The Voting algorithm is robust to various noise models

Aishwaryaprajna; Rowe, Jonathan E.

DOI:
10.1016/j.tcs.2023.113844

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (Harvard):
Aishwaryaprajna, & Rowe, JE 2023, 'The Voting algorithm is robust to various noise models', *Theoretical Computer Science*, vol. 957, 113844. https://doi.org/10.1016/j.tcs.2023.113844

Theoretical Computer Science 957 (2023) 113844

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

The Voting algorithm is robust to various noise models ☆

Aishwaryaprajna [a,*], Jonathan E. Rowe [a,b]

[a] School of Computer Science, University of Birmingham, Birmingham, United Kingdom
[b] The Alan Turing Institute, London, United Kingdom

ARTICLE INFO

Article history:
Received 6 June 2022
Received in revised form 16 March 2023
Accepted 21 March 2023
Available online 29 March 2023
Communicated by B. Doerr

Keywords:
Voting
Recombination
Noise

ABSTRACT

A simple Voting algorithm has been shown to be effective at solving the OneMax problem in the presence of high levels of posterior noise in our previous research. In this paper, we extend this analysis to several different noise models, and show that the Voting algorithm remains robust in all of them. We consider the prior noise model and the partial evaluation of randomly selected bits. The Voting algorithm has superior runtime bounds on these problems compared to other published algorithms. We also introduce a new variant of partial evaluation, and further consider the simple model when a comparison-based oracle produces incorrect results with a fixed probability.

1. Introduction

In *black box optimisation*, we have a search space (typically $\{0,1\}^n$) and an *oracle*. The oracle evaluates elements of the search space, using a hidden objective function. The goal is to find an optimal element of the search space, as evaluated by the oracle, with as few queries as possible [1]. A common benchmark problem class used in theoretical studies is OneMax, in which the oracle returns the bitwise similarity (or, equivalently, Hamming distance) to a hidden target string. This can be solved trivially in $O(n)$ queries by starting with some initial string, and then evaluating the effects of flipping each bit, one at a time. Less trivially, a method originating with Erdős, but derived independently in the context of black box optimisation in [2], shows that $\Theta(n/\ln n)$ queries will suffice — one simply evaluates this number of random strings, and this gives enough information to deduce the target.

The situation is made considerably more challenging when the oracle's response is affected by noise. In the case of *posterior* noise, the value returned by the oracle is the true value plus a random variate from some probability distribution. One obvious approach here is to resample each string several times, and use the average as an estimate of the true value. If the noise comes from a centred Gaussian distribution, with variance σ^2, then it can be shown that $\sigma^2 \ln T$ samples are required for each string to guarantee accurate estimates across T different strings [3]. Thus, for OneMax with posterior Gaussian noise, we get an overall runtime of $O(\sigma^2 n)$, using the Erdős method. It has been proposed [4] that the median provides a better estimate than the mean when re-sampling noisy fitness functions. However, for additive Gaussian posterior noise, the advantage only holds when the variance is very small, i.e. $0 < \sigma < 0.9538$.

The compact genetic algorithm (cGA) is shown to handle posterior noise with (large) polynomial runtime bounds [5]. A better asymptotic runtime for OneMax with posterior Gaussian noise is proved for the Paired Crossover Evolutionary Algorithm (PCEA) which just uses crossover, and no mutation [6].

☆ This article belongs to Section C: Theory of natural computing, Edited by Lila Kari.

* Corresponding author.
E-mail addresses: aishwaryprajna@gmail.com (Aishwaryaprajna), j.e.rowe@cs.bham.ac.uk (J.E. Rowe).

https://doi.org/10.1016/j.tcs.2023.113844

In the case of *prior noise*, the oracle evaluates the value of a string which has been mutated with some probability. The performance of the hill-climber $(1+1)$EA on OneMax with prior noise was examined by an early theoretical result [7], showing that super-polynomial time is needed when the probability of mutating even a single bit is $\omega(\log(n)/n)$. In [8] it is shown that if the fitness is estimated by sampling each string 3 times, then the runtime reduces to $O(n \log n)$. Using the median, rather than the mean, can also reduce the runtime to polynomial, but in this case much larger sample sizes of $O(n^3)$ are required [9].

The well-studied LeadingOnes problem, where the oracle provides the maximum length of the prefix of a string having all values set to one, requires step by step improvement in a particular sequence to reach the optimum. The threshold between polynomial and super-polynomial expected runtimes for the (1+1)EA on LeadingOnes with one-bit prior noise is shown to be located at $p = \Theta(\log n/n^2)$ in [10].

It has been recognised for a long time that the population size can affect the ability of an EA to handle noise [11,12]. The results in [7] was generalised in [13] to consider populations using the $(\mu+\lambda)$EA, and showed that populations are beneficial in both prior and posterior noise. However, they show that $(1+1)$EA can only tolerate posterior Gaussian noise when the variance is very small (less than $1/(4\log n)$). The results in [14] also show that $(1+1)$EA has $O(n \log n)$ only for very small variances of posterior noise. A more recent theoretical study by [15] shows that a low mutation rate enables a particular mutation-population algorithm to handle arbitrary posterior noise for the OneMax problem in polynomial time, although the bounds given are large. A recent study [16] with a non-elitist population-based EA which mutates the tournament winner and adds it to the next population on the OneMax problem in presence of one-bit and multiple-bit prior noise models showed the runtime bounds to be $O(n^2)$ and $o(n^3 \log n)$ for one-bit and multiple-bit prior noise respectively.

In [17], we presented the Voting algorithm, which improves on the runtime for OneMax with posterior noise. For OneMax with posterior noise taken from any unimodal distribution with finite mean and variance, this algorithm has a runtime of $O(n \ln n)$ when $\sigma^2 = O(n)$ and $O(\sigma^2 \ln n)$ when the variance is greater than this. In this paper, we extend this work by analysing the performance of the Voting algorithm on OneMax with several different noise models as follows:

- *Posterior* noise — the results of [17] are restated for completeness.
- *Prior* noise — in which the string to be evaluated is mutated beforehand.
- *Random bit evaluation* — in which bits have a fixed probability of being considered by the oracle.
- *Random subset evaluation* — in which a random subset of bits, of a fixed size, are evaluated.
- *Deceptive oracle* — the oracle lies when comparing two strings with a fixed probability.

It was observed in [17] that, since the Voting algorithm samples only uniformly random strings, its behaviour on OneMax is identical to its behaviour on Jump, in which the gap size is αn for any constant $0 \leq \alpha < 1/2$. The same observation applies in this paper with regards the application of posterior noise, and the deceptive oracle. In the case of prior noise, the gap range must be restricted to $0 \leq \alpha < 1/3$, to avoid the possibility that strings are mutated into the gap.

Voting has previously been considered as a mechanism to enhance evolutionary algorithms, for example in [18–20]. In our work, the selection and voting process may be considered as a simplification or abstraction of various evolutionary approaches. For example, the algorithm is equivalent to doing a single iteration of UMDA [21], using binary tournament selection, and then considering whether the bit frequencies have changed (with high probability) towards the correct values. Alternatively, it can be seen as a version of the compact genetic algorithm [22] in which the bitwise probabilities are not updated between iterations, and the final result depends on information from the entire runtime history. Finally, there are close similarities with the use of crossover, since when crossing two strings, the bits which are the same are untouched, but the bits which are different are randomised. Performing a tournament selection between two offspring of the same parents therefore induces a drift towards the correct bit values on OneMax, which is large enough to overcome significant levels of noise (see particularly [6]).

It is worth observing that the Voting algorithm requires no *adaptation* as it works from a sample of uniformly generated random strings. As such, it is potentially useful in the context of *one-shot* optimisation [23], and it is remarkable that noisy OneMax and Jump can be solved in this context. Similarly, the algorithm can be easily parallelised, since all the evaluations are done independently. The parallel black box complexity of problem classes has recently been examined in [24], although restricted to unary unbiased algorithms [25]. The Voting algorithm we present is unbiased, but not unary (as the voting step requires all tournament winners as input). Consequently, the restrictions of [24] do not apply, and our results show that the unrestricted parallel black box complexity of noisy OneMax and Jump is, in fact, a single generation, requiring a population size of $O(n \log n)$ to ensure finding the solution with high probability.

2. The Voting algorithm

In this section, we introduce the Voting algorithm from [17] and restate the basic results concerning the runtime of this algorithm on OneMax when there is no noise. We prove a generalisation of this result in the situation where strings are generated with a bias towards the correct value. This situation may arise, for example, when strings are the result of running a few steps of a local search algorithm, rather than simply being generated randomly. However, it is important to note that in our analysis, it is essential that the values of the bits are independent, and this would not generally be

the case following the application of local search. These results must therefore be seen as some preliminary steps towards understanding this situation.

In the Voting Algorithm, we generate two uniform random strings, choose the best of the two, and add it to the population. When there are enough strings in the population, we take a bit-wise vote, breaking ties randomly (see Algorithm 1). Please note that, the indicator function is denoted by $[\cdot]$ in the algorithm and, without loss of generality, that the target string is 1^n.

Algorithm 1: The Voting algorithm.

```
Let p = (0, ..., 0);
repeat μ times
    Let x ∈ {0,1}^n be a uniform random string;
    Let y ∈ {0,1}^n be a uniform random string;
    if f(x) > f(y) then
        p = p + x;
    else
        p = p + y;
    end
end
for 1 ≤ i ≤ n do
    if p_i = μ/2 then
        z_i = 0 or 1 chosen uniformly at random;
    else
        z_i = [p_i > μ/2];
    end
end
Return z;
```

The runtime analysis of the Voting algorithm on OneMax without noise hinges on the following lemma (see [17] for the proof).

Lemma 1. *Let $x, y \in \{0,1\}^n$ be uniform random strings and let the tournament winner, z, be decided according to the* OneMax *function. For any $k \in \{1, \dots n\}$,*

$$\Pr(z_k = 1) \geq \frac{1}{2} + \frac{1}{8\sqrt{n}}.$$

The runtime then follows:

Theorem 1. *If $\mu \geq 32(c+1)n \ln n$, then the Voting algorithm correctly solves* OneMax *with probability greater than $1 - 1/n^c$.*

Proof. For any one bit position, k, the probability that the vote is incorrect is

$$\Pr(p_k \leq \mu/2) \leq \exp(-2\mu/64n) \leq \frac{1}{n^{c+1}}$$

by Hoeffding's inequality. So by the union bound, the probability that at least one bit gets the incorrect vote is at most $1/n^c$. □

One can see, from the use of Hoeffding's inequality and the union bound, that we have, in general, the following which we will make use of several times:

Lemma 2. *Let $x, y \in \{0,1\}^n$ be any two strings and let the tournament winner, z, be decided according to the* OneMax *function. Suppose that, for all $k \in \{1, \dots n\}$,*

$$\Pr(z_k = 1) \geq \frac{1}{2} + \delta$$

for some $\delta > 0$ (which may be a function of various parameters, including n). Then if $\mu \geq ((c+1)\ln n)/(2\delta^2)$ the Voting algorithm correctly solves OneMax *with probability greater than $1 - 1/n^c$.*

In each of the following sections, we make use of bounds on the probability

$$\Pr\left(\sum_{i\neq k}(y_i - x_i) = 0\right)$$

under different conditions. This also appears in, for example, [26,5,27]. In our analysis of this quantity, we make use of the following result (see Lemma 1 of [28], and also [29]):

Lemma 3. *For $n \in \mathbb{N}$ and $0 \leq x \leq 1$ we have:*

$$\sum_{k=0}^{n}\binom{n}{k}^2 x^{2k}(1-x)^{2(n-k)} \geq \frac{1}{4^n}\binom{2n}{n}$$

The analysis and experiments with EAs usually involve starting with search strings chosen uniformly at random, where the probability of having one in each bit is set to $\frac{1}{2}$. If non-uniform random initialisation of strings is considered, such that each bit of the random strings would have a one with probability r and a zero otherwise and each of the bits are independent, then it enables the analysis of different scenarios where the initial strings are better than random, i.e., $r > \frac{1}{2}$.

The consideration of the probability r instead of choosing the strings uniformly at random is an initial step towards understanding the impact of biasing the initial population, theoretically, which is needed in certain application domains. When the Voting mechanism is hybridised with an initial phase of local search, such as the $(1+1)$EA that iteratively updates the current solution string with mutation, the strings in the population tend to be closer to the optimum than randomly chosen strings. This should mean that a smaller population would be needed to reach the optimum. However, initialising strings with local search would bring in dependencies within the bits.

Lemma 4. *Let $x, y \in \{0,1\}^n$ be two strings chosen at random, such that the probability of having a one in each bit is at least $r \geq 1/2$, and the bits are chosen independently. The winner of the binary tournament selection, z, decided on the basis of the* OneMax *function, will have a one in position k with probability*

$$\Pr(z_k = 1) \geq r + r(1-r)\frac{1}{2\sqrt{n}}$$

Proof. The probability that the winner of the tournament has a one in position k is given by

$$\Pr(z_k = 1) = \Pr(x_k = 1 \,|\, x \text{ wins})\Pr(x \text{ wins}) + \Pr(y_k = 1 \,|\, y \text{ wins})\Pr(y \text{ wins})$$

Since $\Pr(x \text{ wins}) = \Pr(y \text{ wins})$ holds by symmetry, we can obtain,

$$\Pr(z_k = 1) = \Pr(x_k = 1 \,|\, x \text{ wins})$$

By Bayes' Theorem

$$\begin{aligned}\Pr(x_k = 1 \,|\, x \text{ wins}) &= \frac{\Pr(x_k = 1)}{\Pr(x \text{ wins})}\Pr(x \text{ wins} \,|\, x_k = 1)\\ &\geq 2r \times \Pr(x \text{ wins} \,|\, x_k = 1)\end{aligned}$$

Using the law of total probability, we can say that,

$$\begin{aligned}\Pr(x \text{ wins} \,|\, x_k = 1) &= \Pr(x \text{ wins} \,|\, x_k = 1, y_k = 1)\Pr(y_k = 1)\\ &\quad + \Pr(x \text{ wins} \,|\, x_k = 1, y_k = 0)\Pr(y_k = 0)\end{aligned}$$

(since when $x_k = y_k = 1$ then x and y are equally likely to win the tournament)

$$\begin{aligned}&\geq \frac{1}{2}r + \Pr\left(\sum_{i\neq k} x_i + 1 > \sum_{i\neq k} y_i\right)(1-r)\\ &= \frac{1}{2}r + \Pr\left(\sum_{i\neq k}(y_i - x_i) < 1\right)(1-r)\\ &= \frac{1}{2}r + (1-r)\left[\Pr\left(\sum_{i\neq k}(y_i - x_i) < 0\right) + \Pr\left(\sum_{i\neq k}(y_i - x_i) = 0\right)\right]\end{aligned}$$

By symmetry, we have

$$\Pr\left(\sum_{i\neq k}(y_i - x_i) < 0\right) = \frac{1}{2}\left(1 - \Pr\left(\sum_{i\neq k}(y_i - x_i) = 0\right)\right)$$

Now, we can obtain,

$$\begin{aligned}
&\Pr\left(\sum_{i\neq k}(y_i - x_i) < 1\right)\\
&= \frac{1}{2} + \frac{1}{2}\Pr\left(\sum_{i\neq k}(y_i - x_i) = 0\right)\\
&= \frac{1}{2} + \frac{1}{2}\left(\sum_{j=0}^{n-1}\Pr\left(\sum_{i\neq k} y_i = j\right)\Pr\left(\sum_{i\neq k} x_i = j\right)\right)\\
&= \frac{1}{2} + \frac{1}{2}\sum_{j=0}^{n-1}\binom{n-1}{j}^2 r^{2j}(1-r)^{2n-2-2j}\\
&\geq \frac{1}{2} + \frac{1}{2^{2n-1}}\binom{2n-2}{n-1}\\
&\geq \frac{1}{2} + \frac{1}{4\sqrt{n}}
\end{aligned}$$

where we make use of Lemma 3, and then bounds on central binomial coefficients [30]. Then the required probability of having a one in the bit position k of the tournament winner z is derived as follows,

$$\begin{aligned}
\Pr(z_k = 1) &\geq r^2 + 2r(1-r)\left(\frac{1}{2} + \frac{1}{4\sqrt{n}}\right)\\
&= r + r(1-r)\frac{1}{2\sqrt{n}} \quad \square
\end{aligned}$$

We consider two cases: $r = 1/2 + \Theta(1)$ and $r = 1/2 + o(1)$.

Theorem 2. *Suppose $r = 1/2 + \gamma$, where γ is a constant. If $\mu \geq \frac{(c+1)}{2\gamma^2}\ln n$, then the Voting algorithm correctly solves* OneMax *with probability greater than $1 - 1/n^c$.*

Proof. When $r = \frac{1}{2} + \gamma$,

$$\Pr(z_k = 1) \geq r + r(1-r)\frac{1}{2\sqrt{n}} \geq r = 1/2 + \gamma$$

The result then follows from Lemma 2. $\square$

Theorem 3. *Suppose $r = \frac{1}{2} + \epsilon_n$, where $\epsilon_n \to 0^+$ as $n \to \infty$. If $\mu \geq 32(c+1)n\ln n$, then the Voting algorithm correctly solves* OneMax *with a probability greater than $1 - 1/n^c$.*

Proof. The probability that the tournament winner has a one in bit position k,

$$\begin{aligned}
\Pr(z_k = 1) &\geq \left(\frac{1}{2} + \epsilon_n\right) + \left(\frac{1}{2} + \epsilon_n\right)\left(\frac{1}{2} - \epsilon_n\right)\frac{1}{2\sqrt{n}}\\
&= \frac{1}{2} + \epsilon_n + \frac{1}{8\sqrt{n}} - \frac{{\epsilon_n}^2}{2\sqrt{n}}\\
&\geq \frac{1}{2} + \frac{1}{8\sqrt{n}}, \text{ for sufficiently large } n. \quad \square
\end{aligned}$$

This interesting idea of considering a non-uniform random string initialisation could be considered for the following analyses when noise is included. However, we have avoided this, for simplicity of presentation.

Table 1
Noisy OneMax with additive Gaussian noise. The parameters ϱ and K represent the mutation rate of (1+1)EA and the population size of cGA respectively.

Algorithm	Parameters	Runtime bounds
(1+1)EA	$\varrho = 1/n$, $\sigma^2 \leq 1/4\ln(n)$	$O(n \ln n)$ and super polynomial for larger σ^2 [13]
	$\varrho = 1/n$, $\sigma^2 \geq 1$	$e^{\Omega(n)}$ [35]
(1+1)EA with Sampling	$\varrho = 1/n$	$O(\sigma^2 n(\ln n)^2)$ [3]
Mutation-Population Algorithm		$O\left(\sigma^7 n \ln n \ln(\ln n)\right)$ [36]
Paired Crossover EA (PCEA)	$\sigma^2 = n$	$O(n(\ln n)^2)$ [6]
Compact GA (cGA)	$K = \omega(\sigma^2\sqrt{n}\ln n)$	$O(K\sigma^2\sqrt{n}\ln Kn)$ [5]
Voting Algorithm	$\sigma^2 \leq 3n/8$	$O(n \ln n)$
	$\sigma^2 > 3n/8$	$O(\sigma^2 \ln n)$

3. Voting algorithm on noisy OneMax problems

3.1. Posterior noise

For the OneMax problem with posterior noise, the fitness, at each evaluation, receives an addition of a random value drawn from some probability distribution η with variance σ^2.

$$f^{noisy}(x) = f(x) + \eta(\sigma^2)$$

In [17], we proved that the Voting algorithm has superior runtime than other existing algorithms (refer to Table 1) on OneMax with posterior noise. The result holds for arbitrary posterior noise distributions with finite mean and variance and, in the case where $\sigma^2 = \Omega(n)$ with the restriction that the noise is unimodal. We restate the main result here for completeness.

For empirical results regarding the performances of the considered algorithms in Table 1, please refer to [31,32,17,33,34]. The observations from the experiments in the cited work illustrate better performance of recombination operations in comparison to mutation operations in solving noisy combinatorial optimisation problems. The work in [17,31] illustrates the speed up due to voting operator in traditional EAs.

Theorem 4. *The Voting algorithm correctly solves noisy* OneMax *with high probability, when the noise distribution has finite mean and variance $\sigma^2 \leq 3n/8$, in $O(n \ln n)$ function evaluations. If, in addition, the noise distribution is unimodal, then in the case $\sigma^2 \geq 3n/8$, the algorithm requires at most $O(\sigma^2 \ln n)$ function evaluations.*

3.2. Prior noise

The prior noise model flips a single bit or multiple bits in the search string before the fitness evaluation is performed. Here, we have considered the generalised multiple bit-flipping noise, that is defined as follows,

$$f^{noisy}(x) = \begin{cases} f(x) & \text{with probability } (1-p) \\ f(x') & \text{with probability } p \end{cases}$$

where, x' is generated by independently flipping each bit of x with probability q. Thus, a bit gets flipped with probability pq. In this work, we present a better bound than the existing results (refer to Table 2) for this problem.

Theorem 5. *The Voting algorithm correctly solves noisy* OneMax *in the presence of the generalised multiple bit-flipping prior noise, with high probability, when $q \leq 1/3$ and $p < 1$, in $O(n \ln n)$ function evaluations.*

Proof. When a tournament is held between two uniform random strings x and y, to select a winner, z, there are four different cases, which we will consider separately:

Case A Neither x nor y are mutated, prior to evaluation.
Case B x is mutated, but y is not mutated, prior to evaluation.
Case C x is not mutated, but y is mutated, prior to evaluation.
Case D Both x and y are mutated, prior to evaluation.

It is important to notice that under the conditions of each case, the bit values of each string before and after mutation (should that happen) are independent, and are equally likely to be 0 or 1.

Table 2
Noisy OneMax with multiple bit-flipping prior noise. The parameters ϱ, λ and c represent mutation rate, population size and a positive constant respectively.

Algorithm	Parameters	Runtime
(1+1)-EA	$\varrho = 1/n$, $q = O(1/n^2)$, $p = 1$	$\Theta(n \ln n)$ [13]
	$\varrho = 1/n$, $q = O(\ln n/n^2)$, $p = 1$	polynomial [13]
	$\varrho = 1/n$, $q = \omega(\ln n/n^2)$, $p = 1$	$2^{\omega(\ln n)}$ [13]
(1+1)EA with Sampling	$\varrho = 1/n$, Sampling size $= O(n^{3+2c})$, $q \leq \frac{1}{2} - \frac{1}{n^c}$, $c = O(1)$, $p = 1$	polynomial [37]
Non-elitist EA with tournament selection	$\varrho = O(1/n^{3/2})$, $\lambda = \Omega(n \ln n)$, $q \in (0, 1/2)$, $p = 1$	$O(n^3(\ln n)^2)$ [16]
Voting Algorithm	$q \leq 1/3$, $p < 1$	$O(n \ln n)$ [Theorem 5]

We focus on a particular bit position k. In each case, the analysis is the same as for the noiseless situation, up to the point:

$$\Pr(z_k = 1) = \frac{1}{4} + \frac{1}{2}\Pr(x \text{ wins} \,|\, x_k = 1, y_k = 0)$$

When there is no noise, we have [17]

$$\Pr(x \text{ wins} \,|\, x_k = 1, y_k = 0) \geq \frac{1}{2} + \frac{1}{4\sqrt{n}}$$

Case A. Here neither strings are mutated, and so the analysis is the same as if there were no prior noise:

$$\Pr(z_k = 1 \,|\, \text{Case A}) \geq \frac{1}{2} + \frac{1}{8\sqrt{n}}$$

Case B. Assuming $x_k = 1$ and $y_k = 0$, we consider two sub-cases: when the bit x_k is flipped, and when it is not. When x_k is flipped from 1 to 0, then the strings are equally likely to win. Wen x_k is not flipped, the chance of x winning is the same as in the noiseless case. Hence:

$$\Pr(z_k = 1 \,|\, \text{Case B}) \geq \frac{1}{4} + \frac{1}{2}\left(\frac{q}{2} + (1-q)\left(\frac{1}{2} + \frac{1}{4\sqrt{n}}\right)\right)$$

which gives:

$$\Pr(z_k = 1 \,|\, \text{Case B}) \geq \frac{1}{2} + \frac{1-q}{8\sqrt{n}}$$

Case C. Assuming $x_k = 1$ and $y_k = 0$, we consider two sub-cases: when the bit y_k is flipped, and when it is not. When it is flipped from 0 to 1, the strings are equally likely to win. Otherwise, if y_k is not flipped and remains at zero, we have the same as the noiseless case:

$$\Pr(z_k = 1 \,|\, \text{Case C}) \geq \frac{1}{4} + \frac{1}{2}\left(\frac{q}{2} + (1-q)\left(\frac{1}{2} + \frac{1}{4\sqrt{n}}\right)\right)$$

which gives:

$$\Pr(z_k = 1 \,|\, \text{Case C}) \geq \frac{1}{2} + \frac{1-q}{8\sqrt{n}}$$

Case D. Assuming $x_k = 1$ and $y_k = 0$, there are four sub-cases to consider. If neither bits are flipped, then we have, as before

$$\Pr(z_k = 1 \,|\, \text{Case D}, x'_k = 1, y'_k = 0) \geq \frac{1}{2} + \frac{1}{4\sqrt{n}}$$

When x_k is flipped from 1 to 0, but y_k remains at 0, then the chances of x winning are just 1/2. Similarly for the case when x_k does not flip, but y_k does.

The fourth case to consider is when x_k flips to 0, and y_k flips to 1. Now

$$\Pr(x \text{ wins} \mid \text{Case D}, x'_k = 0, y'_k = 1) = \Pr\left(\sum_{i\neq k} x'_i > \sum_{i\neq k} y'_i + 1\right)$$

By symmetry, we have

$$\Pr\left(\sum_{i\neq k} x'_i > \sum_{i\neq k} y'_i + 1\right) = \frac{1}{2} - \frac{1}{2}\Pr\left(\sum_{i\neq k} y'_i - x'_i = 0\right) - \Pr\left(\sum_{i\neq k} y'_i - x'_i = -1\right)$$

from which we get the bound

$$\Pr\left(\sum_{i\neq k} x'_i > \sum_{i\neq k} y'_i + 1\right) \geq \frac{1}{2} - \frac{3}{2}\Pr\left(\sum_{i\neq k} y'_i - x'_i = 0\right)$$

We then use the following result from [17]

$$\Pr\left(\sum_{i\neq k} y'_i - x'_i = 0\right) = \frac{4}{2^{2n}}\binom{2n-2}{n-1}.$$

Using well-known bounds on central binomial coefficients [30], we approximate

$$\binom{2n-2}{n-1} \leq \frac{2^{2n}}{6\sqrt{n}}$$

to get

$$\Pr(z_k = 1 \mid \text{Case D}, x'_k = 0, y'_k = 1) \geq \frac{1}{2} - \frac{1}{\sqrt{n}}$$

and hence

$$\begin{aligned}\Pr(z_k = 1 \mid \text{Case D}, x_k = 1, y_k = 0) &\geq (1-q)^2\left(\frac{1}{2} + \frac{1}{4\sqrt{n}}\right) + q(1-q) \\ &\quad + q^2\left(\frac{1}{2} - \frac{1}{\sqrt{n}}\right) \\ &= \frac{1}{2} + \frac{1-2q-3q^2}{4\sqrt{n}}.\end{aligned}$$

It follows that:

$$\Pr(z_k = 1 \mid \text{Case D}) \geq \frac{1}{2} + \frac{1-2q-3q^2}{8\sqrt{n}}.$$

If $q \leq 1/3$, then we have

$$\Pr(z_k = 1 \mid \text{Case D}) \geq \frac{1}{2}.$$

Putting cases A, B, C and D together, and assuming $q \leq 1/3$, we conclude

$$\begin{aligned}\Pr(z_k = 1) &\geq (1-p)^2\left(\frac{1}{2} + \frac{1}{8\sqrt{n}}\right) + 2p(1-p)\left(\frac{1}{2} + \frac{1-q}{8\sqrt{n}}\right) + \frac{p^2}{2} \\ &= \frac{1}{2} + \frac{1}{8\sqrt{n}}(1 - p^2 - 2pq + 2p^2q) \\ &= \frac{1}{2} + \frac{1}{8\sqrt{n}}(1-p)(1+p-2pq) \\ &\geq \frac{1}{2} + \frac{1-p}{8\sqrt{n}}\end{aligned}$$

and the result follows, by applying Lemma 2. □

3.3. Partial evaluation via random bits

In noisy data mining and learning problems, incomplete or unavailable data attributes are often encountered. Here, we have considered a similar noise model where, the fitness evaluation is performed only on randomly chosen bits (attributes) of the search point which leads to a partial evaluation of the fitness function. This noise model has been studied for a non-elitist binary selection algorithm and $(1+1)$EA [36].

When solving the OneMax(x) function, where the noisy evaluation takes into consideration bits with a probability d, the noisy fitness function can be written as follows,

$$f^{noisy}(x) = \begin{cases} f(x) & \text{with probability } d^{f(x)}(1-d)^0 \\ f(x)-1 & \text{with probability } d^{(f(x)-1)}(1-d)^1 \\ f(x)-2 & \text{with probability } d^{(f(x)-2)}(1-d)^2 \\ \dots & \\ 0 & \text{with probability } d^0(1-d)^{f(x)} \end{cases}$$

Here, we show that the Voting algorithm solves the OneMax problem with this noise model in $O((n\ln n)/d^2)$ function evaluations. The non-elitist algorithm presented in [36] requires a larger bound:

$$\begin{cases} O\left(\frac{n\ln n}{d^7}\right) & \text{if } d \le 1/n \\ O\left(\frac{n^{9/2}\ln n}{d^{7/2}}\right) & \text{if } d > 1/n \end{cases}$$

Theorem 6. *Let $x, y \in \{0,1\}^n$ be two strings chosen uniformly at random. If z is the winner of the binary tournament selection using the* OneMax *function, with partial evaluation of the fitness function with probability d, then the probability that z will have a one in position k is given by*

$$\Pr(z_k = 1) \ge \frac{1}{2} + \frac{d}{8\sqrt{n}}$$

If $\mu \ge \frac{32}{d^2}(c+1)n\ln n$, then the Voting algorithm correctly solves OneMax *with probability greater than $1 - 1/n^c$.*

Proof. Following as before and using the law of total probability,

$$\begin{aligned} \Pr(z_k = 1) &= \Pr(x_k = 1 \mid x \text{ wins}) \\ &= \Pr(x \text{ wins} \mid x_k = 1) \\ &= \Pr(x \text{ wins} \mid x_k = 1, y_k = 1)\Pr(y_k = 1) \\ &\quad + \Pr(x \text{ wins} \mid x_k = 1, y_k = 0)\Pr(y_k = 0) \\ &= \frac{1}{4} + \frac{1}{2}\Pr(x \text{ wins} \mid x_k = 1, y_k = 0) \end{aligned}$$

Here, for each i, let a_i and b_i be random numbers such that they are equal to one with probability d, when bit position i is being considered in the fitness evaluation, and zero otherwise.

Then we can say,

$$\Pr(x \text{ wins} \mid x_k = 1, y_k = 0) \ge \Pr\left(\sum_{i \ne k} a_i x_i + a_k > \sum_{i \ne k} b_i y_i\right)$$

Now, there may be two cases arising from the value of a_k,

$$\begin{aligned} &\Pr(x \text{ wins} \mid x_k = 1, y_k = 0) \\ &= \Pr(x \text{ wins} \mid x_k = 1, y_k = 0, a_k = 0)\Pr(a_k = 0) \\ &+ \Pr(x \text{ wins} \mid x_k = 1, y_k = 0, a_k = 1)\Pr(a_k = 1) \\ &\ge \frac{1}{2}(1-d) + \Pr\left(\sum_{i \ne k} a_i x_i + 1 > \sum_{i \ne k} b_i y_i\right) d \end{aligned}$$

Now,

$$
\begin{aligned}
&\Pr\left(\sum_{i\neq k} a_i x_i + 1 > \sum_{i\neq k} b_i y_i\right) \\
&= \Pr\left(\sum_{i\neq k} (b_i y_i - a_i x_i) < 1\right) \\
&= \Pr\left(\sum_{i\neq k} (b_i y_i - a_i x_i) = 0\right) + \Pr\left(\sum_{i\neq k} (b_i y_i - a_i x_i) < 0\right) \\
&= \Pr\left(\sum_{i\neq k} (b_i y_i - a_i x_i) = 0\right) + \frac{1}{2} - \frac{1}{2}\Pr\left(\sum_{i\neq k} (b_i y_i - a_i x_i) = 0\right) \\
&= \frac{1}{2} + \frac{1}{2}\Pr\left(\sum_{i\neq k} (b_i y_i - a_i x_i) = 0\right) \\
&= \left(\frac{1}{2} + \frac{1}{2}\sum_{j=0}^{n-1}\binom{n-1}{j}^2 \left(\frac{1}{2}d\right)^{2j}\left(1-\frac{1}{2}d\right)^{2n-2-2j}\right) \\
&\geq \frac{1}{2} + \frac{1}{2^{2n-1}}\binom{2n-2}{n-1} \\
&\geq \frac{1}{2} + \frac{1}{4\sqrt{n}}
\end{aligned}
$$

where we make use of Lemma 3, and then bounds on central binomial coefficients [30]. We can now obtain the required probability of having a one in the bit position k, as follows,

$$
\begin{aligned}
\Pr(z_k = 1) \geq\ & \frac{1}{4} + \frac{1}{2}\left(\frac{1}{2}(1-d) + d\left(\frac{1}{2} + \frac{1}{4\sqrt{n}}\right)\right) \\
=\ & \frac{1}{2} + \frac{d}{8\sqrt{n}}
\end{aligned}
$$

The rest of the proof follows as before. □

3.4. Partial evaluation based on a random subset of bits

An alternative method of partial evaluation, is where a fixed size subset of bits is chosen randomly. Let S be a set of randomly chosen bit positions of size $|S| = s$. Then, the noisy OneMax fitness evaluation is defined as,

$$f^{noisy}(x) = \sum_{k\in S} x_k$$

This models the situation where a machine learning algorithm is trained on a small randomly chosen sample of instances, but is expected to generalise across all instances. Suppose there are n possible instances. A trained classifier, corresponding to certain learned parameters, will either correctly or incorrectly classify each instance. Thus, each trained model has a corresponding bit string in $\{0,1\}^n$. It is typically impractical to train and test models on all n instances, however, so when comparing two models, we look at a small random subset of instances, and prefer the model that performs best on those. Our model is artificial, of course, because we will assume that correct bit values in different positions are independent, and this will typically not be the case in a real machine learning situation.

For our analysis, then, when we compare two strings, we compare them on the same, randomly chosen, subset of bits of size s. A new subset is chosen randomly for each comparison.

Theorem 7. *If $\mu \geq 32(c+1)\frac{n^2}{s}\ln n$, then the Voting algorithm correctly solves* OneMax *with probability greater than $1 - 1/n^c$, where the tournament selection is performed with respect to s randomly sampled bits.*

Proof. When comparing two uniform random strings x and y, to choose a winner z, let S be the set of the s bits that are sampled. Then, the probability that the tournament winner has a one in position k is

$$\begin{aligned}\Pr(z_k = 1) &= \Pr(z_k = 1 \mid k \in S)\Pr(k \in S) \\ &\quad + \Pr(z_k = 1 \mid k \notin S)\Pr(k \notin S) \\ &\geq \left(\frac{1}{2} + \frac{1}{8\sqrt{s}}\right)\frac{s}{n} + \frac{1}{2}\left(1 - \frac{s}{n}\right) \\ &= \frac{1}{2} + \frac{\sqrt{s}}{8n}\end{aligned}$$

The rest of the proof follows as before, following the same arguments with the help of the union bound and Hoeffding's inequality (Lemma 2). □

It is to be noted that when all the bits are known during the tournament selection, i.e. $s = n$, the Voting algorithm requires $O(n \ln n)$ function evaluations as before.

3.5. The deceptive oracle

An oracle may perform comparisons of strings, and say which is better (according to the hidden target string and corresponding objective function). We consider now the case where this comparison is noisy — not because the underlying objective function has noise, but because there is a fixed probability that the worse string is identified as the better one. This kind of uncertainty has been considered in the context of comparison-based optimization methods for sorting [38] and clustering [39]. We suppose that the probability that the oracle returns the incorrect result is l.

Theorem 8. *Let $x, y \in \{0,1\}^n$ be uniformly at random chosen strings. Let the binary tournament winner z be decided according to the* OneMax *function, subject to the oracle returning the incorrect answer with probability $l < 1/7$. That is, if $|x|_1 > |y|_1$ then*

$$z = \begin{cases} x & \text{with probability } 1 - l \\ y & \text{with probability } l \end{cases}$$

Then the probability that there will be a one at position k of the binary tournament winner is at least,

$$\frac{1}{2} + \frac{1}{8\sqrt{n}}(1 - 7l)$$

and the Voting algorithm solves the OneMax *problem in the presence of a noisy comparison oracle with high probability in $O\left(\frac{1}{(1-7l)^2} n \ln n\right)$ function evaluations.*

Proof. As in the previous analyses, with the use of theorem of total probability and Bayes' theorem, the probability that the tournament winner has a one in position k is given by,

$$\begin{aligned}\Pr(z_k = 1) = &\ \Pr(x_k = 1 \mid x \text{ wins}) \\ = &\ \Pr(x_k = 1 \mid x \text{ wins, oracle correct})(1 - l) \\ &+ \ \Pr(x_k = 1 \mid x \text{ wins, oracle incorrect})l \\ = &\ \Pr(x \text{ wins} \mid \text{oracle correct}, x_k = 1)(1 - l) \\ &+ \Pr(x \text{ wins} \mid \text{oracle incorrect}, x_k = 1)l\end{aligned}$$

Now,

$$\begin{aligned}&\Pr(x \text{ wins} \mid \text{oracle correct}, x_k = 1) \\ &= \Pr(x \text{ wins} \mid \text{oracle correct}, x_k = 1, y_k = 1)\Pr(y_k = 1) \\ &+ \Pr(x \text{ wins} \mid \text{oracle correct}, x_k = 1, y_k = 0)\Pr(y_k = 0) \\ &\geq \frac{1}{4} + \frac{1}{2}\Pr\left(\sum_{i \neq k} x_i + 1 > \sum_{i \neq k} y_i\right) \\ &= \frac{1}{4} + \frac{1}{2}\left(\frac{1}{2} + \frac{1}{4\sqrt{n}}\right) \\ &= \frac{1}{2} + \frac{1}{8\sqrt{n}}\end{aligned}$$

Again,

$$
\begin{aligned}
&\Pr(x \text{ wins} \mid \text{oracle incorrect}, x_k = 1) \\
&= \Pr(x \text{ wins} \mid \text{oracle incorrect}, x_k = 1, y_k = 1) \Pr(y_k = 1) \\
&+ \ \Pr(x \text{ wins} \mid \text{oracle incorrect}, x_k = 1, y_k = 0) \Pr(y_k = 0) \\
&\geq \frac{1}{4} + \frac{1}{2} \Pr\left(\sum_{i \neq k} (y_i - x_i) > 1 \right) \\
&= \frac{1}{4} + \frac{1}{2} \left[\Pr\left(\sum_{i \neq k} (y_i - x_i) \geq 0 \right) - \Pr\left(\sum_{i \neq k} (y_i - x_i) = 0 \right) - \Pr\left(\sum_{i \neq k} (y_i - x_i) = 1 \right) \right] \\
&\geq \frac{1}{4} + \frac{1}{2} \left[\frac{1}{2} + \frac{1}{2} \Pr\left(\sum_{i \neq k} (y_i - x_i) = 0 \right) - 2 \Pr\left(\sum_{i \neq k} (y_i - x_i) = 0 \right) \right] \\
&= \frac{1}{4} + \frac{1}{2} \left[\frac{1}{2} - \frac{3}{2} \Pr\left(\sum_{i \neq k} (y_i - x_i) = 0 \right) \right] \\
&= \frac{1}{4} + \frac{1}{2} \left(\frac{1}{2} - \frac{3}{2\sqrt{n}} \right) \\
&= \frac{1}{2} - \frac{3}{4\sqrt{n}}
\end{aligned}
$$

where we have used

$$
\begin{aligned}
\Pr\left(\sum_{i \neq k} (y_i - x_i) = 0 \right) &= \sum_{j=0}^{n-1} \Pr\left(\sum_{i \neq k} y_k = j \right) \Pr\left(\sum_{i \neq k} x_k = j \right) \\
&= \frac{1}{2^{2n-2}} \sum_{j=0}^{n-1} \binom{n-1}{j}^2 \\
&= \frac{1}{2^{2n-2}} \binom{2n-2}{n-1} \\
&\leq \frac{1}{\sqrt{\pi}\sqrt{n-1}} \\
&\leq \frac{1}{\sqrt{n}}
\end{aligned}
$$

Then $\Pr(z_k = 1)$ becomes,

$$
\begin{aligned}
\Pr(z_k = 1) = \ & \Pr(x_k = 1 \mid x \text{ wins}) \\
= \ & \left(\frac{1}{2} + \frac{1}{8\sqrt{n}} \right)(1 - l) + \left(\frac{1}{2} - \frac{3}{4\sqrt{n}} \right) l \\
= \ & \frac{1}{2} + \frac{1}{8\sqrt{n}}(1 - 7l)
\end{aligned}
$$

The final result follows from Lemma 2. □

4. Conclusions

We have studied the use of Voting as a heuristic method. It is particularly effective for the noisy OneMax problem with different variants of noise. We prove that the upper bounds on the runtime of OneMax with posterior and prior noise are better than any other algorithm we are aware of. In case of partial evaluation of fitness functions, as well, the Voting algorithm would require significantly fewer function evaluations than previously published results. We also analyse the runtime on OneMax with two other variants of noise relevant in learning and optimisation problems and show that Voting can be an efficient method in simple noisy combinatorial optimisation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] S. Droste, T. Jansen, I. Wegener, A new framework for the valuation of algorithms for black-box-optimization, in: Foundations of Genetic Algorithms Conference, vol. 3, 2002, pp. 253–270.
[2] G. Anil, R.P. Wiegand, Black-box search by elimination of fitness functions, in: Proceedings of the 10th ACM/SIGEVO Workshop on Foundations of Genetic Algorithms, 2009, pp. 67–78.
[3] Y. Akimoto, S. Astete-Morales, O. Teytaud, Analysis of runtime of optimization algorithms for noisy functions over discrete codomains, Theor. Comput. Sci. 605 (2015) 42–50.
[4] B. Doerr, A.M. Sutton, When resampling to cope with noise, use median, not mean, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2019, pp. 242–248.
[5] T. Friedrich, T. Kötzing, M.S. Krejca, A.M. Sutton, The compact genetic algorithm is efficient under extreme Gaussian noise, IEEE Trans. Evol. Comput. 21 (2017) 477–490.
[6] A. Prugel-Bennett, J. Rowe, J. Shapiro, Run-time analysis of population-based evolutionary algorithm in noisy environments, in: Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII, 2015, pp. 69–75.
[7] S. Droste, Analysis of the $(1+1)$EA for a noisy OneMax, in: Genetic and Evolutionary Computation Conference, Springer, 2004, pp. 1088–1099.
[8] C. Qian, Y. Yu, K. Tang, Y. Jin, X. Yao, Z.-H. Zhou, On the effectiveness of sampling for evolutionary optimization in noisy environments, Evol. Comput. 26 (2018) 237–267.
[9] C. Bian, C. Qian, Y. Yu, K. Tang, On the robustness of median sampling in noisy evolutionary optimization, Sci. China Inf. Sci. 64 (2021) 1–13.
[10] D. Sudholt, Analysing the robustness of evolutionary algorithms to noise: refined runtime bounds and an example where noise is beneficial, Algorithmica 83 (2021) 976–1011.
[11] D.E. Goldberg, K. Deb, J.H. Clark, Genetic algorithms, noise, and the sizing of populations, Complex Syst. 6 (1991) 333–362.
[12] M. Rattray, J. Shapiro, Noisy fitness evaluation in genetic algorithms and the dynamics of learning, in: Foundations of Genetic Algorithms, Morgan Kaufmann, 1997, pp. 117–139.
[13] C. Gießen, T. Kötzing, Robustness of populations in stochastic environments, Algorithmica 75 (2016) 462–489.
[14] R. Dang-Nhu, T. Dardinier, B. Doerr, G. Izacard, D. Nogneng, A new analysis method for evolutionary optimization of dynamic and noisy objective functions, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2018, pp. 1467–1474.
[15] D.-C. Dang, P.K. Lehre, Efficient optimisation of noisy fitness functions with population-based evolutionary algorithms, in: Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII, ACM, 2015, pp. 62–68.
[16] P.K. Lehre, X. Qin, More precise runtime analyses of non-elitist EAs in uncertain environments, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2021, pp. 1160–1168.
[17] J.E. Rowe, Aishwaryaprajna, The benefits and limitations of voting mechanisms in evolutionary optimisation, in: Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms, 2019, pp. 34–42.
[18] D. Whitley, S. Varadarajan, R. Hirsch, A. Mukhopadhyay, Exploration and exploitation without mutation: solving the jump function in $\theta(n)$ time, in: International Conference on Parallel Problem Solving from Nature, Springer, 2018, pp. 55–66.
[19] T. Friedrich, T. Kötzing, M.S. Krejca, S. Nallaperuma, F. Neumann, M. Schirneck, Fast building block assembly by majority vote crossover, in: Proceedings of the ACM Genetic and Evolutionary Computation Conference, ACM Press, 2016, pp. 661–668.
[20] B. Aboutaib, A.M. Sutton, The influence of noise on multi-parent crossover for an island model GA, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2022, pp. 666–674.
[21] H. Mühlenbein, G. Paass, From recombination of genes to the estimation of distributions I. Binary parameters, in: International Conference on Parallel Problem Solving from Nature, Springer, 1996, pp. 178–187.
[22] G.R. Harik, F.G. Lobo, D.E. Goldberg, The compact genetic algorithm, IEEE Trans. Evol. Comput. 3 (1999) 287–297.
[23] J. Bossek, C. Doerr, P. Kerschke, A. Neumann, F. Neumann, Evolving sampling strategies for one-shot optimization tasks, in: Parallel Problem Solving from Nature–XVI: 16th International Conference, Proceedings, Part I 16, Leiden, the Netherlands, September 5–9, 2020, Springer, 2020, pp. 111–124.
[24] P.K. Lehre, D. Sudholt, Parallel black-box complexity with tail bounds, IEEE Trans. Evol. Comput. 24 (2020) 1010–1024.
[25] P.K. Lehre, C. Witt, Black-box search by unbiased variation, Algorithmica 64 (2012) 623–642.
[26] F. Neumann, D. Sudholt, C. Witt, A few ants are enough: ACO with iteration-best update, in: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, 2010, pp. 63–70.
[27] D. Sudholt, C. Witt, On the choice of the update strength in estimation-of-distribution algorithms and ant colony optimization, Algorithmica 81 (2019) 1450–1489.
[28] H. Gonska, I. Rasa, M.-D. Rusu, Chebyshev-Grüss-type inequalities via discrete oscillations, Bul. Acad. Ştiinţe Repub. Mold. Mat. (2014) 63–89, arXiv: 1401.7908.
[29] I. Gavrea, M. Ivan, On a conjecture concerning the sum of the squared Bernstein polynomials, Appl. Math. Comput. 241 (2014) 70–74.
[30] P. Stanica, Good lower and upper bounds on binomial coefficients, JIPAM. J. Inequal. Pure Appl. Math. 2 (2001) 30.
[31] Aishwaryaprajna, Noisy Combinatorial Optimisation with Evolutionary Algorithms, Ph.D. thesis, School of Computer Science, University of Birmingham, 2022.
[32] Aishwaryaprajna, J.E. Rowe, Noisy combinatorial optimisation by evolutionary algorithms, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2019, pp. 139–140.
[33] Aishwaryaprajna, J.E. Rowe, Evolutionary algorithms for solving unconstrained, constrained and multi-objective noisy combinatorial optimisation problems, arXiv preprint, arXiv:2110.02288, 2021.
[34] Aishwaryaprajna, J.E. Rowe, Evolutionary and estimation of distribution algorithms for unconstrained, constrained and multi-objective noisy combinatorial optimisation problems, Evol. Comput. (2023) 1–27.
[35] C. Qian, Y. Yu, Z.-H. Zhou, Analyzing evolutionary optimization in noisy environments, Evol. Comput. 26 (2018) 1–41.
[36] D.-C. Dang, P.K. Lehre, Runtime analysis of non-elitist populations: from classical optimisation to partial information, Algorithmica 75 (2016) 428–461.

[37] C. Qian, C. Bian, W. Jiang, K. Tang, Running time analysis of the (1+1) EA for OneMax and LeadingOnes under bit-wise noise, Algorithmica 81 (2019) 749–795.
[38] T. Gavenčiak, B. Geissmann, J. Lengler, Sorting by swaps with noisy comparisons, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2017, pp. 1375–1382.
[39] R. Addanki, S. Galhotra, B. Saha, How to design robust algorithms using noisy comparison oracle, arXiv:2105.05782, 2021.