parents will now be recombinated and a new offspring generation is born. Finally, the parents and the new offspring generation are mutated with a small probability and both of them represent the next generation. After a certain generation is reached or the fitness value of the best chromosome exceeds a predefined threshold, the best found solution is returned.

Algorithm 1: The canonical GA algorithm.

```
P = Initialization();
t = 0;
while error(t) > τ do
    Fitness = CalculateFitness(P);
    Parents = Selection(P);
    Offspring = Recombination(Parents);
    P = Mutation(Offspring);
    t ← t + 1;
end
Result: x^ga(t), y^ga(t), t
```

Later, researchers expanded the canonical version to also deal with real-valued problems (Wright, 1991; Goldberg, 1991; Eshelman and Schaffer, 1993; Blanco et al., 2001). One key question was how to find an adaptation mechanism which is able to adjust the parameters of the EA during run. Here, the main focus was on the step-size representing the difference of a chromosome before and after a single mutation (Price, 1996; Azad and Hasançebi, 2014). Other authors refined one of the genetic operations, such as a uniform crossover (Syswerda, 1989) or arithmetic recombination types (Michalewicz, 2013). All these findings are improved optimisation algorithms, which try to maximise or minimise an objective function. For the purpose of generality, we focus on a wide variety of not necessarily continuous functions and introduce the optimisation problem formally. The optimisation problem is formulated as a constraint minimisation problem of functions $f : \mathbb{R}^d \to \mathbb{R}$:

$$\min_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x})$$
$$\text{where } a \le x_i \le b \qquad \text{for } i = 1,\ldots,d$$

For each of the test functions $f_1 - f_{23}$ found in (Yao and Liu, 1996; Yao and Liu, 1996; Cakar et al., 2011), we call the function value of the global minimum $y^* = \min_{\boldsymbol{x}\in[a,b]^d} f(\boldsymbol{x})$ and measure the error of the minimum value found by the genetic algorithm after the $t-th$ generation.

$$error(t) = |y^* - y^{ga}(t)|$$

Additionally, we observe the number of generations t^{ga} required to reduce the error up to a tolerance τ

$$t^{ga} = \min_{t\in\mathbb{N}}$$
$$\text{where } error(t) \le \tau$$

3 REFERENCE EXTENSIONS IN EVOLUTIONARY ALGORITHMS

Since the field of EAs is very broad and heterogeneous, we first determine a reference approach. This should then serve a basis for comparisons for the bet-based approach developed in the next section. Before we present the reference, we introduce basic EAs operators that we used as the baseline for comparison but also as starting point of the bet-based EAs. For comparison of our bet-based genetic algorithms, we introduce a baseline, based on the canonical genetic algorithm. Therefore, we describe the steps of the EA.

Initialise Population. The population P is uniformly chosen from the hypercube $[a,b]^d$. N chromosomes $\boldsymbol{x}_i \in P$ are drawn.

Fitness Function. Since all the functions are minimisation problems including negative output values, we define our fitness function as follows:

$$F(\boldsymbol{x}_i) = -(f(\boldsymbol{x}) - y_{min}(P)),$$

where $y_{min}(P) = \min_{\boldsymbol{x}\in P} f(\boldsymbol{x})$ is the worst fitness value.

Selection. The parent population is found with respect to the fitness values of the chromosomes. The naïve selection strategy that takes the top k chromosomes into account leads to a poor diversity of the next generations. Stochastic selection strategies solve the latter. We used the two selection strategies *remainder stochastic sampling* and *universal sampling*. The remainder stochastic sampling is based on (Holland, 1975; Goldberg, 1991; Blanco et al., 2001) on the relative fitness.

$$F_{rel}(\boldsymbol{x}_i) = \frac{F(\boldsymbol{x}_i)}{\overline{F}},$$

where $\overline{F} = \frac{1}{N}\sum_{i=1}^{N} F(\boldsymbol{x}_i)$ is the mean fitness value. Here, chromosomes $\boldsymbol{x_i} \in P$ with a relative fitness

larger or equal than 1, i.e. $F_{rel} \geq 1$ are copied $\lfloor F_{rel}(\boldsymbol{x}_i) \rfloor$ times to the parent population and the residual $F_{rel}(\boldsymbol{x}_i) - \lfloor F_{rel}(\boldsymbol{x}_i) \rfloor$ and all other fitness values below 1 represent the probability of copying the corresponding chromosome to the parent selection. Analogously, universal sampling is a selection strategy with replacement where chromosomes are selected according to the probability:

$$p(\boldsymbol{x}) = \frac{F(\boldsymbol{x}_i)}{\sum_{i=1}^{N} F(\boldsymbol{x}_i)}.$$

For both strategies, there are drawn N_P parents.

Offspring and Recombination. Recombination is done by applying a crossover of a pair of parents. This procedure is repeated until $N - N_P$ children has been created. As we want to focus on a diversity of the offspring generation, we choose the uniform crossover (Syswerda, 1989; De Jong and Spears, 1990) in all of our experiments. Specifically, for child 1 each gene is either taken from parent 1 or parent 2 with equal probability and the other child is given as the inverse of child 1 (see Algorithm 2).

Algorithm 2: Recombination procedure with uniform crossover.

```
Offspring = ∅;
while |Offspring| < N − N_P do
    Parent_1, Parent_2 = random_nonequal_tuple(Parents);
    if p_r ≤ uniform([0,1]) then
        Offspring_1, Offspring_2 = uniform_crossover(Parent_1, Parent_2);
        Offspring ← Offspring ∪ {Offspring_1, Offspring_2}
    end
end
Result: Offspring
```

Mutation. Finally, the next generation is obtained by applying mutation on the parents and their offspring. Three different mutation operations for real vectors are discussed, *random mutation* (Blanco et al., 2001), *non-uniform mutation* (Blanco et al., 2001), and *one-step mutation* (Eiben and Smith, 2015). After randomly choosing an amount of p_m genes out of all $N*d$ possible genes, the genes $\boldsymbol{x}[i]$ are mutated by replacing them with:

- *Random mutation:*
  $\boldsymbol{x}[i]'$ drawn uniformly from the range $[a,b]$.
- *Non-uniform mutation:*

$$\boldsymbol{x}[i]' = \begin{cases} \boldsymbol{x}[i] + \Delta(t, b - \boldsymbol{x}[i]) & \text{if } \gamma = 0 \\ \boldsymbol{x}[i] + \Delta(t, \boldsymbol{x}[i] - a) & \text{if } \gamma = 1 \end{cases},$$

  where γ is equally likely 0 or 1
  and $\Delta(t,y) = y\left(1 - r^{(1-t/g_{max})^{\kappa}}\right)$. κ controls the degree of dependency on the number of iterations g_{max}, s.t. the higher κ the more likely the values of Δ, which lie in the range $[0,y]$, are close to zero. In Figure 1 the influence of this parameter is visualised for different values $\kappa = 1,2,5$ over 100 generations. One drawback of the non-uniform mutation is that the number of iterations must be known a priori. Therefore, this mutation can only be applied for experiments with a fixed number of generations.
- *One-step mutation:*
  Based on the old gene $\boldsymbol{x}[i]$, $\boldsymbol{x}[i]'$ is drawn uniformly from the interval $[\max\{a, \boldsymbol{x}[i]-1\}, \min\{b, \boldsymbol{x}[i]+1\}]$

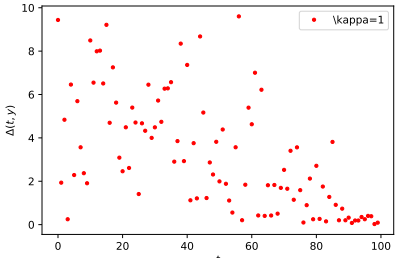
Elitism. Only, a small amount of the best chromosomes in the population, denoted as elites, are immune to the mutation operation and are added directly to the next generation. Therefore, the parameter r_{elite} defines the fraction of elites in the population.

4 NOVEL APPROACHES TO BET-BASED EVOLUTIONARY ALGORITHMS

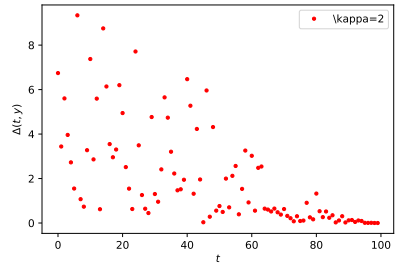
The following idea led us to the approach of bet-based agents: Instead of directly solving the minimisation problem, we convert it to a meta problem, which we call bet-based problem, meaning instead of directly solving the problem and providing a minimum $x^* = min_{x \in \mathcal{D}} f(x)$ the algorithm learns to place an optimal bet on a near-to-optimal solution. In every iteration the algorithm approximate to the solution by giving bets on $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N$. By answering the question *who bets about what*, we come up with two different betting mechanisms:

- *Self-betting EA (self BEA)*: Each chromosome of the population bets egocentrically on its own fitness increase. The bet parameters are encoded in the genes of the individual.
- *External-betting EA (external BEA)*: Another generation (*bet population*) bets on the population that solves the minimisation (*main population*.)

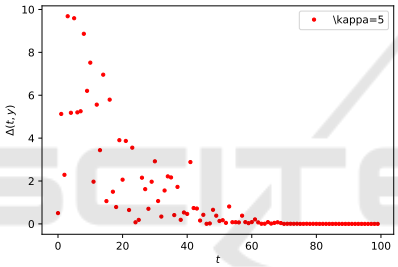
In the remainder of this chapter, we present the two approaches in detail.



(a) $\kappa = 1$



(b) $\kappa = 2$



(c) $\kappa = 5$

Figure 1: Influence of the value κ on the mutation through the iterations.

4.1 Self-betting EAs

We convert the canonical algorithm into a bet-based EA by introducing new learning parameters that are appended to the chromosomes. The first is used to determine the bet amount $c_{wager} \in [0,1]$ and the second one is used to control the influence of the fitness increase $c_{influence} \in [0,1]$. In case the function value of the chromosome changes after a generation, to the fitness value from Section 3 the bet outcome $\pi(\boldsymbol{x})$ is added, either in a positive ($\Delta f(\boldsymbol{x}_i) > 0$) or negative way ($\Delta f(\boldsymbol{x}_i) < 0$), with the function difference after one generation $\Delta f(\boldsymbol{x}_i) = \Delta f(\boldsymbol{x}_i^t) = f(\boldsymbol{x}^{t+1}) - f(\boldsymbol{x}^t)$.

$$
\begin{aligned}
F^{t+1}(\boldsymbol{x}_i) =& F^t(\boldsymbol{x}_i) + \pi(\boldsymbol{x}_i) \\
\pi(\boldsymbol{x}_i) =& c_{global} * f(\boldsymbol{x}_i) * \mathrm{sgn}(\Delta f(\boldsymbol{x}_i)) * (F^t(\boldsymbol{x}) * c_{wager}) \\
& * (\Delta f(\boldsymbol{x}_i) * c_{influence} + 1),
\end{aligned}
$$

where c_{global} is a constant that controls the bet influence, i.e. if $c = 1$ the maximal bet influence is as high as the old function value. The betting mechanism leads to higher oscillation of fitness values.

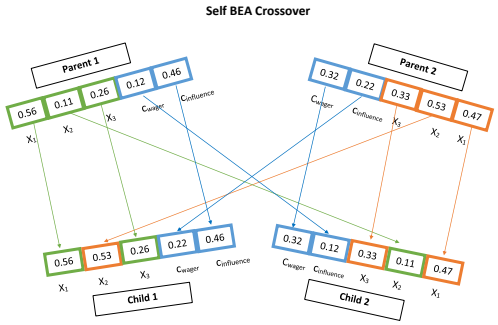


Figure 2: Crossover in self-betting GA procedure.

Our assumption is that the betting mechanism pushes the chance of survival for chromosomes that have recently found new global maximum candidates.

Recombination. As we do not want to mix bet genes with problem genes, we used a simple uniform crossover for each of these parameter sets as visualised in Figure 2.

Mutation. We use one of the three presented mutation strategies but shrink the mutation range for the bet parameters to the interval $[0,1]$.

Elitism. Especially for elite chromosomes, the outcome of the bet is always 0 since $\Delta f(\boldsymbol{x})$ is 0. To prevent weaker chromosomes from displacing the elite, the elite status depends on the raw functional value.

All other mechanisms are exactly as discussed in Section 3.

4.2 External-betting EAs

We call this approach external-betting EAs or simply external BEAs. Beside the main population, we create a second population, called the bet population, where each individual is encoded as chromosomes consisting of genes $\boldsymbol{b}_j[i] \in [-1,1], j = 1,\ldots,N^{bet}, \quad i = 1,\ldots,N^{main}$, where each gene represents a bet for a specific chromosome of the main population. A positive value indicates that the bet chromosome bets on a fitness increase of the chromosome of the main population and a negative value indicates a bet on the decrease of the same (cf. Figure 4). As visualised in Figure 3, for the bet population the bet procedure starts by placing bets and waiting for the outcome to recalculate the fitness value. The fitness values are updated as follows:

$$F^{bet}(\boldsymbol{b}_j) = \sum_{i=1}^{N^{main}} \boldsymbol{b}_j[i] * \Delta f(\boldsymbol{x}_i)$$

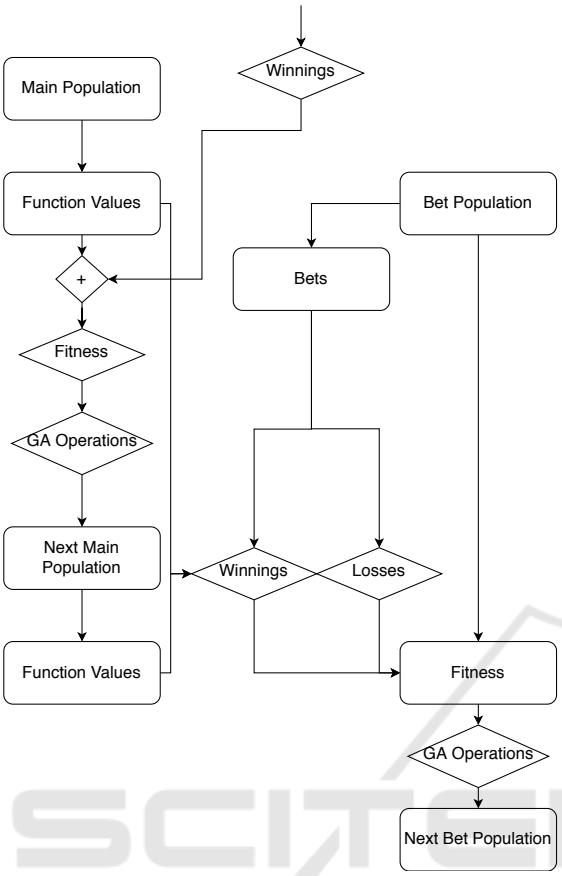


Figure 3: Bet procedure for external BEA.

On the other hand, the bets influences the fitness values of the next main population. More specific, only won bets on fitness increase can influence the next main population. The new fitness values are calculated as follows:

$$F^{t+1}(\boldsymbol{x}) = F^t(\boldsymbol{x}) + \pi(\boldsymbol{x})$$

$$\pi(\boldsymbol{x}) = c_{global} * f(\boldsymbol{x}) \frac{1}{|P_i^{bet,+}|} \sum_{\boldsymbol{b} \in P_i^{bet,+}} \boldsymbol{b}[i]$$

$$P_i^{bet,+} := \{\boldsymbol{b} \in P^{bet} | \boldsymbol{b}[i] > 0 \wedge \Delta f(\boldsymbol{x}_i) > 0\},$$

where $P_i^{bet,+}$ is the set of a all chromosomes in the bet population that have bet on an increase of the function value of the chromosome $\boldsymbol{x}_i$ of the main population and have won.

Since the bet procedure is based on a 1-to-1 correlation between the genes of a chromosomes in the bet population and the chromosomes in the main population, we have to reassign bet chromosome genes that are obsolete after the offspring have displaced parts of the main population. This is done by using a weighted average of the bets of the parents. Based on the number of genes that had been chosen from parent 1 in the uniform crossover n_{parent_1} and the number of genes of parent 2 n_{parent_2} the bet genes for the offspring are calculated as follows:

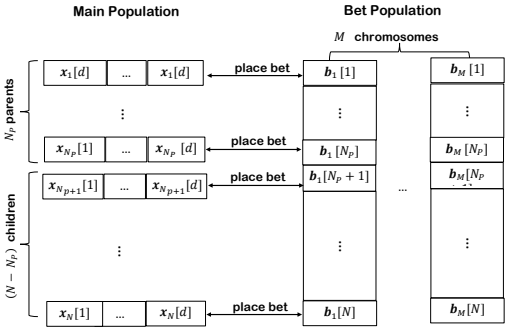


Figure 4: Betting in external BEA.

$$\boldsymbol{b}_j[i] = \frac{n_{parent_2}}{d} * parent_1(\boldsymbol{b}_j)[i] + \frac{n_{parent_2}}{d} * parent_2(\boldsymbol{b}_j)[i].$$

All other mechanisms are exactly as discussed in Section 3, except the fact that we have to apply each genetic operator double since we have two populations.

5 EXPERIMENTAL EVALUATION

For the experiments, we used the suggested numerical problems in (Cakar et al., 2011) as defined in Table 3 and choose a 30 dimensions whenever the dimension was adaptable. The first two dimensions of the functions are visualised in Figure 5. To solve the hyper parameter optimisation of all discussed parameters, we applied a grid search over all function by applying the grid parameters in Table 1.

Table 1: Evaluated parameter grid and optimal parameters found. Remainder stochastic sampling strategy is abbreviated as *rss* and universal stochastic sampling as *us*.

name	range	opt. param
selection	$[rsss, uss]$	$rsss$
mutation	$[rnd, non-uni., 1-step]$	$1-step$
parents ratio	$[0.25, 0.5, 0.75]$	0.25
mutation rate	$[0, 0.01, 0.02, 0.03]$	0.01
elite rate	$[0, 0.01, 0.02, 0.03]$	0.03
c_{global}	$[0.1, 0.3, 0.5]$	0.1

Here we observed the mean value over 5 runs, used a fixed number of 100 generations and a population size of 100. For the non-uniform mutation strategy, we choose $\kappa = 2$ and the crossover type was a uniform crossover. After ranking the parameter grid applied over a single function, we took the sum of all ranked

Table 2: Comparison of the algorithms EA, self BEA, and external BEA applied on 23 functions (see Table 3).

fun.	err(GA)	err(self BEA)	err(ext. BEA)
f1	17.74 ± 2.46	$\mathbf{17.61} \pm 1.48$	18.62 ± 0.92
f2	37.12 ± 4.01	40.1 ± 13.67	$\mathbf{35.12} \pm 6.57$
f3	0.0 ± 0.0	$\mathbf{0.0} \pm 0.0$	0.0 ± 0.0
f4	$\mathbf{12.18} \pm 5.1$	28.61 ± 8.17	21.31 ± 4.36
f5	$17e4 \pm 2.2e4$	$19e4 \pm 4.3e4$	$\mathbf{16e4} \pm 5.6e4$
f6	19.0 ± 1.9	$\mathbf{18.2} \pm 2.93$	19.0 ± 2.19
f7	21.93 ± 13.72	20.88 ± 11.7	$\mathbf{18.07} \pm 12.52$
f8	$\mathbf{2.6e3} \pm 1.6e3$	$2.9e3 \pm 0.3e3$	$2.7e3 \pm 1.5e3$
f9	232.99 ± 5.64	232.43 ± 9.77	$\mathbf{221.45} \pm 15.15$
f10	4.34 ± 0.16	13.67 ± 7.36	$\mathbf{4.12} \pm 0.14$
f11	$\mathbf{11.04} \pm 4.71$	102.33 ± 8.23	40.7 ± 12.94
f12	5.31 ± 1.0	6.25 ± 1.11	$\mathbf{4.94} \pm 1.21$
f13	8.49 ± 1.56	$\mathbf{6.49} \pm 0.9$	7.35 ± 1.56
f14	3.15 ± 3.0	3.9 ± 5.11	$\mathbf{1.97} \pm 2.42$
f15	$7e{-}4 \pm 3e{-}4$	$\mathbf{0.0} \pm 2e{-}4$	$0.0012 \pm 5e{-}4$
f16	$4e{-}4 \pm 4e{-}4$	$\mathbf{0.0} \pm 2e{-}4$	$4e{-}4 \pm 2e{-}4$
f17	$1e{-}4 \pm 1e{-}4$	$\mathbf{0.0} \pm 1e{-}4$	$1e{-}4 \pm 2e{-}4$
f18	$\mathbf{0.01} \pm 0.0071$	0.0192 ± 0.0098	0.0179 ± 0.0097
f19	0.0018 ± 0.0017	0.001 ± 0.0017	$\mathbf{0.0} \pm 0.0015$
f20	0.21 ± 0.03	0.25 ± 0.04	$\mathbf{0.17} \pm 0.06$
f21	3.17 ± 2.77	4.08 ± 3.35	$\mathbf{2.43} \pm 2.29$
f22	1.01 ± 0.29	1.99 ± 2.87	$\mathbf{0.75} \pm 0.22$
f23	0.84 ± 0.45	2.22 ± 2.99	$\mathbf{0.81} \pm 0.3$

grids into consideration and found the best parameter set as the minimum rank. For all the presented algorithms, i.e. EA, self BEA, external BEA, these optimal parameters were used for a longer test run, in order to compare them. Over 10 runs, 500 generations and a population size of 100, we analysed the mean of the remaining absolute error (cf. Table 2) and the mean progress over the generations(cf. Figure 6). As result, for the functions the reference EAs is better in 4 cases, the self BEAs in 6 cases, and the external BEAs in 12 cases. The high standard deviation of the algorithms self BEA and external BEA in Figure 6 indicates a high variance in the population but also shows that only in the long-term the two new algorithms are able to beat the reference.

Table 3: Numerical problems.

Function	Dim d	Ranges	Minimum value
$f_1(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2$	$30-100$	$-5.12 \le x_i \le 5.12$	$f_1(\mathbf{0}) = 0$
$f_2(\boldsymbol{x}) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} x_i$	$30-100$	$-10 \le x_i \le 10$	$f_2(\mathbf{0}) = 0$
$f_3(\boldsymbol{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_i\right)^2$	$30-100$	$-100 \le x_i \le 100$	$f_3(\mathbf{0}) = 0$
$f_4(\boldsymbol{x}) = \max_i \{ \lvert x_i \rvert, 0 \le i < n \}$	$30-100$	$-100 \le x_i \le 100$	$f_4(\mathbf{0}) = 0$
$f_5(\boldsymbol{x}) = \sum_{i=1}^{n-1} (100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$	$30-100$	$-30 \le x_i \le 30$	$f_5(\mathbf{1}) = 0$
$f_6(\boldsymbol{x}) = \sum_{i=1}^{n} \lfloor x_i + \frac{1}{2} \rfloor^2$	$30-100$	$-1.28 \le x_i \le 1.28$	$f_6(\boldsymbol{p}) = 0$ $-\frac{1}{2} \le p_i < \frac{1}{2}$
$f_7(\boldsymbol{x}) = (\sum_{i=1}^{n} (i+1) \cdot x_i^4) + \text{rand}[0,1[$	$30-100$	$-1.28 \le x_i < 1.28$	$f_7(\mathbf{0}) = 0$
$f_8(\boldsymbol{x}) = \sum_{i=1}^{n} -x_i \cdot \sin\left(\sqrt{\lvert x_i \rvert}\right)$	$30-100$	$-500 \le x_i \le 500$	$f_8(\mathbf{420.9687}) = -418.9829 * d$
$f_9(\boldsymbol{x}) = \sum_{i=1}^{n} (x_i^2 - 10 \cdot \cos(2\pi x_i) + 10)$	$30-100$	$-5.12 \le x_i \le 5.12$	$f_9(\mathbf{0}) = 0$
$f_{10}(\boldsymbol{x}) = -20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + e$	$30-100$	$-32 \le x_i \le 32$	$f_{10}(\mathbf{0}) = 0$
$f_{11}(\boldsymbol{x}) = \frac{1}{4000}\left(\sum_{i=1}^{n} x_i^2\right) + \left(\prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i+1}}\right)\right)$	$30-100$	$-600 \le x_i \le 600$	$f_{11}(\mathbf{0}) = 0$
$f_{12}(\boldsymbol{x}) = \frac{\pi}{n}\left(10 \cdot (\sin(\pi y_1))^2 + \sum_{i=1}^{n-1}\left((y_i - 1)^2 \cdot \left(1 + 10(\sin(\pi y_{i+1}))^2\right)\right) + (y_n - 1)^2\right) + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	$30-100$	$-50 \le x_i \le 50$	$f_{12}(-\mathbf{1}) = 0$
$f_{13}(\boldsymbol{x}) = 0.1\left((\sin(3\pi x_1))^2 + \sum_{i=1}^{n-1}\left((x_i - 1)^2 \cdot \left(\sin(3\pi x_{i+1})^2\right)\right) + (x_n - 1) \cdot \left(1 + (\sin(2\pi x_n))^2\right)\right) + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	$30-100$	$-50 \le x_i \le 50$	$f_{13}(\mathbf{1}) = 0$
$f_{14}(\boldsymbol{x}) = \left(\frac{1}{500} + \sum_{j=1}^{25}\left(j + \sum_{i=1}^{2}(x_i - a_{ij})^6\right)^{-1}\right)^{-1}$	2	$-65.54 \le x_i \le 65.54$	$f_{14}(-\mathbf{32}) = 0.9980$
$f_{15}(\boldsymbol{x}) = \sum_{i=1}^{11}\left(a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_2 + x_3}\right)^2$	4	$-5 \le x_i \le 5$	$f_{15}(.19, .19, .12, .14) = 3.42e{-}4$
$f_{16}(\boldsymbol{x}) = 4x_0^2 - 2.1x_0^4 + \frac{1}{3}x_0^6 + x_0x_1 - 4x_1^2 + 4x_1^4$	2	$-5 \le x_i \le 5$	$f_{16}(0.09, -0.71) = -1.032$
$f_{17}(\boldsymbol{x}) = (x_1 - \frac{5.1}{4\pi^2}x_0^2 + \frac{5}{\pi}x_0 - 6)^2 + 10 \cdot (1 - \frac{1}{8\pi}) \cdot \cos(x_0) + 10$	2	$-5 \le x_i \le 15$	$f_{17}(-3.14, 12.26) = 0.398$
$f_{18}(\boldsymbol{x}) = (1 + (x_0 + x_1 + 1)^2 \cdot (19 - 14x_0 + 3x_0^2 - 14x_1 + 6x_0x_1 + 3x_1^2)) \cdot (30 + (2x_0 - 3x_1)^2 \cdot (18 - 32x_0 + 12x_0^2 + 48x_1 - 36x_0x_1 + 27x_1^2))$	2	$-2 \le x_i \le 2$	$f_{18}(0, -1) = 3$
$f_{19} = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2\right)$	4	$0 \le x_i \le 1$	$f_{19}(.114, .556, .852) = -3.86$
$f_{20} = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2\right)$	4	$0 \le x_i \le 1$	$f_{20}(.201, .150, .477, .275, .311, .657) = -3.32$
$f_{21}(\boldsymbol{x}) = -\sum_{i=1}^{5}\left((x - a_i)^T(x - a_i) + c_i\right)^{-1}$	4	$0 \le x_i \le 10$	$f_{21}(\mathbf{4}) = -10.15$
$f_{22}(\boldsymbol{x}) = -\sum_{i=1}^{7}\left((x - a_i)^T(x - a_i) + c_i\right)^{-1}$	4	$0 \le x_i \le 10$	$f_{22}(\mathbf{4}) = -10.40$
$f_{23}(\boldsymbol{x}) = -\sum_{i=1}^{10}\left((x - a_i)^T(x - a_i) + c_i\right)^{-1}$	4	$0 \le x_i \le 10$	$f_{23}(\mathbf{4}) = -10.54$

Functions for f_{12}, f_{13}:

$$u(x,u,v,w) = \begin{cases} v(x-u)^w & \text{if } x > u \\ 0 & \text{if } x < -u \\ v(-x-u)^w & \text{if } -u \le x \le u \end{cases}$$

$y_i = 1 + \frac{1}{4}(x_i + 1)$

Vectors a, b for f_{15}:

$a = (.1957, .1947, .1735, .1600, .0844, .0627, .0456, .0342, .0323, .0235, .0246)$

$b_i^{-1} = (0.25, 0.5, 1, 2, 4, 6, 8, 10, 12, 14, 16)$

2×25 matrix a for f_{14}:

$$(a_{ij}) = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \dots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \dots & 32 & 32 & 32 \end{pmatrix}$$

4×3 matrices a, p and vector c for f_{19}

$$(a_{ij}) = \begin{pmatrix} 3 & 10 & 30 \\ .1 & 10 & 35 \\ 3 & 10 & 30 \\ .1 & 10 & 35 \end{pmatrix} \qquad (p_{ij}) = \begin{pmatrix} .3689 & .1170 & .2673 \\ .4699 & .4387 & .7470 \\ .1091 & .8732 & .5547 \\ .038150 & .5743 & .8828 \end{pmatrix}$$

$c = (1, 1.2, 3, 3.2)$

4×6 matrices a, p and vector c for f_{20}:

$$(a_{ij}) = \begin{pmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ .05 & 10 & 17 & .1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & .05 & 10 & .1 & 14 \end{pmatrix} \qquad (p_{ij}) = \begin{pmatrix} .1312 & .1696 & .5569 & .0124 & .8283 & .5886 \\ .2329 & .4135 & .8307 & .3736 & .1004 & .9991 \\ .2348 & .1415 & .3522 & .2883 & .3047 & .6650 \\ .4047 & .8828 & .8732 & .5743 & .1091 & .0381 \end{pmatrix}$$

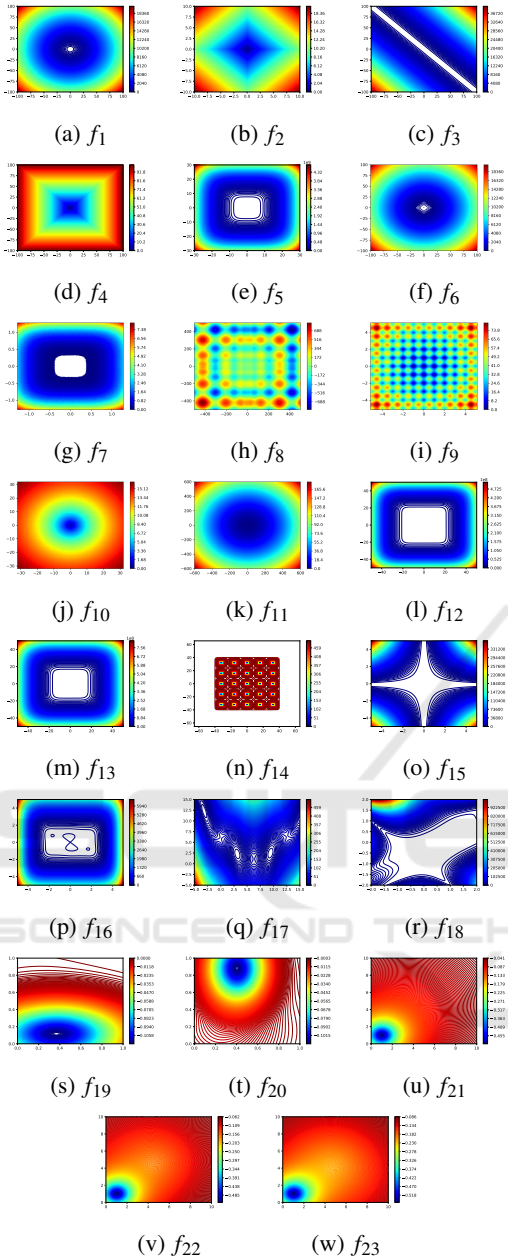
$c = (1, 1.2, 3, 3.2)$

10×4 matrix a and vector c for f_{21}, f_{22}, f_{23}

$$(a_{ij}) = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{pmatrix} \qquad c = (.1, .2, .2, .4, .4, .6, .3, .7, .5, .5)$$

6 CONCLUSIONS

We gave a short recap of the history of EP and presented a variety of operations of EAs. In perspective of these, we derived a reference EA for the comparison of two new bet-based EAs, called self-betting evolutionary algorithms (self BEAs) and external-betting evolutionary algorithms (external BEAs). These new bet-based algorithms are able to learn how to successfully bet, either bet on themselves (self BEAs) or on an external population (external BEAs). We presented experiments on 23 high dimensional test functions and showed that the new algorithms are able to beat the reference in the majority of all analysed test functions. As discussed in (Ghoreishi et al., 2017), the stopping criterion plays an important role for the practical applicability of EAs. Since our experiments provide evidence of an advantage of bet-based EAs over canonical EAs after a fixed number of function evaluations, an analysis of suitable stop criteria for bet-based EAs should be investigated in further research. We assume that for example choosing the k-iteration stop criterion, where the algorithm stops after a period of k iterations without improvement, the bet-based EAs may use this duration more efficiently. That is, the dynamic of bets pushes weak chromosomes on their way to the global optimum and let them dominate over chromosomes that are trapped in local optima. Additionally to the latter, for further research we want to extend the test procedure to learn how the designed algorithms can be refined. Also,



(a) f_1 (b) f_2 (c) f_3

(d) f_4 (e) f_5 (f) f_6

(g) f_7 (h) f_8 (i) f_9

(j) f_{10} (k) f_{11} (l) f_{12}

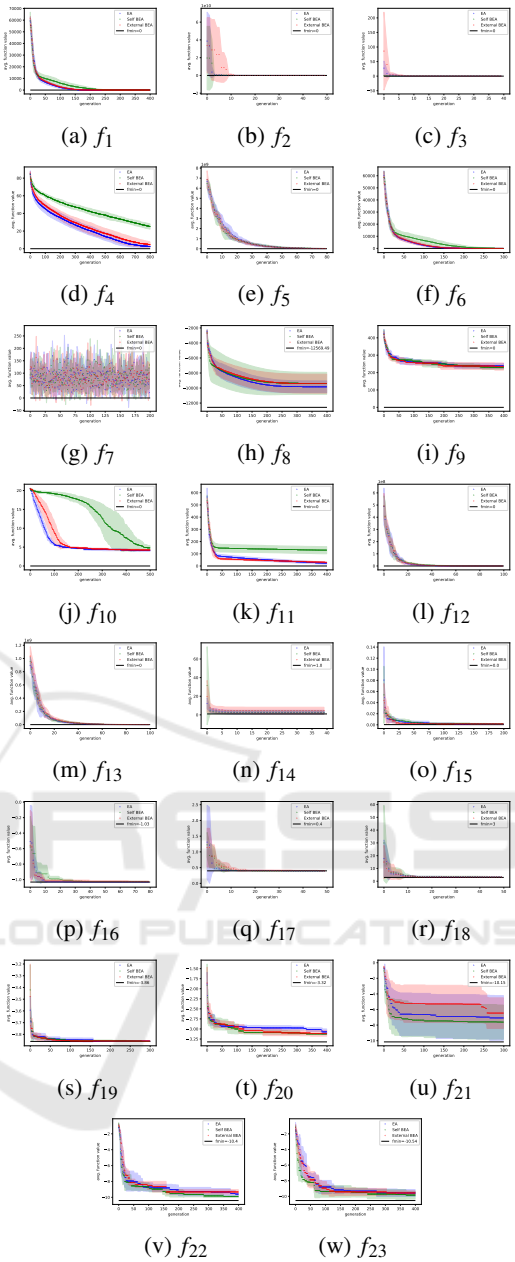
(m) f_{13} (n) f_{14} (o) f_{15}

(p) f_{16} (q) f_{17} (r) f_{18}

(s) f_{19} (t) f_{20} (u) f_{21}

(v) f_{22} (w) f_{23}

Figure 5: Functions specified in 3 found in (Yao et al., 1999). Only the first two dimensions are visualised.



(a) f_1 (b) f_2 (c) f_3

(d) f_4 (e) f_5 (f) f_6

(g) f_7 (h) f_8 (i) f_9

(j) f_{10} (k) f_{11} (l) f_{12}

(m) f_{13} (n) f_{14} (o) f_{15}

(p) f_{16} (q) f_{17} (r) f_{18}

(s) f_{19} (t) f_{20} (u) f_{21}

(v) f_{22} (w) f_{23}

Figure 6: Comparison of the algorithms EA, self BEA, and external BEA applied on 23 functions (see Table 3).

we want to adapt the betting mechanism to machine learning models that can be evolved.

REFERENCES

Azad, S. K. and Hasançebi, O. (2014). An elitist self-adaptive step-size search for structural design optimization. *Applied Soft Computing*, 19:226 – 235.

Blanco, A., Delgado, M., and Pegalajar, M. C. (2001). A real-coded genetic algorithm for training recurrent neural networks. *Neural networks*, 14(1):93–105.

Cakar, E., Tomforde, S., and Müller-Schloer, C. (2011). A role-based imitation algorithm for the optimisation in dynamic fitness landscapes. In *2011 IEEE Symposium on Swarm Intelligence*, pages 1–8.

De Jong, K. A. and Spears, W. M. (1990). An analysis of the interacting roles of population size and crossover in genetic algorithms. In *International Conference on Parallel Problem Solving from Nature*, pages 38–47. Springer.

Eiben, A. E. and Smit, S. K. (2011). Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31.

Eiben, A. E. and Smith, J. E. (2015). *Introduction to evolutionary computing*. Springer.

Eshelman, L. J. and Schaffer, J. D. (1993). Real-coded genetic algorithms and interval-schemata. In *Foundations of genetic algorithms*, volume 2, pages 187–202. Elsevier.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). Artificial intelligence through simulated evolution.

Ghoreishi, S. N., Clausen, A., and Joergensen, B. N. (2017). Termination criteria in evolutionary algorithms: A survey. In *Proceedings of the 9th International Joint Conference on Computational Intelligence - IJCCI,*, pages 373–384. INSTICC, SciTePress.

Goldberg, D. (1991). Real-coded genetic algorithms, virtual alphabets, and blocking. *Complex Syst.*, 5.

Holland, J. (1975). Adaptation in natural and artificial systems, univ. of mich. press. *Ann Arbor*.

Jin, Y. and Branke, J. (2005). Evolutionary optimization in uncertain environments-a survey. *IEEE Transactions on evolutionary computation*, 9(3):303–317.

Koza, J. R. and Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press.

Michalewicz, Z. (2013). *Genetic algorithms+ data structures= evolution programs*. Springer Science & Business Media.

Price, K. V. (1996). Differential evolution: a fast and simple numerical optimizer. In *Proceedings of North American Fuzzy Information Processing*, pages 524–527.

Prothmann, H., Branke, J., Schmeck, H., Tomforde, S., Rochner, F., Hähner, J., and Müller-Schloer, C. (2009). Organic traffic light control for urban road networks. *Int. J. Auton. Adapt. Commun. Syst.*, 2(3):203–225.

Rechenberg, I. (1978). Evolutionsstrategien. In *Simulationsmethoden in der Medizin und Biologie*, pages 83–114. Springer.

Schwefel, H.-P. (1965). Kybernetische evolution als strategie der experimentellen forschung in der stromungstechnik. *Diploma thesis, Technical Univ. of Berlin*.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In *Proceedings of the 3rd international conference on genetic algorithms*, pages 2–9.

Wright, A. H. (1991). Genetic algorithms for real parameter optimization. In *Foundations of genetic algorithms*, volume 1, pages 205–218. Elsevier.

Yao, X. and Liu, Y. (1996). Fast evolutionary programming. *Evolutionary programming*, 3:451–460.

Yao, X., Liu, Y., and Lin, G. (1999). Evolutionary programming made faster. *IEEE Transactions on Evolutionary computation*, 3(2):82–102.

Zelinka, I. (2015). A survey on evolutionary algorithms dynamics and its complexity–mutual relations, past, present and future. *Swarm and Evolutionary Computation*, 25:2–14.