THE COMPARATIVE PERFORMANCE OF A TWO-RATE SELF-ADJUSTING GENETIC ALGORITHM AND A SELF-ADAPTING EVOLUTION STRATEGY

Arsen Ignatosyan
s4034538
a.ignatosyan@umail.leidenuniv.nl

Quirijn B. van Woerkom
s3366766
woerkom@mail.strw.leidenuniv.nl

December 15, 2023

ABSTRACT

In this paper, we examine the efficacy in a fixed-budget context of a version of the two-rate self-adjusting $(1 + \lambda)$-genetic algorithm proposed by Doerr et al. (2019a) generalised to a $(\mu + \lambda)$-genetic algorithm, as compared to a more classical self-adapting evolution strategy. In particular, we examine their performance on the one-dimensional Ising ring and low-autocorrelation binary sequences problems. We first describe the generalised version of the two-rate genetic algorithm to an arbitrary number of parents, as well as a correlated self-adapting evolution strategy, and perform global parameter searches for the best-performing algorithm parameters of either on the two reference problems. On the basis of these results, we conclude that (1) the two-rate genetic algorithm outperforms the evolution strategy on discrete, highly local problems, that (2) the two-rate algorithm outperforms several state-of-the-art reference optimisers on the one-dimensional Ising ring for particular configurations, but fails to do so in general for the low-autocorrelation binary sequences problem and (3) that some implicit assumptions made by Doerr et al. (2019a) hold, even upon generalisation of the algorithm. We finish by discussing some potential avenues of future research for the two-rate genetic algorithms that parallel so-called "fast" genetic algorithms that have already found success in prior literature.

1 Introduction

Evolutionary algorithms provide a means by which problems can be solved that are particularly resilient to other forms of analysis that would require a more restricted formulation (e.g. those where smoothness, continuity or differentiability are needed); in particular, they are well-suited to optimisation of functions that have no or an ill-defined derivative or global evaluation of which is too expensive to achieve in a reasonable timeframe on modern hardware (see e.g. Bäck et al. 2013; Eiben & Smith 2015; Emmerich et al. 2018). This class of algorithms concerns itself with a procedure inspired by the process of evolution in nature, and in general comprises three elements: (1) some notion of "fitness", or an objective function to be maximised, (2) a selection operator that selects parents for reproduction based on this fitness and (3) a process of mutation, which produces the "genetic makeup" of the offspring population from the parents (see e.g. De Jong 1988 for a discussion of this in the context of more classical genetic algorithms or Bäck et al. 2013 for an evolution strategy-specific treatment).

In this paper, we examine the performance of two types of evolutionary algorithms: a genetic algorithm (GA) with self-adjusting mutation rate, and a self-adapting evolution strategy (ES). In particular, we examine their performance when applied to two test-problems that have the properties discussed above for which evolutionary algorithms are well-suited: the one-dimensional Ising ring problem (see e.g. Fischer & Wegener 2004; Sudholt 2005), and the problem of low-autocorrelation binary sequences or LABS-problem (see e.g. Packebusch & Mertens 2016 for an overview or Militzer et al. 1998; Mow et al. 2015 for an evolutionary perspective). For a short description of these problems, the reader is referred to Doerr et al. (2019b); we will also adopt their notation of F18 for the LABS-problem and F19 for the one-dimensional Ising ring model where convenient. For the Ising model in general, the (unique) optimal values

are known to be 0^n and 1^n (i.e. only zeroes or only ones) for all connected graphs, and so this also holds for the Ising ring in particular (Fischer & Wegener, 2004); the LABS-problem is not known to have an analytically constructable optimum, but optimum values have been found by exhaustive search for dimensions of 66 and lower (Packebusch & Mertens, 2016).

2 Algorithms

We shall begin by giving a full overview of the two algorithms used. Sec. 2.1 will give an overview of the genetic algorithm: why this particular form of the algorithm was chosen, a description of the implementation and an overview of the parameter space chosen for further exploration. Sec. 2.2 gives a parallel discussion of these same points but for the evolution strategy.

2.1 A $(\mu + \lambda)$-genetic algorithm with self-adjusting mutation rate

One of the prime advantages of evolutionary algorithms in general is the fact that they require no more knowledge about the objective function than how to compute it - in particular, this means that evolution-based methods are very useful for optimisation of non-smooth functions or other functions whose derivatives are ill-posed or difficult/expensive to evaluate (e.g. Emmerich et al. 2018). Hence, genetic algorithms can be more generally applied than most other optimisation algorithms; in pursuit of this generality, we choose to employ a self-adjusting genetic algorithm that dynamically alters its mutation rate. This removes the need to pre-determine the (evolution of the) mutation rate, the optimal value of which can be (heavily) dependent on the objective function (Schaffer et al., 1989).

2.1.1 Supporting arguments for this form of the genetic algorithm

There are many examples of self-adjusting genetic algorithms in literature (e.g. Bäck & Schütz 1996; Doerr & Doerr 2018; Doerr et al. 2019a; Rodionova et al. 2019; Srinivas & Patnaik 1994), and so it is necessary to pick our approach in a well-supported manner. In particular, we choose to use a version of the $(1 + \lambda)$-genetic algorithm by Doerr et al. (2019a) generalised to a $(\mu + \lambda)$-genetic algorithm. Our reasoning for this is three-fold: (1) this self-adjusting approach does not require tuning of sensitive parameters such as the algorithm used by Srinivas & Patnaik (1994); (2) using a population-level mutation rate rather than individual rates (as used by e.g. Bäck & Schütz 1996) sets the genetic algorithm apart from the evolution strategy, which otherwise become sufficiently similar that consideration of two algorithms becomes unwarranted; (3) it does not (necessarily) involve crossover, unlike the algorithm by Doerr & Doerr (2018). The latter argument might seem counterintuitive (why should this be a good thing?), but as discussed by Doerr & Doerr (2018) the use of crossover has limited theoretical backing in a general sense. As the Ising model on a ring (which is one of our test problems) has been considered the prime example of a problem where crossover performs well due to the building block-structure of the problem (Fischer & Wegener, 2004)[1], we fear that including crossover would unfoundedly ascribe our optimisation algorithm a better performance than it will attain in a more general sense, i.e. when used with other problems. Indeed, preliminary experimentation indicates that crossover seems to yield a distinct advantage on the one-dimensional Ising ring problem, while admitting little to no improvement for the LABS problem.

Finally, the choice for $+$-selection rather than ,-selection is not as well-founded; Doerr et al. (2019a) do not give any particularly convincing arguments for this choice in the $(1 + \lambda)$-version of the two-rate GA either. We recommend that future work explore this possibility, but as the self-adjusting nature of this algorithm already serves to push it out of local optima, we do not see a change to ,-selection as necessary.

2.1.2 Description of the two-rate genetic algorithm

The algorithm is presented in pseudocode in Algorithm 1. Note that in line with the notation used by Doerr et al. (2019a), we present the mutation probability as a genome-wide mutation rate r, corresponding to the expected number of bits to flip. The bit-wise mutation rate is of course recovered upon division by the dimension of the problem. As addition to the algorithm presented in Algorithm 1, we will shortly discuss in particular the two-rate mutation process and the selection operator that is used.

The main feature of the two-rate algorithm is the splitting of the offspring population into two sub-populations; one of these is mutated with a lower mutation rate r/F, the other with a higher rate rF. With 50% probability (in line with Doerr et al. (2019a)), the next generation's rate r is taken to be that of the sub-population that produced the best-performing individual; else, the rate is determined uniformly at random from $\{r/F, rF\}$. If the resulting rate r were to fall outside the interval $[F, n/(2F)]$ (where n is the dimension), which would mean that the bit-wise mutation

[1] In fact, this property holds for more general versions of the Ising model (Sudholt, 2005).

probability for one of the sub-populations of the next generation would fall outside the "meaningful" range $[1/n, 1/2]$, the rate is instead clipped to the corresponding endpoint.

As for the selection operator; we perform linearly proportional roulette wheel selection, with the lowest-performing individual assigned 0 probability of selection. Effectively, this a-priori eliminates the lowest-performing parent except in the $\mu = 1$ case; we choose this algorithm setup as it will easily allow later generalisations into algorithms with a variable selection probability offset if desired.

2.1.3 Parameter exploration setup

The problem-defining budget and dimension are fixed to $B_{\max} = 5000$ and $n = 50$, respectively; we do not yet fix the four algorithm parameters $(r_{\text{init}}, F, \mu, \lambda)$. The first two of these algorithm parameters we explore in particular to test the claim by Doerr et al. (2019a) that their algorithm does not have parameters (as they had set in particular $F = 2$ and $r_{\text{init}} = F = 2$); if these parameters do not affect the performance of the algorithm much, we may conclude that their claim was justified. The latter two are of course the population parameters that one would explore to tune a classic genetic algorithm with fixed or pre-determined mutation rates.

Initial exploration shows that values of F between 1 and 1.1 are most effective; this is in line with the results by Doerr et al. (2019a), who find that between the values of $F = 1.2, 1.5, 2$ the lowest out-performs the others. We therefore choose the to explore 10 equidistant values of $F \in [1.01, 1.1]$.

For the initial mutation rate, initial trials show that all reasonable values are acceptable at times, except for very low initial mutation rates of $r \approx 1$, which allow very little diversity early on. For this reason, we explore values over $r_{\text{init}} \in \{2, 3, 4, 5, 10, 15, 20, 25, 30\}$; while it might seem counterintuitive to allow values over $n/2$, this allows a greater mutation probability for the lower-mutation sub-population to be passed on to the next generation. This could potentially promote diversity initially even if a well-performing individual is already found in the first generation.

The parent and offspring population sizes μ and λ are chosen more heuristically; initial trials show that high values of μ and λ (greater than 5 and 10, respectively) do not allow reaching useful fitness values, presumably because they do not undergo sufficiently many generations for the mutation rate to settle before the function evaluations budget is exceeded. Hence, we choose to explore $\mu \in \{1, 2, 3, 4, 5\}$ and $\lambda \in \{2, 3, 4, 5, 6, 7, 8, 9, 10\}$.

2.2 Evolution Strategy

Evolution strategies are classical variants of evolutionary algorithms which are frequently used to heuristically solve optimization problems, in particular in continuous domains Martí et al. (2018). The pseudo-code of the self-adaptive evolution strategy defined is presented in the Algorithm 2. Note that there are various mutation types, recombination types and selection strategies and based on these the strategy parameters of each individual changes.

2.2.1 Mutation Types

One σ Mutation: For one step size mutation each individual $\mathbf{a} = ((x_1, \ldots, x_n), \sigma)$ has n search variables with one step size σ. In this case the mutation of the search variables and step size is done in the following manner:

$$
\begin{aligned}
\mathbf{a} &= ((x_1, \ldots, x_n), \sigma) \\
\mathbf{a}' &= ((x'_1, \ldots, x'_n), \sigma') \\
\sigma' &= \sigma \exp(\tau_0 \mathcal{N}(0, 1)) \\
x'_i &= x_i + \sigma' \mathcal{N}(0, 1) \quad \forall i \in [n]
\end{aligned}
$$

where $\tau_0 = \frac{1}{\sqrt{n}}$.

Individual σ_i Mutation: For individual step size mutation each individual $\mathbf{a} = ((x_1, \ldots, x_n), (\sigma_1, \ldots, \sigma_n))$ has n search variables x_i and each one of these has its corresponding step size parameter σ_i, $\forall i \in [n]$. The mutation of search variables and individual step sizes is done the following way:

$$
\begin{aligned}
\mathbf{a} &= \left((x_1, \ldots, x_n), (\sigma_1, \ldots, \sigma_n)\right) \\
\mathbf{a}' &= \left((x'_1, \ldots, x'_n), (\sigma'_1, \ldots, \sigma'_n)\right) \\
g &\sim \mathcal{N}(0,1) \\
\sigma'_i &= \sigma_i \exp\left(\tau' g + \tau \mathcal{N}(0,1)\right) \quad \forall i \in [n] \\
x'_i &= x_i + \sigma'_i \mathcal{N}(0,1) \quad \forall i \in [n]
\end{aligned}
$$

where $\tau' = \frac{1}{\sqrt{2n}}$ and $\tau = \frac{1}{\sqrt{2\sqrt{n}}}$.

Correlated Mutation: For correlated mutation each individual $\mathbf{a} = \left((x_1, \ldots, x_n), (\sigma_1, \ldots, \sigma_n), \left(\alpha_1, \ldots, \alpha_{n(n-1)/2}\right)\right)$ has n search variables x_i and each one of these has its corresponding step size parameter $\sigma_i, \forall i \in [n]$. Besides this another strategy parameter is added, $\alpha_j, \forall j \in [n(n-1)/2]$, which is a rotation angle for each pair of coordinates. In this case the mutation of the search variables and strategy parameters is done in the following fashion:

$$
\begin{aligned}
\boldsymbol{a} &= \left((x_1, \ldots, x_n), (\sigma_1, \ldots, \sigma_n), \left(\alpha_1, \ldots, \alpha_{n(n-1)/2}\right)\right) \\
\boldsymbol{a}' &= \left((x'_1, \ldots, x'_n), (\sigma'_1, \ldots, \sigma'_n), \left(\alpha'_1, \ldots, \alpha'_{n(n-1)/2}\right)\right) \\
g &\sim \mathcal{N}(0,1) \\
\sigma'_i &= \sigma_i \exp\left(\tau' g + \tau \mathcal{N}(0,1)\right) \quad \forall i \in [n] \\
\alpha'_j &= \alpha_j + \mathcal{N}(0,\beta) \quad \forall j \in [n(n-1)/2] \\
\vec{x}' &= \vec{x} + \mathcal{N}\left(\mathbf{0}, \boldsymbol{C}'\right) \quad \forall i \in [n]
\end{aligned}
$$

where $\mathbf{C}^{\frac{1}{2}} = \left(\prod_{i=1}^{n-1} \prod_{j=i+1}^{n} R\left(\alpha_{ij}\right)\right) \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{bmatrix}$, $\mathbf{C} = \mathbf{C}^{\frac{1}{2}} \mathbf{C}^{\frac{1}{2}\top}$ and $\boldsymbol{C}'$ is calculated based on the computed α'_j and σ'_i. The learning rates $\tau' = \frac{1}{\sqrt{2n}}$ and $\tau = \frac{1}{\sqrt{2\sqrt{n}}}$ are the same as for individual mutation and $\beta = \frac{\pi}{36}$.

2.2.2 Recombination Types

Discrete Recombination: In the case of simple discrete recombination, the offspring is generated from two randomly selected parents, where each search variable i of the offspring is randomly selected from i-th search variable of the first or second parent. There is also a variation of discrete recombination, called global discrete recombination, where all parents are considered for the offspring, and each search variable i of the offspring is chosen randomly from all parent's i-th search variable.

Intermediate Recombination: In the case of simple intermediate recombination, the offspring is generated from two randomly selected parents, where each search variable i is the average of i-th values of the selected parents. There is also a variation of intermediate recombination, called global intermediate recombination, where all parents are considered for the offspring, and each search variable i of the offspring is the average of all parent's i-th search variable.

2.2.3 Selection Types

There are two selection types $(\mu + \lambda)$ and (μ, λ). $(\mu + \lambda)$ selection means that from μ parents λ offsprings are produced at each time-step and the best set of μ individuals are chosen to be parents in the next time step from the $\mu + \lambda$ individuals combined. (μ, λ) selection means that from μ parents λ offsprings are produced at each time-step and the best set of μ individuals are chosen to be parents in the next time step from the λ offsprings only.

2.2.4 Parameter Exploration Setup

The defined maximum budget and dimension are fixed to $B_{\max} = 5000$ and $n = 50$. The population is initialized to be of size μ, with each individual being $x \in \{0,1\}^n$. As evolution strategy works with individuals represented as real-valued vectors, consisting of object variable vectors $\vec{x} \in \mathbb{R}^n$, then an encoding and decoding technique is needed to convert from bit vectors to real-valued vectors and vice versa. To do this an encoding technique is used , that takes as an input a $n = 50$ dimensional vectors of bit strings and outputs $m = 10$ dimensional real-valued vectors, by converting

each 5-chunk of bits into a real value. The decoding is done by taking the 10 dimensional real-valued vector, rounding the values to the nearest integer value, fitting the values in the boundaries of $[0, 2^5)$, so that each value can be converted back to an bit string of size 5. The encoding and decoding are only performed on the object variables of the population and the encoding is conducted before mutation operation, decoding is done after mutation operation. All of the mutation types are implemented and experimented with. For recombination operation discrete recombination is chosen for object variables, and global intermediate recombination is chosen for strategy parameters.

For experimentation, $\mu \in \{1, 2, 5, 10, 12, 15\}$, $\lambda \in \{1, 2, 5, 10, 20, 50, 100\}$, all three of mutation types, both selection types and initial $\sigma \in \{0.5, 1, 1.5, 2\}$ are chosen, and all of the combinations for these settings are run and evaluated.

Algorithm 1: The two-rate self-adjusting $(\mu + \lambda)$-genetic algorithm

Input : Number of parents μ
Number of offspring λ
Initial bit-flip rate r_{init}
Probability modifier F
Evaluation budget $B_{\max}$
Dimension n

Termination : The algorithm terminates when the function evaluations budget is exceeded.

Select μ initial parents $\{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x}_\mu\}$ uniformly at random from $\{0,1\}^n$;
Set the mutation rate $r \leftarrow r_{\text{init}}$;
Set the number of function evaluations $B \leftarrow 0$;
while B $\leq B_{\max}$ **do**
  Produce λ offspring $\{\mathbf{x}_{\mu+\mathbf{1}}, \mathbf{x}_{\mu+\mathbf{2}}, \ldots, \mathbf{x}_{\mu+\lambda}\}$ from the parents through linearly proportional roulette wheel selection;
  Mutate the first $\lfloor \lambda/2 \rfloor$ offspring $\{\mathbf{x}_{\mu+\mathbf{1}}, \mathbf{x}_{\mu+\mathbf{2}}, \ldots, \mathbf{x}_{\mu+\lfloor \lambda/\mathbf{2} \rfloor}\}$ with bit-wise probability $r/(Fn)$;
  Mutate the remaining offspring $\{\mathbf{x}_{\lfloor \lambda/\mathbf{2} \rfloor+\mathbf{1}}, \mathbf{x}_{\lfloor \lambda/\mathbf{2} \rfloor+\mathbf{2}}, \ldots, \mathbf{x}_{\lambda+\mu}\}$ with bit-wise probability Fr/n;
  Evaluate the performance of the mutated offspring;
  Increment $B \leftarrow B + \lambda$;
  Select from the population $\{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x}_{\mu+\lambda}\}$ the best performing μ to be the new parents $\{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x}_\mu\}$;
  Perform one of the following with probability 1/2;
  (1) Set r to the mutation rate that the best-performing individual was created with;
  (2) Set r to either r/F or rF with probability 1/2;
  Clip r into the interval $[F, n/(2F)]$;
end

3 Experimental Results

As the two algorithms have been tested separately and the numerical experiments involved have produced a great deal of data, we will present separately the results for the genetic algorithm in Sec. 3.1 and for the evolution strategy in Sec. 3.2.

3.1 Results for the two-rate genetic algorithm

We must discuss the results for the two-rate genetic algorithm in two parts: we shall first present the results for the global parameter search in Sec. 3.1.1 (which are too extensive to present in a plot), and then show a more detailed analysis of particular well-performing parameter settings in Sec. 3.1.2.

3.1.1 Global results

The results of the global parameter exploration are shown for the LABS problem in Tab. 1 and for the Ising ring in Tab. 2; the first row of these tables contains the global population statistics and the parameters of the globally best-performing setting, with consequent rows detailing the performance conditional to given parameter values and the best-performing parameter settings for the other parameters corresponding to them. While a run was also performed on the ONEMAX problem for verification purposes, the results have not been appended in the interest of brevity. It should be noted, however, that the results for this problem agree well with the results for the two test problems presented in Tabs. 1 and 2.

Algorithm 2: Self-Adaptive Evolution Strategy

Input : Number of parents μ
Number of offspring λ
Evaluation budget $B_{\max}$

Termination : The algorithm terminates when the function evaluations budget is exceeded.

$t \to 0$
Initialize $(P(t))$ which includes μ individuals;
Evaluate $(P(t))$;
Set the number of function evaluations $B \leftarrow \mu$;
while B $\leq B_{\max}$ **do**
  Produce λ offsprings $P'(t) \leftarrow \text{Recombine}(P(t))$;
  Mutate all λ offsprings $P''(t) \leftarrow \text{Mutate}\,(P'(t))$;
  Select the best performing μ new parents for the next iteration
  $P'''(t) \leftarrow \text{Select}\,(P''(t) \cup Q)\,; \quad \triangleright Q \in \{\emptyset, P(t)\}$
  Evaluate $(P'''(t))$;
  $P(t+1) \leftarrow P'''(t)$;
  Increment $B \leftarrow B + \lambda$;
  $t \leftarrow t + 1$;
end

3.1.2 Performance of top parameter configurations

Based on the global results, several parameter configurations are selected for further study; IOHanalyzer is unfortunately not able to process the amount of data our exploratory runs have generated, and the resulting plots for this amount of data would not be insightful to start with.

For r_{init} and F we see that they do not have a strong effect on the performance in terms of best-reached fitness: for both test problems, the standard deviation among the best obtained fitnesses is lower for lower values of both parameters. This suggests that while r_{init} and F do not affect the optimisation performance directly, they rather influence the consistency of the algorithm. Consequently, we choose $r_{\text{init}} = 2$ and $F = 1.01$ for further study; so as to not discard the optimum realisations in our runs, which have $r_{\text{init}} = 20$ and $F = 1.05$ for the LABS problem and $r_{\text{init}} = 3$ and $F = 1.07$ for the LABS problem, we also include those.

The parameters μ and λ do show a strong performance influence in both problems, however; $\mu = 1$ and $\mu = 2$ outperform the other values significantly, and $\lambda \in \{8, 9, 10\}$ also do significantly better than lower values. Of particular note is the fact that while $\lambda = 2$ performs relatively poorly for both problems in general, the particular combination $\mu = 1$ with $\lambda = 2$ provides the global best performance in either case. This combination thus also warrants further examination.

The algorithms selected for further examination are presented in Tab. 3, with the labels those are marked with in the plots that follow as well as the area-under-the curve of the aggregated empirical cumulative distribution, a performance metric generated by IOHprofiler (see Doerr et al. 2019b). Additionally, we include a set of reference data produced by reference optimisation algorithms, provided by H. Wang (personal communication, 2023). The aggregated empirical cumulative distributions for the algorithms applied to the two test problems are given in Figs. 1 and 2: these distributions describe the fraction of (run, target) pairs that exceed a given set of target values. For a precise definition, see Doerr et al. (2018); we use equispaced target values between 0.31 and 5.19 for F18, and between 16 and 50 for F19.

3.2 Results for the evolution strategy

An extensive set of experiments were done both for LABS and Ising ring problems, and as all of the experiments would not fit into the plots, the best performing ones are selected. The chosen algorithms and their respective performances can be seen in Figures 3 and 4. It can be seen that evolution strategy for these problems benefits from large λ and μ values. Also, for the mutation types, all of the algorithms implement correlated mutation, so that is the best one out of all three. For setting initial step sizes σ from the results it can be interpreted that small step sizes in give better results. All of these results for settings of evolution strategy are found through empirical testing. The setting of parameters for each chosen algorithm can be found in Table 4 with their respective AUC values for F18 and F19 problems. Additionally,

Table 1: Aggregate performance statistics for the various parameter settings of the two-rate genetic algorithm applied to the LABS problem; the first row contains data on the full explored parameter space, with the consequent rows containing the data conditional to one parameter kept fixed while the others are allowed to vary.

Fixed parameter	**Parameter value**	**Averaged best fitness over 20 runs**				**Opt. values**			
		Min.	**Mean**	**Max.**	**Std. dev.**	r_{init}	F	μ	λ
None	-	2.815	3.696	4.261	0.338	20	1.05	1	2
r_{init}	2	2.86	3.698	4.255	0.326	-	1.05	3	9
	3	2.895	3.701	4.231	0.323	-	1.04	2	7
	4	2.869	3.712	4.203	0.345	-	1.1	4	9
	5	2.885	3.698	4.251	0.347	-	1.01	2	5
	10	2.863	3.697	4.25	0.344	-	1.06	3	5
	15	2.908	3.687	4.211	0.341	-	1.1	2	7
	20	2.82	3.693	4.261	0.337	-	1.05	1	2
	25	2.843	3.692	4.195	0.34	-	1.07	1	7
	30	2.815	3.687	4.205	0.341	-	1.08	4	8
F	1.01	3.011	3.725	4.251	0.251	5	-	2	5
	1.02	2.967	3.719	4.21	0.295	20	-	5	10
	1.03	2.908	3.715	4.179	0.314	10	-	2	7
	1.04	2.943	3.703	4.231	0.333	3	-	2	7
	1.05	2.885	3.699	4.261	0.347	20	-	1	2
	1.06	2.885	3.696	4.25	0.354	10	-	3	5
	1.07	2.86	3.681	4.251	0.358	5	-	5	9
	1.08	2.82	3.68	4.247	0.368	5	-	2	7
	1.09	2.874	3.67	4.173	0.367	25	-	2	2
	1.1	2.815	3.673	4.211	0.371	15	-	2	7
μ	1	3.525	3.888	4.261	0.098	20	1.05	-	2
	2	3.543	3.894	4.251	0.105	5	1.01	-	5
	3	2.815	3.711	4.255	0.351	2	1.05	-	9
	4	2.863	3.56	4.205	0.379	30	1.08	-	8
	5	2.82	3.427	4.251	0.343	5	1.07	-	9
λ	2	2.815	3.399	4.261	0.417	20	1.05	1	-
	3	2.874	3.417	4.142	0.392	20	1.03	2	-
	4	2.972	3.617	4.167	0.346	15	1.09	3	-
	5	2.922	3.644	4.251	0.351	5	1.01	2	-
	6	3.086	3.779	4.195	0.266	4	1.01	1	-
	7	3.021	3.785	4.247	0.267	5	1.08	2	-
	8	3.622	3.885	4.205	0.099	30	1.08	4	-
	9	3.585	3.872	4.255	0.105	2	1.05	3	-
	10	3.418	3.867	4.21	0.104	20	1.02	5	-

Table 2: Aggregate performance statistics for the various parameter settings of the two-rate genetic algorithm applied to the one-dimensional Ising ring problem; the first row contains data on the full explored parameter space, with the consequent rows containing the data conditional to one parameter kept fixed while the others are allowed to vary.

Fixed parameter	**Parameter value**	**Averaged best fitness over 20 runs**				**Opt. values**			
		Min.	**Mean**	**Max.**	**Std. dev.**	$r_{\textbf{init}}$	F	μ	λ
None	-	37.4	45.137	48.8	3.376	3	1.07	1	2
r_{init}	2	38.3	45.491	48.5	2.932	-	1.06	1	6
	3	38.2	45.424	48.8	3.061	-	1.07	1	2
	4	38.3	45.289	48.7	3.241	-	1.08	1	3
	5	38	45.243	48.6	3.332	-	1.05	1	3
	10	38	45.048	48.8	3.495	-	1.07	1	3
	15	37.8	44.99	48.7	3.53	-	1.08	1	3
	20	37.8	44.955	48.6	3.55	-	1.07	1	3
	25	37.4	44.89	48.5	3.569	-	1.02	1	2
	30	37.6	44.902	48.6	3.543	-	1.1	1	3
F	1.01	37.8	45.164	48.5	3.039	3	-	1	3
	1.02	37.4	45.211	48.5	3.219	25	-	1	2
	1.03	37.5	45.175	48.6	3.318	10	-	1	6
	1.04	37.8	45.135	48.4	3.355	25	-	1	2
	1.05	38	45.128	48.6	3.422	5	-	1	3
	1.06	37.6	45.102	48.6	3.497	4	-	1	3
	1.07	37.8	45.119	48.8	3.462	3	-	1	2
	1.08	37.8	45.107	48.7	3.491	4	-	1	3
	1.09	38.1	45.118	48.5	3.443	2	-	1	3
	1.1	38.2	45.11	48.7	3.482	3	-	1	2
μ	1	45.4	47.725	48.8	0.454	3	1.07	-	2
	2	45.6	47.353	48.4	0.369	3	1.1	-	3
	3	37.4	45.099	47.7	3.231	10	1.03	-	6
	4	37.4	43.46	47.7	3.485	20	1.1	-	6
	5	37.6	42.049	47.2	3.176	2	1.06	-	8
λ	2	37.4	42.623	48.8	4.355	3	1.07	1	-
	3	37.5	42.766	48.8	4.253	10	1.07	1	-
	4	38.5	44.42	48.6	3.767	4	1.07	1	-
	5	38	44.383	48.4	3.634	2	1.01	1	-
	6	38.8	45.908	48.6	2.811	10	1.03	1	-
	7	38.7	45.715	48.1	2.636	4	1.09	1	-
	8	45.2	46.87	48.2	0.565	2	1.06	1	-
	9	45.1	46.852	48.2	0.586	3	1.07	1	-
	10	44.3	46.695	47.9	0.522	4	1.07	1	-

Table 3: Labels for each of the functions as well as the area-under-the-curve (AUC) i.e. the integral of the aggregated empirical cumulative distribution for the genetic algorithm. Note that the label IDs vary between the F18 and F19 results as they were generated using timestamps, and so the ID prefix must be combined with the corresponding timestamp to find the label that is used for a given parameter configuration. Also included is a set of data produced by reference optimisation functions provided by H. Wang (personal communication, 2023). AUC values were produced using the default target values assigned by IOHanalyser.

r_{init}	F	μ	λ	**ID prefix**	**ID timestamp**		**AUC**	
					F18	F19	F18	F19
2	1.01	1	10	2_rate_(1+10)-GA2023_12_06_	181939	181941	0.693	0.884
2	1.01	1	2	2_rate_(1+2)-GA2023_12_06_	181817	181823	0.735	0.900
20	1.05	1	2	2_rate_(1+2)-GA2023_12_06_	232230	232236	0.752	0.898
3	1.07	1	2	2_rate_(1+2)-GA2023_12_07_	005500	005506	0.699	0.916
2	1.01	1	8	2_rate_(1+8)-GA2023_12_06_	181925	181927	0.715	0.878
2	1.01	1	9	2_rate_(1+9)-GA2023_12_06_	181933	181935	0.705	0.896
2	1.01	2	10	2_rate_(2+10)-GA2023_12_06_	182119	182122	0.695	0.869
2	1.01	2	8	2_rate_(2+8)-GA2023_12_06_	182104	182107	0.699	0.865
2	1.01	2	9	2_rate_(2+9)-GA2023_12_06_	182112	182114	0.716	0.877
-	-	-	-	(1+10)-2rate-EA>0	-	-	0.678	0.887
-	-	-	-	(1+1)>_0 EA	-	-	0.701	0.919
-	-	-	-	(1+1) fast GA	-	-	0.703	0.915
-	-	-	-	gHC	-	-	0.684	0.306
-	-	-	-	(1+(10,10))>_0 EA	-	-	0.669	0.851
-	-	-	-	RLS	-	-	0.692	0.925
-	-	-	-	random search	-	-	0.467	0.610
-	-	-	-	sa_auto	-	-	0.480	0.744
-	-	-	-	sars_auto	-	-	0.514	0.836
-	-	-	-	UMDA	-	-	0.591	0.804

Table 4: Labels for each of the functions as well as the area-under-the-curve (AUC) i.e. the integral of the aggregated empirical cumulative distribution for the evolution strategy. Note that included is a set of data produced by reference optimisation functions provided by H. Wang (personal communication, 2023).

Mutation Type	**Initial** σ	μ	λ	**ID prefix**	**AUC**	
					F18	F19
correlated	1	12	50	(12+50)-ES-correlated-1	0.510	0.857
correlated	0.5	15	100	(15,100)-ES-correlated-0.5	0.522	0.843
correlated	1	15	100	(15,100)-ES-correlated-1	0.507	0.862
correlated	1	2	50	(2+50)-ES-correlated-1	0.524	0.857
correlated	2	12	100	(12,100)-ES-correlated-2	0.477	0.899
correlated	0.5	15	100	(15+100)-ES-correlated-0.5	0.564	0.830
correlated	0.5	15	50	(15+50)-ES-correlated-0.5	0.554	0.846
correlated	1	10	20	(10+20)-ES-correlated-1	0.533	0.878
correlated	0.5	12	100	(12,100)-ES-correlated-0.5	0.564	0.843
-	-	-	-	(1+10)-2rate-EA>0	0.678	0.887
-	-	-	-	(1+1)>_0 EA	0.701	0.919
-	-	-	-	(1+1) fast GA	0.703	0.915
-	-	-	-	gHC	0.684	0.306
-	-	-	-	(1+(10,10))>_0 EA	0.669	0.851
-	-	-	-	RLS	0.692	0.925
-	-	-	-	random search	0.467	0.610
-	-	-	-	sa_auto	0.480	0.744
-	-	-	-	sars_auto	0.514	0.836
-	-	-	-	UMDA	0.591	0.804

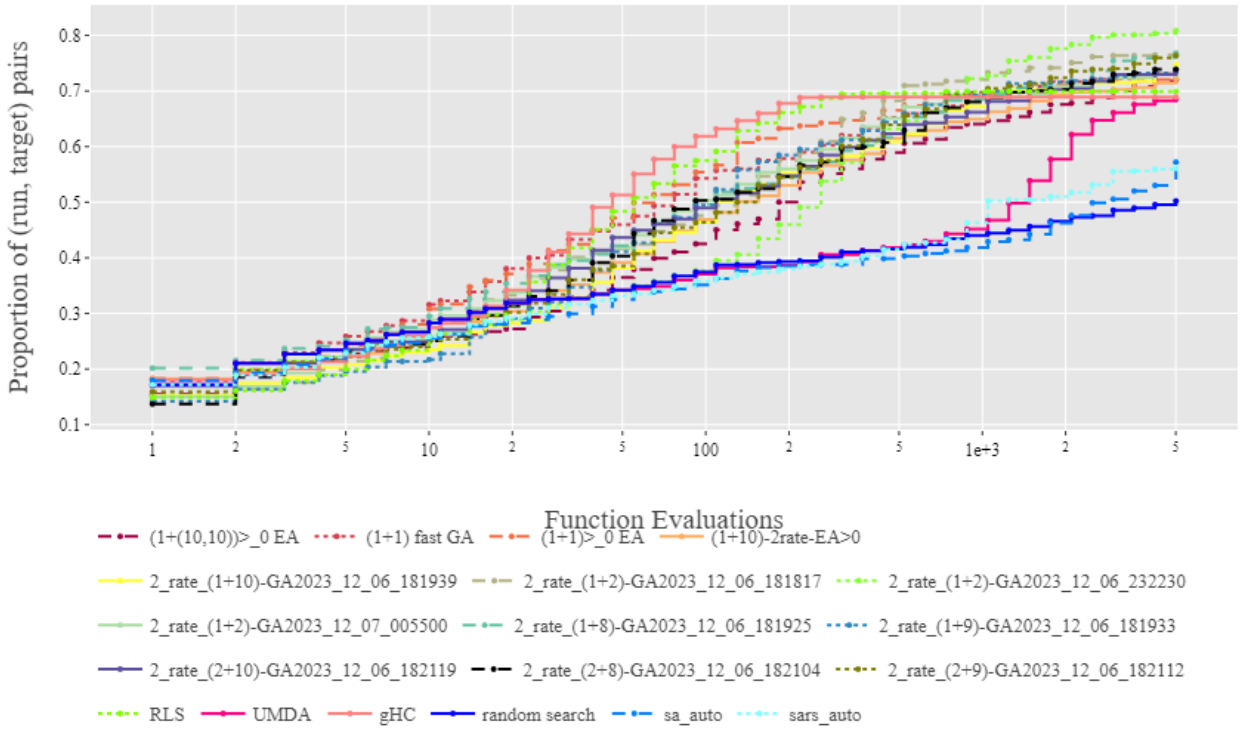

Figure 1: Aggregated empirical cumulative distribution for the various optimisation algorithms chosen for further consideration, applied to F18; label meanings are listed in Tab. 3.

a reference data is included in the Table and Figures produced by reference optimisation algorithms, provided by H. Wang (personal communication, 2023).

4 Discussion and Conclusion

We will discuss the presented results as follows: we will first give a per-algorithm discussion, on the genetic algorithm in Sec. 4.1 and on the evolution strategy in Sec. 4.2. We will follow this up with a comparison between the two in Sec. 4.3, to be followed by a conclusion of the discussed matters in Sec. 4.5

4.1 Two-rate genetic algorithm

Several things can be said about the two-rate genetic algorithm, now generalised to any number of parents. We will start with a short discussion of the results for various parameters, move on to a comparison to previous work on the single-parent two-rate genetic algorithm, followed by a comparison with respect to other optimisation methods. We will end by discussing some recommendations for future work on two-rate genetic algorithms.

4.1.1 Parameter influences

In general, the various parameter settings perform roughly comparably in terms of the maximum objective value achieved, as can be seen in Tabs. 1 and 2. What is noteworthy though is that some seem to yield far more consistent optimisation values than others, as indicated by (1) a smaller spread between minimum and maximum attained values, (2) a smaller standard deviation and (3) a larger mean value. In particular, this is notably what happens for $\in \{8, 9, 10\}$ and $\mu \in \{1, 2\}$. Consequently, even if this family of algorithms did not produce the best obtained averaged best fitness over 20 runs, they did prove to be most consistent.

No such effect is clear for r_{init} and F, where only a marginal difference is found. Roughly, this seems to indicate that the performance of the algorithm only marginally depends on r and F, while μ and λ have a much greater influence -

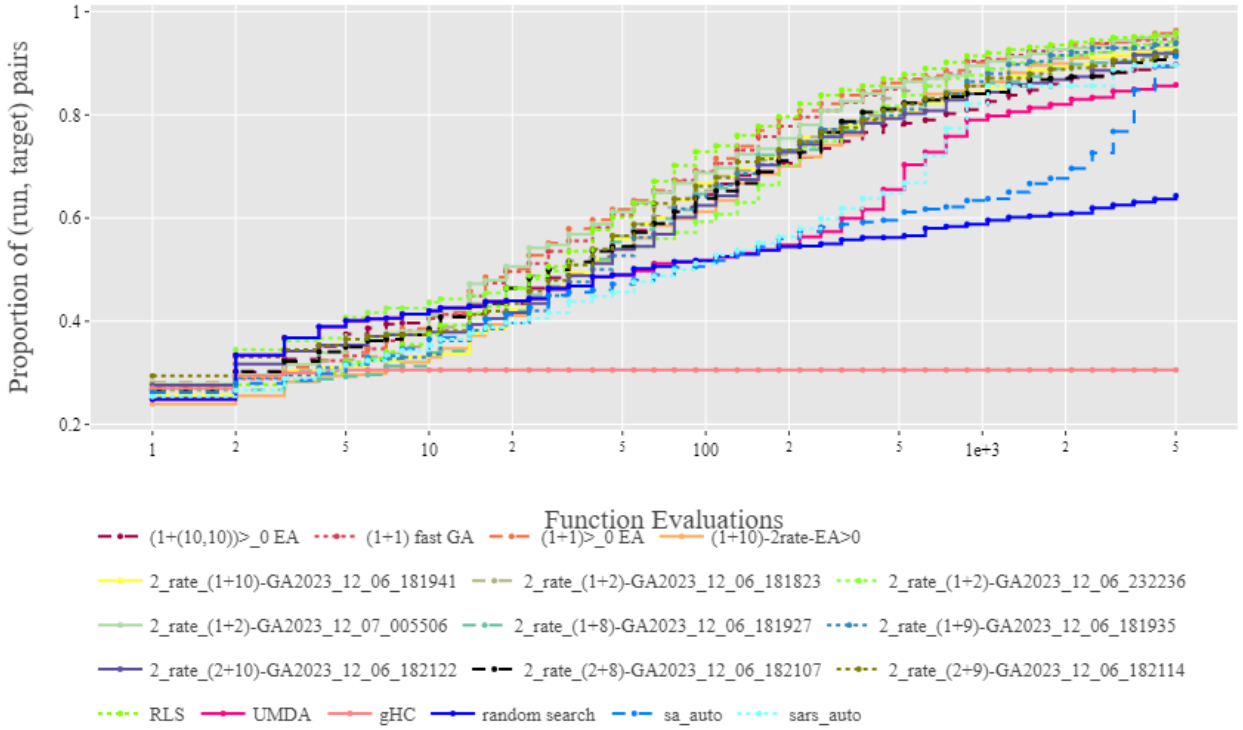

Figure 2: Aggregated empirical cumulative distribution for the various optimisation algorithms chosen for further consideration, applied to F19; label meanings are listed in Tab. 3.

justifying the choice for a two-rate algorithm over a pre-determined static or dynamic mutation rate, as this safes us having to tune a parameter.

4.1.2 Comparison to previous work

Doerr et al. (2019a) introduced the two-rate genetic algorithm as a way to get around having to select particular mutation rates which are by their nature going to be suitable only for particular problem, which undermines the strengths that genetic algorithms can play to. In this work, we have checked and validated that their idea that the performance of the two-rate genetic algorithm is not particular sensitive to the mutation rate change factor F and the initial mutation rate r_{init}. However, we do note that our results as presented in Tabs. 1 and 2 indicate that lower values for both these parameters lead to more consistent behaviour of the optimisation algorithm, while having little to no effect on the algorithm performance in terms of best-obtained value. Lastly, we can conclude that the decision by Doerr et al. (2019a) to use a single parent was justified.

4.1.3 Comparison to other algorithms

In Figs. 1 and 2 we have compared the most promising two-rate $(\mu + \lambda)$-GAs to data produced by other optimisation algorithms. In general, we observe performance that is at least comparable to other algorithms, with some algorithms even significantly outperforming them for F18. We conjecture that this difference is mainly caused by the relatively local nature of the one-dimensional Ising ring (i.e. a gene only interacts with its neighbours in its contribution to the fitness function), which means that algorithms employing crossover or other “guided mutation" techniques can do better than our genetic algorithm, whose mutation is random (as opposed to the evolution of the mutation rate, of course). This is supported by the fact that our algorithms perform better on F18, where the strong non-locality of the LABS problem means that multiple “lucky" random mutations might be necessary to overcome a local optimum.

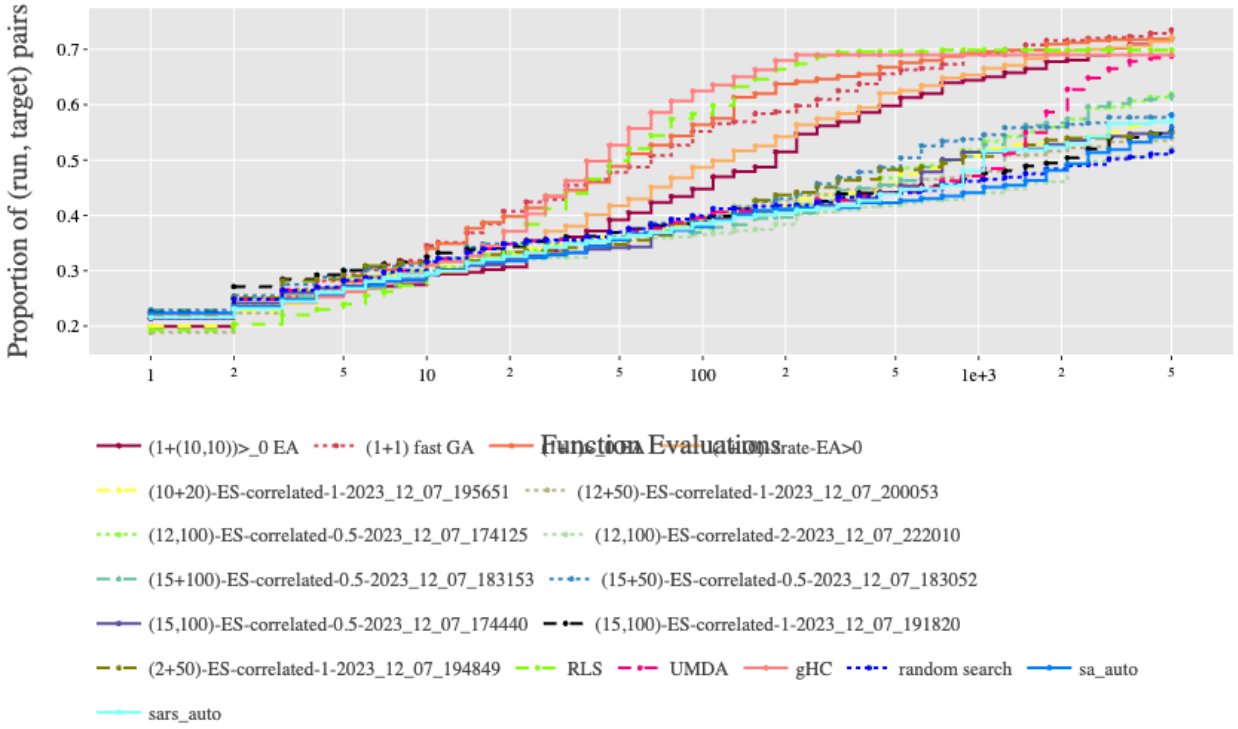

Figure 3: Aggregated empirical cumulative distribution for Evolution Strategy algorithms and the reference data provided chosen for further consideration, applied to F18.

4.1.4 Shortcomings and recommendations for future work

We have made an initial start with the generalisation of the algorithm proposed by Doerr et al. (2019a), but many options to generalise it further still remain. We shortly discuss several.

Generalising the selection operator to have a variable offset in the proportional selection probability might prove helpful in pushing the algorithm out of local optima: by setting this offset equal to the minimum function value obtained by the parent population, we effectively use a parent population that is one lower than indicated (as the lowest-performing parent is always discarded), except for when $\mu = 1$, in which case we always select the one parent. Hence, our results preferring $\mu = 1$ and $\mu = 2$ indicates that single-parent two-rate GAs outperform multiple-parent two-rate GAs. Increasing the proportional selection probability offset will punish slightly sub-optimal solutions less severely than we currently do, and so in combination with a greater parent population this might result in more efficient escape from local optima.

Doerr et al. (2019a) already experimented with three-rate GAs comprising three equally-sized subpopulations, where one exploits the current mutation rate r without any multiplicative factor. Our result that the factor F is not particularly influential for the performance of the algorithm indicate that perhaps the optimal mutation rate varies relatively slowly. Hence, it might be possible to have one “main population" comprising the majority of the offspring, mutated with the rate r, with two small “mutation rate-control" populations. This would allow a substantial number of individuals to exploit the optimal rate, while only a relatively small number are “sacrificed" to mutation rate-control.

4.2 Evolution strategy

For the general discussion of the performance of ES, it can be said based on the results present in Table 4 that the algorithms do not perform up to par with the reference optimisation algorithms provided by H. Wang (personal communication, 2023), and it may be the case because of encoding and decoding part, where a lot of information can be lost. The encoding and decoding bounds each value of real-spaced vector into bounds of 0 and 32. This causes some limitations in the performance of the algorithm. Hence, it can be stated that the self-adapting evolution strategy

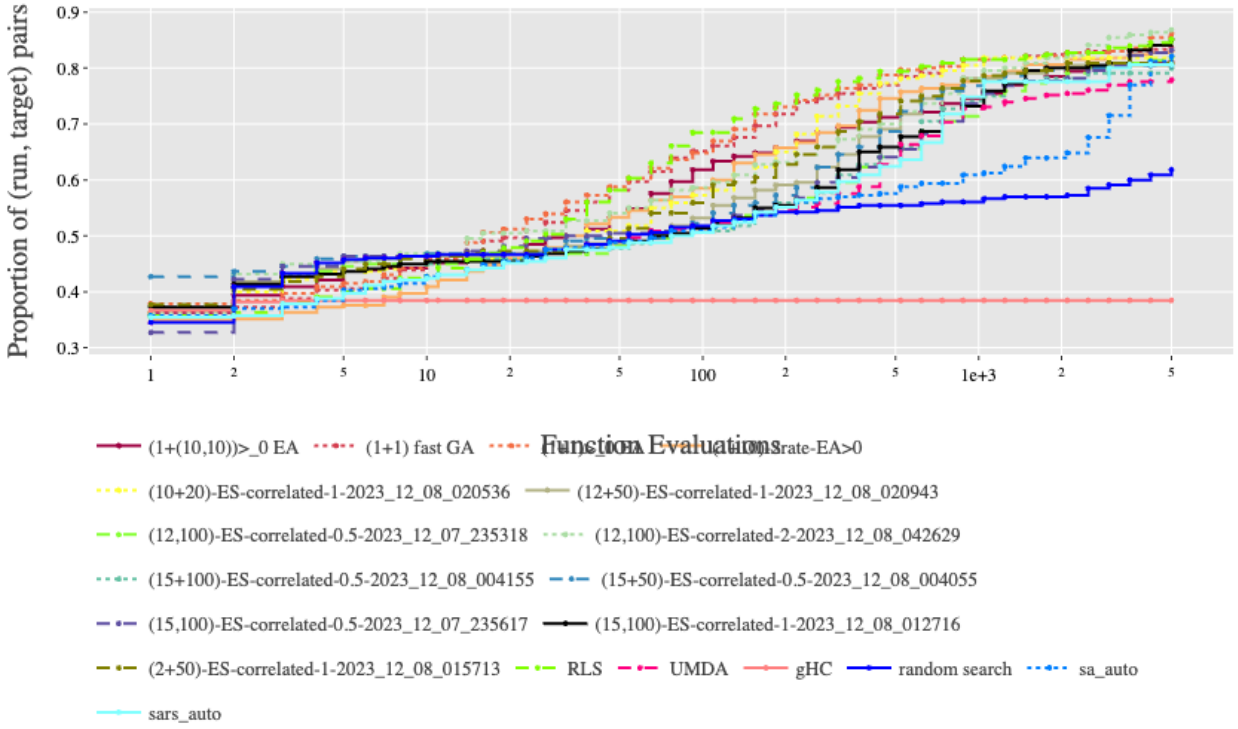

Figure 4: Aggregated empirical cumulative distribution for Evolution Strategy algorithms and the reference data provided chosen for further consideration, applied to F19.

does not work well for binary search spaces, where encoding and decoding of the data needs to be performed for each iteration. Evolution strategy is a better fit for real-valued search spaces, and all of the operations, such as mutation and recombination, are designed that way.

4.3 Comparison between the two algorithms

In general, we observe that the genetic algorithms vastly outperform the evolution strategies. We conjecture that this is a result of two factors: (1) the genetic algorithms are by their very nature better-suited toward the discrete problems we have tackled in this paper and (2) the use of a mutation rate controlled at population level for the genetic algorithm means that it can more rapidly overcome local minima than the individual control for the evolution strategy.

A point that we should stress, however, is the fact that evolution strategies can more readily have different genotypes lead to the same phenotype using our encoding algorithm. While the advantage this brings is not apparent in our current implementations, this may be useful for future work on F18, given its many local optima (and non-trivial nature of the global optimum, as opposed to F19): it allows for hidden/non-expressive genotypes, where the continuous underlying nature of evolution strategies might allow an individual to be “closer" to mutation to a new optimum if a local optimum is reached. Consequently, a hybrid approach between the two-rate genetic algorithm and the evolution strategy might prove useful in the future.

4.4 Recommendations for future work

A point of interest for future work might be to take inspiration from the work by Doerr et al. (2017), who show that small-population genetic algorithms with heavy-tailed mutation operators (so-called “fast genetic algorithms") are a promising optimisation algorithm. The good performance of some of our (1+2)-GAs with relatively high values of F (resulting in more volatile mutation behaviour) might be indicative of a similar phenomenon, and so altering the manner in which the mutation rates for the two sub-populations are chosen might yield similarly promising results for the GAs.

Given that we have only performed 20 runs for each algorithm, the good performance of these (1+2)-GAs might also just be a product of random chance, meaning this warrants further examination.

This latter point is indicative of a major recommendation we wish to provide for future work: proper analysis of the statistical properties of the performance of these algorithms requires more than 20 runs per algorithm. Putting proper numbers on, say, the standard deviation of the optimisation values produced by each algorithm may allow a more targeted selection of parameter settings: a consistently "good enough" algorithm might be preferred in some scenarios where running time is limited, whereas in a scenario where running time is no issue an algorithm that is more exploratory might be preferred, to name one example of such a trade-off. Exploring the algorithms that this analysis has found promising using more sample runs for each should allow determination of this statistical behaviour.

Finally, it should be stressed that our current implementations are fully agnostic toward the underlying problem they are trying to solve; while this is a prerequisite if we wish for our results to be indicative of the general performance of these algorithms, for the particular applications of F18 and F19 we may benefit from properties of the problems known from literature. Packebusch & Mertens (2016), for example, were able to restrict the search space of F18 from 2^N (with N the number of dimensions) to 1.72^N using a branch-and-bound algorithm, where branches of the search tree are pruned based on rigorous bounds constructed from already computed function values; if such a bound can be cheaply evaluated, it may be possible to, for example, a-priori reject mutations that would result in offspring that can be guaranteed to perform more poorly than previously obtained results, without the need to evaluate these individuals. This holds for both the genetic algorithms and the evolution strategies. Similarly, Fischer & Wegener (2004) were able to show using properties of the Ising ring model that crossover is from a probabilistic point of view expected to be an effective mechanism for optimisation of that particular problem. Future work may thus benefit from a more problem-tailored approach (though at the loss of generality).

4.5 Conclusion

To summarise, we can conclude the following from this research with respect to genetic algorithms:

1. In terms of genetic algorithms applied to F18 and F19 in particular, we recommend use of the two-rate (1+2)-GA with $F = 1.05$ and $r = 20$.
2. Two-rate (1+λ)-GAs with $\lambda \approx 10$ in general provide a reliable, consistent optimiser, without requiring any knowledge on the problem to be optimised, though the (1+2)-GA does (though less consistently) outperform it.
3. Two-rate (1+2)-GAs with volatile self-adjusting mutation rates may well provide an interesting avenue of research that can parallel the success of fast GAs.

As for evolution strategies, we have identified the following main points:

1. The ES does not perform well for discrete binary search spaces.
2. It benefits from a correlated mutation and large λ and μ values.

Finally, we recommend that future research explore the possibility of a two-rate evolutionary strategy, as there is no indication in our results that the success of the two-rate algorithm stems from its discrete representation. This may provide a useful middle ground between single step-size evolution strategies and those that use individual step sizes.

References

Bäck, T., Foussette, C., & Krause, P. 2013, Contemporary Evolution Strategies. http://www.springer.com/series/4190

Bäck, T., & Schütz, M. 1996, in International Symposium on Methodologies for Intelligent Systems (Springer Berlin Heidelberg), 158–167

De Jong, K. 1988, Machine Learning, 3, 121

Doerr, B., & Doerr, C. 2018, Algorithmica, 80, 1658, doi: 10.1007/s00453-017-0354-9

Doerr, B., Gießen, C., Witt, C., & Yang, J. 2019a, Algorithmica, 81, 593, doi: 10.1007/s00453-018-0502-x

Doerr, B., Makhmara, R., Le, H. P., & Nguyen, T. D. 2017, in Proceedings of the Genetic and Evolutionary Computation Conference (Association for Computing Machinery, Inc), 777–784, doi: 10.1145/3071178.3071301

Doerr, C., Wang, H., Ye, F., et al. 2019b, in GECCO 2019 Companion - Proceedings of the 2019 Genetic and Evolutionary Computation Conference Companion (Association for Computing Machinery, Inc), 1798–1806, doi: 10.1145/3319619.3326810

Doerr, C., Wang, H., Ye, F., van Rijn, S., & Bäck, T. 2018, arXiv e-prints:1810.05281. http://arxiv.org/abs/1810.05281

Eiben, A. E., & Smith, J. E. 2015, Introduction to Evolutionary Computing, 2nd edn. www.springer.com/series/

Emmerich, M., Shir, O. M., & Wang, H. 2018, in Handbook of Heuristics, Vol. 1-2 (Springer International Publishing), 89–119, doi: 10.1007/978-3-319-07124-4{_}13

Fischer, S., & Wegener, I. 2004, in Proceedings of the Genetic and Evolutionary Computation Conference 2004, 1113–1124

Martí, R., Pardalos, P. M., & Resende, M. G. C., eds. 2018, Handbook of Heuristics

Militzer, B., Zamparelli, M., & Beule, D. 1998, IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, 2, 34

Mow, W. H., Du, K. L., & Wu, W. H. 2015, IEEE Transactions on Aerospace and Electronic Systems, 51, 290, doi: 10.1109/TAES.2014.130518

Packebusch, T., & Mertens, S. 2016, Journal of Physics A: Mathematical and Theoretical, 49, doi: 10.1088/1751-8113/49/16/165001

Rodionova, A., Antonov, K., Buzdalova, A., & Doerr, C. 2019, in Proceedings of the Genetic and Evolutionary Computation Conference, 855–863

Schaffer, J. D., Caruana, R. A., Eshelman, L. J., & Das, R. 1989, in Proceedings of the third international conference on Genetic Algorithms, 51–60. https://www.researchgate.net/publication/220885653

Srinivas, M., & Patnaik, L. M. 1994, IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS, 24, 656

Sudholt, D. 2005, in GECCO '05: Proceedings of the 7th annual conference on Genetic and evolutionary computation