Algorithm Portfolios for Noisy Optimization: Compare Solvers Early

Marie-Liesse Cauwet, Jialin Liu, Olivier Teytaud

Algorithm Portfolios for Noisy Optimization: Compare Solvers Early

Marie-Liesse Cauwet, Jialin Liu, and Olivier Teytaud

TAO, INRIA-CNRS-LRI, Univ. Paris-Sud,
91190 Gif-sur-Yvette, France
firstname.lastname@lri.fr
https://tao.lri.fr

Abstract. Noisy optimization is the optimization of objective functions corrupted by noise. A portfolio of algorithms is a set of algorithms equipped with an algorithm selection tool for distributing the computational power among them. We study portfolios of noisy optimization solvers, show that different settings lead to different performances, obtain mathematically proved performance (in the sense that the portfolio performs nearly as well as the best of its algorithms) by an ad hoc selection algorithm dedicated to noisy optimization. A somehow surprising result is that it is better to compare solvers with some lag; i.e., recommend the *current* recommendation of the best solver, selected from a comparison based on their recommendations *earlier* in the run.

1 Introduction

Given an objective function, also termed fitness function, from a domain $\mathcal{D} \in \mathbb{R}^d$ to $\mathbb{R}$, numerical optimization or simply optimization, is the research of points, also termed individuals or search points, with approximately optimum (e.g. minimum) objective function values.

Noisy optimization is the optimization of objective functions corrupted by noise. Black-box noisy optimization is the noisy counterpart of black-box optimization, i.e. functions for which no knowledge about the internal processes involved in the objective function can be exploited.

Algorithm Selection (AS) consists in choosing, in a portfolio of solvers, the one which is (approximately) most efficient on the problem at hand. AS can mitigate the difficulties for choosing a priori the best solver among a portfolio of solvers. This means that AS leads to an adaptive version of the algorithms. In some cases, AS outperforms all individual solvers by combining the good properties of each of them (with information sharing or with chaining, as discussed later). It can also be used for the sake of parallelization or parameter tuning. In this paper, we apply AS to the black-box noisy optimization problem.

1.1 Noisy optimization

Noisy optimization is a key component of machine learning from supervised learning to unsupervised or reinforcement learning; it is also relevant in stream-

ing applications. The black-box setting is crucial in reinforcement learning where gradients are difficult and expensive to get; direct policy search [31] usually boils down to (i) choosing a representation and (ii) black-box noisy optimization.

Order-zero methods, including evolution strategies [6] and derivative-free optimization [11] are natural solutions in such a setting; as they do not use gradients, they are not affected by the black-box scenario. However, the noise has an impact even on such methods [3, 28]. Using surrogate models [20] reduces the impact of noise by sharing information over the domain. Surrogate models are also a step towards higher order methods; even in black-box scenarios, a Hessian can be approximated thanks to observed fitness values.

[12, 29] have shown that stochastic gradient by finite differences (finite differences at each iteration or by averaging over multiple iterations) can provide tight convergence rates (see tightness in [9]) in the case of an additive noise with constant variance. [13] has also tested the use of second order information. Algorithms such as evolution strategies[19] are efficient (log-linear convergence rate) with variance decreasing to zero around the optimum. We will consider a parametrized objective function (Eq. 6), with some parameter z such that $z = 0$ corresponds to a noise variance $\Theta(1)$ in the neighborhood of the optimum; whereas large values of z correspond to noise variance quickly decreasing to 0 around the optimum.

In this paper, our portfolio will be made of the following algorithms: (i) an evolution strategy; (ii) a first-order method using gradients estimated by finite differences (two variants included); (iii) a second-order method using a Hessian matrix approximated also by finite differences. We present these methods in more details in Section 2.

Simple regret criterion. In the black-box setting, let us define :

- x_n the n^{th} search point at which the objective function (also termed fitness function) is evaluated;
- $\tilde{x}_n$ the point that the solver recommends as an approximation of the optimum after having evaluated the objective function at $x_1, \ldots, x_n$ (i.e. after spending n evaluations from the budget).

Some algorithms make no difference between x_n and $\tilde{x}_n$, but in a noisy optimization setting the difference usually matters.

The simple regret for noisy optimization is expressed in terms of objective function values, as follows:

$$SR_n = \mathbb{E}\left(f(\tilde{x}_n) - f(x^*)\right), \tag{1}$$

where $f : \mathcal{D} \mapsto \mathbb{R}$ is a noisy fitness function and x^* minimizes $x \mapsto \mathbb{E}f(x)$. SR_n is the simple regret after n evaluations; n is then the budget.

The slope of the simple regret is then defined as

$$\limsup_n \frac{\log(SR_n)}{\log n}. \tag{2}$$

For example, the gradient method proposed in [12] (approximating the gradient by finite differences) reaches a simple regret slope -1 on sufficiently smooth

problems, for an additive centered noise, without assuming variance decreasing to zero around the optimum.

1.2 Algorithm selection

Combinatorial optimization is probably the most classical application domain for algorithm selection [23]. However, machine learning is also a classical test case for algorithm selection [32]; in this case, algorithm selection is sometimes referred to as meta-learning [1].

No free lunch. [34] claims that it is not possible to do better, on average (uniform average) on all optimization problems from a given finite domain to a given finite codomain. This implies that no algorithm selection can improve existing algorithms on average on this uniform probability distribution of problems. Nonetheless, reality is very different from a uniform average of optimization problems, and algorithm selection does improve performance in many cases.

Chaining and information sharing. Algorithm chaining [7] means switching from one solver to another during the portfolio optimization run. More generally, an hybrid algorithm is a combination of existing algorithms by any means [33]. This is an extremal case of sharing. Sharing consists, more generally, in sending information from some solvers to other solvers; they communicate in order to improve the overall performance.

Static portfolios & parameter tuning. A portfolio of algorithms is usually static, i.e. combines a finite number of given solvers. SatZilla is probably the most well known portfolio method, combining several SAT-solvers [25]. [27] has pointed out the importance of having "orthogonal" solvers in the portfolio, so that the set of solvers is not too large, but covers as far as possible the set of possible solvers. [35] combines algorithm selection and parameter tuning; parameter tuning can be viewed as an algorithm selection over a large but structured space of solvers. We refer to [23] and references therein for more information on parameter tuning and its relation to algorithm selection; this is beyond the scope of this paper.

Fair or unfair sharing of computation budgets. In [26], different strategies are compared for distributing the computation time over different solvers. The first approach consists in running all solvers during a finite time, then selecting the best performing one, and then keep it for all the remaining time. Another approach consists in running all solvers with the same time budget independently of their performance on the problem at hand. Surprisingly enough, they conclude that uniformly distributing the budget is a good and robust strategy. The situation changes when a training set is available, and when we assume that the training set is relevant for the future problems to be optimized; [21], using a training set of problems for comparing solvers, proposes to use 90% of the time allocated to the best performing solver, the other 10% being equally distributed among other solvers. [14, 15] propose 50% for the best solver, 25% for the second best, and so on. Some selection solvers[15, 2] do not need a separate training phase, and performs entirely online solver selection; a weakness of this approach is that it is only possible when a large enough budget is available, so

that the training phase has a minor cost. At the moment, the case of portfolios of noisy optimization solvers has not been discussed in the literature.

Restart strategies. A related problem is the restart of stochastic strategies: when should we restart a local optimization solver ? Deciding if an optimization solver should "restart" is related to deciding if we should switch to another optimization solver; this is relevant for our continuous optimization case below. [18, 10, 30] propose strategies which are difficult to apply in a black-box scenario, when the optimum fitness value is not known.

Parallelism. Portfolios can naturally benefit from parallelism; however, the situation is different in the noisy case, which is highly parallel by nature (as noise is reduced by averaging multiple resamplings); we refer to [17] for more on parallel portfolio algorithms (though not on the noisy optimization case).

Cooperation and information sharing. A crucial question in portfolio algorithms is how to make different solvers in the portfolio cooperate, instead of just competing. Knowledge sharing has been shown to provide great improvements in domains where a concise information (such as inconsistent assignments in satisfiability problems) can save up a huge computation time [17]; it is not easy to see what type of information can be shared in noisy optimization. Already established upper bounds on possible fitness values (in minimization) can help for deciding restarts as detailed above; good approximate solutions can also be shared, possibly leading to a diversity loss. We will investigate in this paper the sharing of current approximate solutions.

Bandit literature. During the last decade, a wide literature on bandits [24, 4, 8] have proposed many tools for distributing the computational power over stochastic options to be tested. The application to our context is however far from being straightforward. In spite of some adaptations to other contexts (time varying as in [22] or adversarial [16, 5]), maybe due to deep differences such as the very non-stationary nature of bandit problems involved, these methods did not, for the moment, really found their way to selection algorithms.

In this paper, we will focus on (i) designing an orthogonal portfolio (ii) distributing the budget nearly equally over possible solvers (iii) possibly sharing information between the different solvers.

1.3 Outline of this paper

Section 2 introduces several noisy optimization solvers. Section 3 explains the portfolio algorithm we applied on top of it. Section 4 provides experimental results.

2 Noisy optimization solvers

Following [27], we will focus on selecting a portfolio of solvers with some "orthogonality", i.e. as different as possible from each other. We selected two extremal cases of Fabian's algorithm [12], a self-adaptive evolutionary algorithm with resamplings, and a variant of Newton's algorithm adapted for noisy optimization. These solvers are more precisely defined in Algs. 2, 3, 1.

3 Algorithms and analysis

3.1 Definitions and Notations

In all the paper, $\mathbb{N}^* = \{1, 2, 3, \dots\}$. Let $f : \mathcal{D} \to \mathbb{R}$ be a noisy function. f is a random process, and equivalently it can be viewed as a mapping $(x, \omega) \mapsto f(x, \omega)$, where

- the user can only choose x;
- and a random variable ω is independently sampled at each call to f.

For short, we will keep the notation $f(x)$; the reader should keep in mind that this function is stochastic. A black-box noisy optimization solver, here referred to as a solver, is a program which aims at finding the minimum x^* of $x \mapsto \mathbb{E}f(x)$, thanks to multiple black-box calls to the unknown function f. The portfolio algorithm, using algorithm selection, has the same goal, and can use M different given solvers; a good algorithm selection tool should ensure that it is nearly as efficient as each of the individual solvers, for any problem in some class of interest. If X is a random variable, then $(X^{(1)}, X^{(2)}, \dots)$ denotes a sample of independent identically distributed random variables, copies of X. Let $k : \mathbb{N}^* \to \mathbb{N}^*$ be a non-decreasing function, called lag function, such that for all $n \in \mathbb{N}^*$, $k(n) \leq n$. For any $i \in \{1, \dots, M\}$, $\tilde{x}_{i,n}$ denotes the point

- that the solver number i *recommends* as an approximation of the optimum (see Section 1.1 for more on the difference between evaluated and recommended search points);
- after this solver has spent n evaluations from the budget.

Similarly, the simple regret given by Eq. 1 corresponding to solver number i after n evaluations, is denoted by $SR_{i,n}$.

For $n \in \mathbb{N}^*$, i_n^* denotes the solver chosen by the selection algorithm after n function evaluations per solver.

Another important concept is the two kinds of terms in the regret of the portfolio.

Definition : Solvers' regret. The solvers' regret with index n, denoted $SR_n^{Solvers}$, is the minimum simple regret among the solvers after n evaluations each, i.e $SR_n^{Solvers} := \min_{i \in \{1, \dots M\}} SR_{i,n}$.

Definition : Selection regret. The selection regret with index n, denoted by $SR_n^{Selection}$ includes the additional regret due to mistakes in choosing among these M solvers, i.e $SR_n^{Selection} := \mathbb{E}\left(f(\tilde{x}_{i_n^*,n}) - f(x^*)\right)$.

3.2 Simple Case : Uniform Portfolio NOPA

We present a simple noisy optimization portfolio algorithm (NOPA) which does not apply any sharing and distributes the computational budget equally over the noisy optimization solvers. Consider an increasing sequence $r_1, \dots, r_n, \dots$ with

values in $\mathbb{N}^*$. These numbers are iteration indices, at which the M recommendations from the M solvers are compared. Consider a sequence $s_1, \ldots, s_n, \ldots$ with values in $\mathbb{N}^*$; $\forall n \in \mathbb{N}^*$, s_n is the number of resamplings of $f(\tilde{x}_{i,n}), \forall i \in \{1, \ldots, M\}$ at iteration r_n; these resamplings are used for comparing these recommendations. More precisely, NOPA works as follows:

- Iteration 1: one evaluation for solver 1, one evaluation for solver 2, ..., one evaluation for solver M.
- Iteration 2: one evaluation for solver 1, one evaluation for solver 2, ..., one evaluation for solver M.
- ...
- Iteration r_1: one evaluation for solver 1, one evaluation for solver 2, ..., one evaluation for solver M.
- **Selection Algorithm:** Evaluate $X = \{\tilde{x}_{1,k(r_1)}, \ldots, \tilde{x}_{M,k(r_1)}\}$, each of them s_1 times; for $m \in \{r_1, \ldots, r_2 - 1\}$, the recommendation of the selection algorithm is $\tilde{x}_{i^*_{r_1},m}$ with $i^*_{r_1} = \underset{i \in \{1,\ldots,M\}}{\arg\min} \sum_{\ell=1}^{s_1} f(\tilde{x}_{i,k(r_1)})^{(\ell)}$.
- Iteration $r_1 + 1$: one evaluation for solver 1, one evaluation for solver 2, ..., one evaluation for solver M.
- ...
- Iteration $r_2 - 1$: one evaluation for solver 1, one evaluation for solver 2, ..., one evaluation for solver M.
- Iteration r_2: one evaluation for solver 1, one evaluation for solver 2, ..., one evaluation for solver M.
- **Selection Algorithm:** Evaluate $X = \{\tilde{x}_{1,k(r_2)}, \ldots, \tilde{x}_{M,k(r_2)}\}$, each of them s_2 times; for $m \in \{r_2, \ldots, r_3 - 1\}$, the recommendation of the selection algorithm is $\tilde{x}_{i^*_{r_2},m}$ with $i^*_{r_2} = \underset{i \in \{1,\ldots,M\}}{\arg\min} \sum_{\ell=1}^{s_2} f(\tilde{x}_{i,k(r_2)})^{(\ell)}$.
- ...
- Iteration r_n: one evaluation for solver 1, one evaluation for solver 2, ..., one evaluation for solver M.
- **Selection Algorithm:** Evaluate $X = \{\tilde{x}_{1,k(r_n)}, \ldots, \tilde{x}_{M,k(r_n)}\}$, each of them s_n times; for $m \in \{r_n, \ldots, r_{n+1} - 1\}$, the recommendation of the selection algorithm is $\tilde{x}_{i^*_{r_n},m}$ with $i^*_{r_n} = \underset{i \in \{1,\ldots,M\}}{\arg\min} \sum_{\ell=1}^{s_n} f(\tilde{x}_{i,k(r_n)})^{(\ell)}$.

Please note that

- **Stable choice of solver:** The selection algorithm follows the recommendation of the same solver $i^*_{r_t}$ at all iterations in $\{r_t, \ldots, r_{t+1} - 1\}$.
- **Use of solvers' current recommendations:** But for such iteration indices m and p in $\{r_t, \ldots, r_{t+1} - 1\}$, the portfolio does not necessarily recommends the same point because possibly $\tilde{x}_{i^*_{r_t},m} \neq \tilde{x}_{i^*_{r_t},p}$.
- Please note that, in this algorithm, we compare, at iteration r_n, recommendations chosen at iteration $k(r_n)$; and this comparison is based on s_n resamplings.

Effect of the lag: due to $k(.)$, we compare recommendations from *earlier* iterations. This is somehow surprising, because the optimum solver at iteration

$k(n)$ might be different from the optimum solver at iteration n. However, the key point, in this algorithm, is that comparing recommendations at iteration $k(r_n)$ is much cheaper than comparing recommendations at iteration r_n. This is because at iteration $k(r_n)$, points are not that good, and therefore can be compared with a budget *smaller than* r_n - which is necessary for not wasting evaluations.

We see that there are two kinds of evaluations:

- **Portfolio budget:** this corresponds to the M evaluations per iteration, dedicated to running the M solvers (one evaluation per solver and per iteration).
- **Comparison budget (algorithm selection step):** this corresponds to the s_n evaluations per solver. This is a key difference with deterministic optimization; in deterministic optimization, this budget is zero as the exact fitness value is readily available.

We have Mr_n evaluations in the portfolio budget for the r_n first iterations. We will see below conditions under which the other costs can be made negligible, whilst preserving the same regret as the best of the M solvers.

3.3 Theoretical analysis : The log(M)-shift

Main property: regret of NOPA. *Let $(r_n)_{n\in\mathbb{N}^*}$ and $(s_n)_{n\in\mathbb{N}^*}$ be some sequences as in Section 3.2. Assume that :*

- $\forall x \in \mathcal{D},\ Var\ f(x) \leq 1$*;*
- *for some positive sequence $(\epsilon_n)_{n\in\mathbb{N}^*}$, almost surely, there exists some $n_0 \in \mathbb{N}^*$ such that :*

$$\forall n \geq n_0,\ SR^{Solvers}_{k(r_n)} < \min_{i \notin \operatorname*{arg\,min}_{j\in\{1,\dots,M\}} SR_{j,k(r_n)}} SR_{i,k(r_n)} - 2\epsilon_n; \tag{3}$$

$$\text{and } \forall n \geq k(r_{n_0}),\ \operatorname*{arg\,min}_{i\in\{1,\dots,M\}} SR_{i,n} = \operatorname*{arg\,min}_{i\in\{1,\dots,M\}} SR_{i,n+1}. \tag{4}$$

Then, almost surely there exists some n_0 such that for any $n > n_0$, NOPA has simple regret $SR^{Selection}_{r_n}$ equal to $SR^{Solvers}_{r_n}$ with probability at most $1 - \frac{M}{s_n\epsilon_n^2}$ after $e_n = r_n \times M \times (1 + \sum_{i=1}^{n} \frac{s_i}{r_n})$ evaluations.

Corollary 1 : Asymptotic case.
Under assumptions above and if $s_n\epsilon_n^2 \to \infty$ for some sequence ϵ_n satisfying Eq. 3, and $\frac{1}{r_n}\sum_{i=1}^{n} s_i = o(1)$, then the regret $SR^{Selection}_m$ of the portfolio after $Mm(1+o(1))$ evaluations is at most $SR^{Solvers}_m$ with probability converging to 1 as $m \to \infty$.

Corollary 2 : the $\log(M)$ shift.
Let $r > 1$ and $r' > 0$, $\forall n \in \mathbb{N}^*$, the following parametrization satisfies the assumptions of Corollary 1 for some sequence $\epsilon_n = \Theta(\frac{1}{n})$ satisfying Eq. 3:

- $r_n = n^{1+2r+r'}$,
- $s_n = n^{2r}$ *and*
- $k(n) = \lceil n^{1/(1+2r+r')} \rceil$.

Notice that the comparison budget (sum of the s_n) increases polynomially, slower than the portfolio budget. Moreover, in the case of a constant variance noise, typical rates are SR_n scaling as $O(1/n)$ (see e.g. [12, 9, 29]). Hence, with these parameters or others parameters which satisfy the assumptions of Corollary 1, on classical log-log graphs (x-axis: log(number of evaluations); y-axis: log(simple regret)), cf Eq. 2, the portfolio should perform similarly to the best solver, within the $\log(M)$ shift on the x-axis.

Remark on Corollary 2: Corollary 2 holds under assumption Eq. 3. This means that the two best solvers have a difference of order $1/n$. In order to get similar results when solvers are very close to each other (ϵ_n smaller), it is necessary to use a slower k function.

Proof of the main property:
First, notice that the total number of evaluations, up to the construction of $\tilde{x}_{i^*_{r_n},r_n}$ at iteration r_n, is: $M(r_n+\sum_{i=1}^{n} s_i)$ whereas each solver has spent r_n evaluations.

Let us denote $\hat{\mathbb{E}}_s[f(x)]$ the empirical evaluation of $\mathbb{E}[f(x)]$ over s resamplings, i.e. $\hat{\mathbb{E}}_s[f(x)] := \frac{1}{s}\sum_{j=1}^{s}(f(x))^{(j)}$.

By Chebyshev's inequality,

$$P(|\mathbb{E}\left[f(x_{i,k(r_n)})\right] - \hat{\mathbb{E}}_{s_n}\left[f(x_{i,k(r_n)})\right]| > \epsilon_n) \leq \frac{Var\left[f\left(x_{i,k(r_n)}\right)\right]}{s_n\epsilon_n^2} \leq \frac{1}{s_n\epsilon_n^2}.$$

By union bound,

$$P(\exists i \in \{1,\dots,M\}; |\mathbb{E}\left[f(x_{i,k(r_n)})\right] - \hat{\mathbb{E}}_{s_n}\left[f(x_{i,k(r_n)})\right]| > \epsilon_n) \leq \frac{M}{s_n\epsilon_n^2}.$$

With notation $i^* = i^*_{r_n} := \underset{i\in\{1,\dots,M\}}{\arg\min}\ \hat{\mathbb{E}}_{s_n}\left[f(\tilde{x}_{i,k(r_n)})\right]$, it follows that, with probability $1-\frac{M}{s_n\epsilon_n^2}$:

$$\begin{aligned}
\mathbb{E}\left[f(\tilde{x}_{i^*,k(r_n)})\right] &\leq \hat{\mathbb{E}}_{s_n}\left[f(\tilde{x}_{i^*,k(r_n)})\right] + \epsilon_n;\\
\mathbb{E}\left[f(\tilde{x}_{i^*,k(r_n)})\right] &\leq \hat{\mathbb{E}}_{s_n}\left[f(\tilde{x}_{j,k(r_n)})\right] + \epsilon_n,\ \forall j \in \{1,\dots,M\};\\
\mathbb{E}\left[f(\tilde{x}_{i^*,k(r_n)})\right] &\leq \mathbb{E}\left[f(\tilde{x}_{j,k(r_n)})\right] + 2\epsilon_n,\ \forall j \in \{1,\dots,M\};\\
\mathbb{E}\left[f(\tilde{x}_{i^*,k(r_n)})\right] - \mathbb{E}\left[f(x^*)\right] &\leq \min_{j\in\{1,\dots,M\}} SR_{j,k(r_n)} + 2\epsilon_n;\\
SR_{i^*,k(r_n)} &< \min_{i\notin \underset{j\in\{1,\dots,M\}}{\arg\min}\ SR_{j,k(r_n)}} SR_{i,k(r_n)}. \qquad (5)
\end{aligned}$$

By Eqs. 3 and 5, $i^* \in \underset{i\in\{1,\dots,M\}}{\arg\min}\ SR_{i,k(r_n)}$ with probability $1-\frac{M}{s_n\epsilon_n^2}$, by Eq. 4, $i^* \in \underset{i\in\{1,\dots,M\}}{\arg\min}\ SR_{i,r_n}$. □

3.4 Real world constraints & introducing sharing

Real world introduces various constraints. Most solvers do not allow you to run one single fitness evaluation at a time, so that it becomes difficult to have

exactly the same number of fitness evaluations per solver. We will here adapt the algorithm above for such a case; an additional change is the possible use of "Sharing" options (i.e. sharing information between the different solvers). The proposed algorithm is as follows:

- Iteration 1: one iteration for solver 1, one iteration for solver 2, ..., one iteration for solver M.
- Iteration 2: one iteration for each solver which received less than 2 evaluations.
- ...
- Iteration i: one iteration for each solver which received less than i evaluations.
- ...
- Iteration r_1: one iteration for each solver which received less than r_1 evaluations.
- **Selection Algorithm:** Evaluate $X = \{\tilde{x}_{1,k(r_1)}, \ldots, \tilde{x}_{M,k(r_1)}\}$, each of them s_1 times; the recommendation of NOPA is $\tilde{x}_{i^*,m}$ for iterations $m \in \{r_1, \ldots, r_2 - 1\}$, where $i^* = \operatorname{argmin}_{i \in \{1,\ldots,M\}} \sum_{j=1}^{s_1} f(\tilde{x}_{i,k(r_1)})^{(j)}$. If sharing is enabled, all solvers receive $\tilde{x}_{i^*,r_1}$ as next iterate.
- Iteration $r_1 + 1$: one iteration for each solver which received less than $r_1 + 1$ evaluations.
- ...
- Iteration r_2: one iteration for each solver which received less than r_2 evaluations.
- **Selection Algorithm:** Evaluate $X = \{\tilde{x}_{1,k(r_2)}, \ldots, \tilde{x}_{M,k(r_2)}\}$, each of them s_2 times; the recommendation of NOPA is $\tilde{x}_{i^*,m}$ for iterations $m \in \{r_2, \ldots, r_3 - 1\}$, where $i^* = \operatorname{argmin}_{i \in \{1,\ldots,M\}} \sum_{j=1}^{s_2} f(\tilde{x}_{i,k(r_2)})^{(j)}$. If sharing is enabled, all solvers receive $\tilde{x}_{i^*,r_2}$ as next iterate.
- ...
- Iteration r_n: one iteration for each solver which received less than r_n evaluations.
- **Selection Algorithm:** Evaluate $X = \{\tilde{x}_{1,k(r_n)}, \ldots, \tilde{x}_{M,k(r_n)}\}$, each of them s_n) times; the recommendation of NOPA is $\tilde{x}_{i^*,m}$ for iterations $m \in \{r_n, \ldots, r_{n+1} - 1\}$, where $i^* = \operatorname{argmin}_{i \in \{1,\ldots,M\}} \sum_{j=1}^{s_n} f(\tilde{x}_{i,k(r_n)})^{(j)}$. If sharing is enabled, all solvers receive $\tilde{x}_{i^*,r_n}$ as next iterate.

4 Experimental results

For our experiments below, we use four noisy optimization solvers and portfolio of these solvers with and without information sharing:

- Solver 1: Fabian's solver, as detailed in Alg. 3, with parametrization $\gamma = 0.1$, $a = 1$, $c = 100$. This variant will be termed Fabian1.
- Solver 2: Another Fabian's solver with parametrization $\gamma = 0.49$, $a = 1$, $c = 2$. This variant will be termed Fabian2.

- Solver 3: A version of Newton's solver adapted for black-box noisy optimization (gradients and Hessians are approximated on samplings of the objective function), as detailed in Alg. 1, with parametrization $B = 1$, $\beta = 2$, $A = 100$, $\alpha = 4$. For short this solver will be termed Newton.
- Solver 4: A self-adaptive evolution strategy with resampling as explained in Alg. 2, with parametrization $\lambda = 10d$, $\mu = 5d$, $K = 10$, $\zeta = 2$ (in dimension d). This solver will be termed RSAES (resampling self-adaptive evolution strategy).
- Portfolio: Portfolio of solvers 1, 2, 3, 4. Functions are $k(n) = \lceil n^{0.1} \rceil$, $r_n = n^3$, $s_n = 15n^2$ at iteration n.
- P.+Sharing: Portfolio of solvers 1, 2, 3, 4, with information sharing enabled. Same functions.

We approximate the slope of the linear convergence in log-log scale by the logarithm of the average simple regret divided by the logarithm of the number of evaluations.

Experiments have been performed on

$$f(x) = ||x||^2 + ||x||^z \mathcal{N} \tag{6}$$

with $\mathcal{N}$ a Gaussian standard noise. The results in dimension 2 and dimension 15 are shown in Table 1.

We see on these experiments:

- that the portfolio algorithm successfully reaches almost the same slope as the best of its solvers;
- that for $z = 2$ the best algorithm is the second variant of Fabian (consistently with [12]);
- that for $z = 1$ the approximation of Newton's algorithm performs best;
- that for $z = 0$ the first variant of Fabian's algorithm performs best (consistently with [12]);
- that the sharing has little or no impact.

5 Conclusion

We have seen that noisy optimization provides a very natural framework for portfolio methods. Different noisy optimization algorithms have extremely different rates on different test cases, depending on the noise level, on the dimension. We show mathematically and empirically a $\log(M)$ shift when using M solvers, when working on a classical log-log scale (classical in noisy optimization). Contrarily to noise-free optimization (where a $\log(M)$ shift would be a trivial result), such a shift is not so easily obtained in noisy optimization.

Importantly, it is necessary, for getting the $\log(M)$ shift, that:

- the selection algorithm compares *old* recommendations (and selects a solver from this point of view),

– the portfolio recommends the *current* recommendation of this selected solver.

Sharing information in portfolios of noisy optimization algorithms is not so easy. A further work consists in identifying relevant information for sharing; maybe the estimate of the asymptotic fitness value of a solver is the most natural information for sharing; if a fitness value A is already found and a solver claims that it will never do better than A we can stop its run and save up computational power. This should allow better than the $\log(M)$ shift. Another further work is the extension beyond simple unimodal objective functions; the crucial assumption for our result is that the best algorithm does not change too often, this might not always be the case.

References

1. D. W. Aha. Generalizing from case studies: A case study. In *Proceedings of the 9th International Workshop on Machine Learning*, pages 1–10. Morgan Kaufmann Publishers Inc., 1992.
2. W. Armstrong, P. Christen, E. McCreath, and A. P. Rendell. Dynamic algorithm selection using reinforcement learning. In *International Workshop on Integrating AI and Data Mining*, pages 18–25, 2006.
3. D. V. Arnold and H.-G. Beyer. A general noise model and its effects on evolution strategy performance. *IEEE Transactions on Evolutionary Computation*, 10(4):380–391, 2006.
4. P. Auer. Using confidence bounds for exploitation-exploration trade-offs. *The Journal of Machine Learning Research*, 3:397–422, 2003.
5. P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: the adversarial multi-armed bandit problem. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 322–331. IEEE Computer Society Press, Los Alamitos, CA, 1995.
6. H.-G. Beyer. *The Theory of Evolutions Strategies*. Springer, Heidelberg, 2001.
7. J. Borrett, E. P. K. Tsang, and C. C. Sq. Towards a formal framework for comparing constraint satisfaction problem formulations, 1996.
8. S. Bubeck, R. Munos, and G. Stoltz. Pure exploration in multi-armed bandits problems. In *ALT*, pages 23–37, 2009.
9. H. Chen. Lower rate of convergence for locating the maximum of a function. *Annals of statistics*, 16:1330–1334, Sept. 1988.
10. V. A. Cicirello and S. F. Smith. The max k-armed bandit: A new model of exploration applied to search heuristic selection. In *Proceedings of the 20th National Conference on Artificial Intelligence*, pages 1355–1361. AAAI Press, 2005.
11. A. Conn, K. Scheinberg, and L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives, 1997.
12. V. Fabian. Stochastic Approximation of Minima with Improved Asymptotic Speed. *Annals of Mathematical statistics*, 38:191–200, 1967.
13. V. Fabian. *Stochastic Approximation*. SLP. Department of Statistics and Probability, Michigan State University, 1971.
14. M. Gagliolo and J. Schmidhuber. A neural network model for inter-problem adaptive online time allocation. In *15th International Conference on Arti cial Neural Networks: Formal Models and Their Applications*, pages 7–12. Springer, 2005.

15. M. Gagliolo and J. Schmidhuber. Learning dynamic algorithm portfolios. volume 47, pages 295–328, 2006.
16. M. D. Grigoriadis and L. G. Khachiyan. A sublinear-time randomized approximation algorithm for matrix games. *Operations Research Letters*, 18(2):53–58, Sep 1995.
17. Y. Hamadi. *Search: from Algorithms to Systems*. PhD thesis, Université Paris-Sud, 2013.
18. E. Horvitz, Y. Ruan, C. P. Gomes, H. A. Kautz, B. Selman, and D. M. Chickering. A bayesian approach to tackling hard computational problems. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, pages 235–244. Morgan Kaufmann Publishers Inc, 2001.
19. M. Jebalia, A. Auger, and N. Hansen. Log-linear convergence and divergence of the scale-invariant (1+1)-es in noisy environments. *Algorithmica*, pages 1–36, 2010. online first.
20. Y. Jin and J. Branke. Evolutionary optimization in uncertain environments. a survey, ieee transactions on evolutionary computation, vol. 9, no. 3, pp. 303 317. 2005.
21. S. Kadioglu, Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann. Algorithm selection and scheduling. In *17th International Conference on Principles and Practice of Constraint Programming*, pages 454–469, 2011.
22. L. Kocsis and C. Szepesvari. Discounted-ucb. In *2nd Pascal-Challenge Workshop*, 2006.
23. L. Kotthoff. Algorithm selection for combinatorial search problems: A survey. *CoRR*, abs/1210.7959, 2012.
24. T. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6:4–22, 1985.
25. E. Nudelman, K. Leyton-Brown, H. H. Hoos, A. Devkar, and Y. Shoham. Understanding random sat: beyond the clauses-to-variables ratio. In M. Wallace, editor, *Principles and Practice of Constraint Programming CP 2004, LLNCS 3258*, volume 3258 of Lecture Notes in Computer Science, pages 438–452. Springer Berlin / Heidelberg, 2004.
26. L. Pulina and A. Tacchella. A self-adaptive multi-engine solver for quantified boolean formulas. *Constraints*, 14(1):80–116, 2009.
27. H. Samulowitz and R. Memisevic. Learning to solve qbf. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, pages 255–260. AAAI, 2007.
28. B. Sendhoff, H.-G. Beyer, and M. Olhofer. The influence of stochastic quality functions on evolutionary search, in recent advances in simulated evolution and learning, ser. advances in natural computation, k. tan, m. lim, x. yao, and l. wang, eds. world scientific, pp 152-172. 2004.
29. O. Shamir. On the complexity of bandit and derivative-free stochastic convex optimization. *CoRR*, abs/1209.2388, 2012.
30. M. J. Streeter, D. Golovin, and S. F. Smith. Restart schedules for ensembles of problem instances. pages 1204–1210. AAAI Press, 2007.
31. R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction.* MIT Press., Cambridge, MA, 1998.
32. P. E. Utgoff. Perceptron trees: A case study in hybrid concept representations. In *National Conference on Artificial Intelligence*, pages 601–606, 1988.
33. V. Vassilevska, R. Williams, and S. L. M. Woo. Confronting hardness using a hybrid approach. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1–10. ACM, 2006.

34. D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr. 1997.
35. L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Hydra-mip: automated algorithm configuration and selection for mixed integer programming. In *RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion at the International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

Algorithm 1 Newton algorithm with gradient and Hessian approximated by finite differences and revaluations.

Parameters: a dimension $d \in \mathbb{N}^*$, $A > 0$, $B > 0$, $\alpha > 0$, $\beta > 0$, $\epsilon > 0$
Input: $\hat{h} \leftarrow$ identity matrix, an initial $x_1 \in \mathbb{R}^d$
$n \leftarrow 1$
while (true) **do**
  Compute

$$\sigma_n = A/n^\alpha$$

  Evaluate the gradient g at x_n by finite differences, averaging over $\lceil Bn^\beta \rceil$ samples at distance $\Theta(\sigma_n)$ of x_n
  for $i = 1$ to d **do**
    Evaluate $h_{i,i}$ by finite differences at $x_n + \sigma e_i$ and $x_n - \sigma e_i$, averaging each evaluation over $\lceil Bn^\beta \rceil$ resamplings
    for $j = 1$ to d **do**
      if $i == j$ **then**
        Update $\hat{h}_{i,j}$ using $\hat{h}_{i,i} = (1 - \epsilon)\hat{h}_{i,i} + \epsilon h_{i,i}$
      else
        Evaluate $h_{i,j}$ by finite differences thanks to evaluations at each of $x_n \pm \sigma e_i \pm \sigma e_j$, averaging over $\lceil Bn^\beta/10 \rceil$ samples
        Update $\hat{h}_{i,j}$ using $\hat{h}_{i,j} = (1 - \frac{\epsilon}{d})\hat{h}_{i,j} + \frac{\epsilon}{d} h_{i,j}$
      end if
    end for
  end for
  $\delta \leftarrow$ solution of $\hat{h}\delta = -g$
  if $\delta > C\sigma_n$ **then**
    $\delta = C\sigma_n \frac{\delta}{||\delta||}$
  end if
  Apply $x_{n+1} = x_n + \delta$
  $n \leftarrow n + 1$
end while

Algorithm 2 Self-adaptive Evolution Strategy with revaluations. $\mathcal{N}$ denotes some independent standard Gaussian random variable, with dimension as required in equations above.

Parameters: $K > 0$, $\zeta \geq 0$, $\lambda \geq \mu > 0$, a dimension $d \in \mathbb{N}^*$
Input: an initial parent population $x_{1,i} \in \mathbb{R}^d$ and an initial $\sigma_{1,i} = 1$, $i \in \{1, \ldots, \mu\}$
$n \leftarrow 1$
while (true) **do**
  Generate λ individuals i_j, $j \in \{1, \ldots, \lambda\}$, independently using

$$\sigma_j = \sigma_{n,mod(j-1,\mu)+1} \times \exp\left(\frac{1}{2d}\mathcal{N}\right) \text{ and } i_j = x_{n,mod(j-1,\mu)+1} + \sigma_j \mathcal{N}$$

  Evaluate each of them $\lceil Kn^\zeta \rceil$ times and average their fitness values
  Define $j_1, \ldots, j_\lambda$ so that

$$\mathbb{E}_{\lceil Kn^\zeta \rceil}[f(i_{j_1})] \leq \mathbb{E}_{\lceil Kn^\zeta \rceil}[f(i_{j_2})] \cdots \leq \mathbb{E}_{\lceil Kn^\zeta \rceil}[f(i_{j_\lambda})]$$

  where $\mathbb{E}_m$ denotes the average over m resamplings
  Update: compute $x_{n+1,k}$ and $\sigma_{n+1,k}$ using

$$\sigma_{n+1,k} = \sigma_{j_k} \text{ and } x_{n+1,k} = i_{j_k},\ k \in \{1, \ldots, \mu\}$$

  $n \leftarrow n + 1$
end while

Algorithm 3 Fabian's stochastic gradient algorithm with finite differences. Several variants have been defined, in particular versions in which only one point (or a finite number of points independently of the dimension) is evaluated at each iteration[9, 29]. We refer to [12] for more details and in particular for the choice of weights and scales.

Parameters: a dimension $d \in \mathbb{N}^*$, $\frac{1}{2} > \gamma > 0$, $a > 0$, $c > 0$, $m \in \mathbb{N}^*$, weights $w_1 > \cdots > w_m$ summing to 1, scales $1 \geq u_1 > \cdots > u_m > 0$
Input: an initial $x_1 \in \mathbb{R}^d$
$n \leftarrow 1$
while (true) **do**
  Compute

$$\sigma_n = c/n^\gamma$$

  Evaluate the gradient g at x_n by finite differences, averaging over $2m$ samples per axis. $\forall i \in \{1, \ldots, d\}, \forall j \{1 \ldots m\}$

$$x_n^{(i,j)+} = x_n + u_j e_i \text{ and } x_n^{(i,j)-} = x_n - u_j e_i$$
$$g_i = \frac{1}{2\sigma_n} \sum_{j=1}^{m} w_j \left(f(x_n^{(i,j)+}) - f(x_n^{(i,j)-}) \right)$$

  Apply $x_{n+1} = x_n - \frac{a}{n} g$
  $n \leftarrow n + 1$
end while

comp. time	algorithm	obtained slope for $d=2$			obtained slope for $d=15$		
		$z=0$	$z=1$	$z=2$	$z=0$	$z=1$	$z=2$
10	Portfolio	-1.00±0.28	-1.63±0.06	-2.69±0.07	-0.72±0.02	-1.06±0.01	-1.90±0.02
10	P.+Sharing	-0.93±0.31	-1.64±0.05	-2.71±0.07	-0.72±0.02	-1.05±0.03	-1.90±0.03
10	Fabian1	-1.24±0.05	-1.25±0.06	-1.23±0.06	-0.83±0.02	-1.03±0.02	-1.02±0.02
10	Fabian2	-0.17±0.09	-1.75±0.10	-3.16±0.06	0.11±0.02	-1.30±0.02	-2.39±0.02
10	Newton	-0.20±0.09	-1.84±0.34	-1.93±0.00	0.00±0.02	-1.27±0.23	-1.33±0.00
10	RSAES	-0.41±0.08	-0.61±0.13	-0.60±0.16	0.15±0.01	0.14±0.02	0.15±0.01
20	Portfolio	-0.92±0.26	-1.58±0.05	-2.66±0.06	-0.70±0.02	-1.02±0.02	-1.85±0.02
20	P.+Sharing	-0.94±0.22	-1.60±0.00	-2.67±0.06	-0.69±0.02	-1.02±0.02	-1.84±0.02
20	Fabian1	-1.20±0.07	-1.25±0.10	-1.24±0.05	-0.83±0.03	-1.01±0.02	-1.02±0.02
20	Fabian2	-0.15±0.06	-1.76±0.06	-3.18±0.06	0.11±0.02	-1.32±0.01	-2.45±0.01
20	Newton	-0.14±0.05	-1.96±0.00	-1.96±0.00	0.00±0.02	-1.32±0.24	-1.39±0.00
20	RSAES	-0.41±0.07	-0.54±0.11	-0.54±0.04	0.12±0.01	0.12±0.02	0.13±0.01
40	Portfolio	-0.91±0.25	-1.60±0.00	-2.63±0.05	-0.69±0.01	-1.03±0.01	-1.86±0.03
40	P.+Sharing	-0.99±0.18	-1.58±0.06	-2.66±0.06	-0.69±0.01	-1.02±0.02	-1.88±0.02
40	Fabian1	-1.21±0.06	-1.21±0.03	-1.19±0.07	-0.82±0.02	-1.00±0.02	-0.99±0.02
40	Fabian2	-0.18±0.07	-1.78±0.09	-3.18±0.07	0.11±0.02	-1.36±0.02	-2.52±0.02
40	Newton	-0.17±0.08	-1.99±0.00	-1.68±0.61	0.00±0.02	-1.32±0.33	-1.45±0.00
40	RSAES	-0.41±0.08	-0.64±0.12	-0.55±0.11	0.11±0.01	0.11±0.01	0.11±0.01
80	Portfolio	-0.92±0.25	-1.61±0.05	-2.65±0.05	-0.68±0.02	-1.02±0.02	-1.85±0.02
80	P.+Sharing	-0.83±0.28	-1.60±0.05	-2.64±0.04	-0.68±0.03	-1.01±0.01	-1.86±0.02
80	Fabian1	-1.15±0.05	-1.20±0.05	-1.22±0.04	-0.82±0.02	-0.99±0.01	-1.00±0.02
80	Fabian2	-0.17±0.09	-1.76±0.07	-3.11±0.09	0.10±0.02	-1.38±0.02	-2.58±0.01
80	Newton	-0.12±0.06	-2.01±0.00	-2.01±0.00	0.00±0.01	-1.42±0.29	-1.50±0.00
80	RSAES	-0.37±0.06	-0.54±0.12	-0.56±0.14	0.10±0.01	0.11±0.01	0.09±0.01
160	Portfolio	-1.01±0.07	-1.61±0.00	-2.67±0.12	-0.65±0.02	-1.01±0.07	-1.89±0.03
160	P.+Sharing	-0.90±0.20	-1.60±0.03	-2.66±0.07	-0.67±0.02	-1.02±0.01	-1.89±0.02
160	Fabian1	-1.14±0.04	-1.20±0.05	-1.19±0.05	-0.83±0.01	-0.98±0.01	-0.98±0.01
160	Fabian2	-0.21±0.08	-1.79±0.04	-2.97±0.06	0.09±0.02	-1.42±0.02	-2.62±0.02
160	Newton	-0.13±0.08	-2.04±0.00	-2.04±0.00	0.00±0.01	-1.48±0.24	-1.55±0.00
160	RSAES	-0.37±0.04	-0.61±0.13	-0.56±0.12	0.09±0.01	0.09±0.01	0.09±0.01

Table 1. Experiments on $f(x) = ||x||^2 + ||x||^z \mathcal{N}$ in dimension 2 and dimension 15. We see that the portfolio successfully keeps the best of each world (nearly same slope as the best). Importantly, without lag (i.e. if we use $k(n) = n$), this property was **not** reproduced. Comp. time refers to the computational time.