Portfolio methods in uncertain contexts

Jialin Liu

UNIVERSITÉ PARIS-SACLAY

ÉCOLE DOCTORALE D'INFORMATIQUE, ED427

INRIA Saclay

DISCIPLINE: COMPUTER SCIENCE

PHD THESIS

Presented and publicly defended on December 11, 2015 by

Jialin Liu

PORTFOLIO METHODS IN UNCERTAIN CONTEXTS

Advisor:	Olivier Teytaud	CR	(INRIA Saclay, LRI/TAO)
Co-advisor:	Marc Schoenauer	DR	(INRIA Saclay, LRI/TAO)

With the following jury:

Reviewers:	Bruno Bouzy	Ass. Professor	(Univ. Paris Descartes)
	Marcus Gallagher	Ass. Professor	(Univ. of Queensland)
Examiners	Philippe Dague	Professor	(Univ. Paris-Saclay)
	Simon Lucas	Professor	(Univ. of Essex)
	Petr Posik	Researcher	(Gerstner Laboratory)
	Günter Rudolph	Professor	(Univ. of Dortmund)
Advisor:	Olivier Teytaud	CR	(INRIA Saclay, LRI/TAO)
Co-advisor:	Marc Schoenauer	DR	(INRIA Saclay, LRI/TAO)

TAO Project Team, INRIA Saclay-CNRS-LRI, Université Paris-Saclay
Bât. Claude Shannon, Rue Noetzlin, 91190 Gif-sur-Yvette, FRANCE

Abstract

This manuscript concentrates in studying methods to handle the noise, including using resampling methods to improve the convergence rates and applying portfolio methods to cases with uncertainties (games, and noisy optimization in continuous domains).

Part I will introduce the manuscript, then review the state of the art in noisy optimization, portfolio algorithm, multi-armed bandit algorithms and games.

Part II concentrates on the work on noisy optimization:

- Chapter 4 provides a generic algorithm for noisy optimization recovering most of the existing bounds in one single noisy optimization algorithm.

- Chapter 5 applies different resampling rules in evolution strategies for noisy optimization, without the assumption of variance vanishing in the neighborhood of the optimum, and shows mathematically log-log convergence results and studies experimentally the slope of this convergence.

- Chapter 6 compares resampling rules used in the differential evolution algorithm for strongly noisy optimization. By mathematical analysis, a new rule is designed for choosing the number of resamplings, as a function of the dimension, and validate its efficiency compared to existing heuristics - though there is no clear improvement over other empirically derived rules.

- Chapter 7 applies "common random numbers", also known as pairing, to an intermediate case between black-box and white-box cases for improving the convergence.

Part III is devoted to portfolio in adversarial problems:

- Nash equilibria are cases in which combining pure strategies is necessary for designing optimal strategies. Two chapters are dedicated to the computation of Nash equilibria:

  - Chapter 9 investigates combinations of pure strategies, when a small set of pure strategies is concerned; basically, we get improved rates when the support of the Nash equilibrium is small.
  - Chapter 10 applies these results to a power system problem. This compares several bandit algorithms for Nash equilibria, defines parameter-free bandit algorithms, and shows the relevance of the sparsity approach discussed in Chapter 9.

- Then, two chapters are dedicated to portfolios of game methods:
  - Chapter 11 shows how to generate multiple policies, from a single one, when only one such policy is available. This kind of bootstrap (based on random seeds) generates many deterministic policies, and then combines them into one better policy. This has been tested on several games.
  - Chapter 12 extends chapter 11 by combining policies in a position-specific manner. In particular, we get a better asymptotic behavior than MCTS.

Part IV is devoted to portfolios in noisy optimization:

- Chapter 14 is devoted to portfolio of noisy optimization methods in continuous domains.
- Chapter 15 proposed differential evolution as a tool for non-stationary bandit problems.

Contents

Contents (detailed)

Part I

Introduction

Disclaimer

M.-L. Cauwet had a strong contribution to the theoretical part of the work on portfolio [Cauwet et al., 2014] and of the work on noisy optimization in continuous domains [Astest Morales et al.,]; S. Astete-Morales also contributed to this continuous noisy optimization work [Astest Morales et al.,]. B. Rozière contributed to the portfolio work, for the INOPA part [Cauwet et al., 2014]. I am the author of the experimental works in Tables 14.1, 14.2, 14.3, 14.4 and 14.5. S. Astete-Morales contributed to the work on resamplings in continuous optimization [Astete-Morales et al., 2013]. We collaborated for the theory and I am the author of experimental results in Figures 5.2, 5.3, 5.4 and 5.5. J. Decock contributed to the work on grey-box noisy optimization [Decock et al., 2015], Chapter 7. I am the author of results in Tables 7.1 and 7.2. J. Decock is the author of the results in Figures 7.1. I am, with my ph.D. advisor O. Teytaud, the author of the papers used for Chapters 5, 9, 11 and 14. I used some codes from National Dong Hwa University (Hualien), AILAB team, for the work on noisy optimization with differential evolution.

Two strong amateur go players, S.-J. Yen from AILAB team and K.-H. Yeh from National Chiao-Tung University (Hsinchu), have played with our modified version of GnuGo (Figure 11.10).

1 Motivation

1.1 Big news in AI

There are recently quite a lot of big news in AI, including an impact in general audience newspapers:

- Deep learning: how far from human performance (in terms of image recognition)? Deep learning provides great performance in image recognition, outperforming existing algorithms; it also performs surprisingly well for recognizing good moves on a Go board. Recently, a chess engine called *Giraffe*, which uses a neural network for training, was announced to play at approximately International Master Level [Lai, 2015]. However, improvement is always necessary.

- There is also a strong progress in domains in which computers have been widely used for dozens of years, such as numerical optimization. In particular, portfolio methods are essential components for successful combinatorial optimization. On 2007, *SATzilla* [Xu et al., 2008], which uses empirical hardness models to choose among their constituent solvers, got an excellent performance (three gold, one silver and one bronze medal) in the SAT Competition[1]. In this thesis, we will focus on portfolio methods and in particular their application in uncertain settings.

- Another example can be found in games. Monte Carlo Tree Search (MCTS) provided the best algorithm for the game of

[1] `http://www.satcompetition.org`.

Go, outperforming alpha-beta by far, and is now routinely used in many difficult games. MoGo [Lee et al., 2009][2] made the first wins against professionals in 9x9 and with large handicap in 19x19; nowadays, there are strong programs such as Zen and CrazeStone who can win against top players with handicap 5. During the Human *vs.* Computer Go Competition at CIG2015[3], Aya won against a 9P player with handicap 5 in 19x19.

MoGo is empirically close to perfect play in 7x7. Still, in spite of various great successes in Go and in other games, there are unsolved issues in MCTS, and in particular scalability issues, i.e. a plateau in performance when the number of simulations becomes large. We will propose weighted Monte Carlo Tree Search, which outperformed Monte Carlo Tree Search on a family of Tsumego problems.

1.2 Biological intelligence versus computational intelligence

One of the lessons from noisy optimization is that the best convergence rates are obtained when we use sampling all over the domain, and not only close to the approximate optimum; this is the key for the difference between simple regret decreasing as the inverse of the number of evaluations, instead of as the square root of the inverse of the number of evaluations [Fabian, 1967, Astest Morales et al.,]. It is known that nature contains random exploration, for kids mainly; this is a (late) justification for large mutations. We here discuss a different, less well known, inspiration from biology: emotions as a (biological) tool for portfolios.

[2] `https://www.lri.fr/~teytaud/mogo.html`.

[3] 2015 IEEE Conference on Computational Intelligence and Games, `http://cig2015.nctu.edu.tw`.

1.2.1 Emotions and portfolios

Emotion is a key difference between human beings and machines. (The physical differences such as blood, cells are out of my area of interest, yet what need to be mentioned is that Neural networks are already a common tool in AI.) For an agent, the lack of emotion is ambiguous as the emotion can be sometimes regarded as noise for human beings. However, emotion can also trigger a choice between several kinds of behaviors: it is known that, in case of fear, humans have more power in legs (for running away) and arms (for attacking); we are somehow in a portfolio method. Also, for preserving equilibrium, there are several methods available in the brain, and different people prefer different methods (using more or less visual stimuli). When several methods are available, they are termed "vicariant processes", and animals/humans can choose between several of them [Reuchlin, 1978].

1.2.2 Is the emotion a harmful noise ?

The emotion is often unneglected but difficult to be modelized. Besides, it cannot be arbitrarily considered as a harmful noise. For instance, little white lies may be far more innocuous in real life; AI may make optimal choice but not reasonable or friendly choice. The real "intelligence" can not be achieved by an AI without emotion. AI may learn the emotion by words, tone analyze, facial recognition and brainwave detection. Recently, IBM has proposed a service called "IBM Watson Tone Analyzer" to detect emotional tones[4]. Regardless of its performance, there is still unsolved question: can AI simulate emotion and make decisions with emotion? On 2009, Henry Markram, director of the "Blue Brain Project", has claimed that a functional artificial human brain can be built within the next 10 years. However, the reality is not very optimistic. *The idea is plump, however, the reality is scrawny*, says a Chinese proverb.

[4] `https://developer.ibm.com/watson/blog/2015/07/16/ibm-watson-tone-analyzer-service-experimental-release-announcement/`.

1.2.3 Portfolios in practice

There are a large number of state-of-the-art black-box continuous noisy optimization algorithms, which have their own specialization and are dominant in some different particular frameworks. In real life, however, the situation is more complicated and unstable. The choice of algorithms is thus not trivial. It makes sense to combine the existing algorithms rather than defining new algorithms. A classical application is Combinatorial Optimization in discrete domains.

We propose in this manuscript two novel applications for portfolio methods:

- Noisy Optimization in continuous domains, where the comparison of two similar solutions is more expensive than their determinations. This difference cannot be neglected. This is our main motivation of working on algorithm portfolios for noisy optimization, which chooses **online** between several algorithms.

- Games, using the concept of Nash Equilibrium, where non mono-selection is made, but some combinations of strategies are recommended.

These two applications have been performed on both games and electrical systems.

By creating multiple strategies based on the perturbation and combination of random seeds, our method outperforms a standard MCTS implementation in Go, for large numbers of simulations, without computational overhead.

1.3 Application fields in this thesis

The main application fields of this thesis are games and electrical systems. In electrical systems, the transition of energy is a main issue due to some stochastic effects such as faults, climate changes, oil resources, political and geopolitical problems, technology developments and accidents, such as post-Fukushima. The stochasticity

also increases as renewable energy increases. In the traditional optimization of energy systems, disappointingly, the noise is usually badly treated by deterministic management. This leads us to apply noisy optimization to energy systems. The robustness is much more important than the precision in energy problems, therefore, we prefer to spend more evaluations (computational time) to make a better choice.

Another interesting question, emerging by comparing power systems and games, is as follows: shall we propose a probability distribution of strategies to some electricity companies in order to make a choice according to the distribution? A key point is that some decision criteria naturally lead to stochastic decisions (mixed Nash equilibria), as in games. That's a gambling for high stakes. This inspires us to work on some robust decision making approaches.

2 State of the art

Artificial intelligence is the art of automatic decision making. Decisions can be made in various settings:

- We have a function f (termed objective function) which quantifies the quality of a decision; then, we look for a decision x such that $f(x)$ is optimal. This is termed optimization.
- We have such a function f, but its results are noisy; we wish to find an x such that $f(x)$ is optimal on average. This is termed noisy optimization.
- The function f can only be applied to a remote state. The decision to be made has an impact later; i.e., the reward is delayed, and possibly depends on other later decisions, from us, but also, possibly, from other persons (termed agents). This is termed control or reinforcement learning when there is no other agent than us, and games when several agents are concerned.

In this manuscript, we will work on noisy optimization and games. The central tool is the use of portfolio methods; rather than defining new methods, we combine existing ones. As portfolio methods are already a well established domain in noise-free optimization, we focus on cases with uncertainties, such as noisy optimization, and adversarial problems, such as games.

2.1 Noisy Optimization

The term Noisy Optimization refers to the search for the optimum of a given stochastic objective function $f : (x, w) \mapsto f(x, w)$ where x is in the search domain $\mathcal{D} \in \mathbb{R}^d$ and w is some random process. From now on, we assume the existence of some x^* such that $\mathbb{E}_w f(x^*, w)$ is minimum. Many results regarding the performance of algorithms at solving these problems and at the complexity of the problems themselves have been developed in the past, always trying to broaden the extent of them. In this manuscript we propose an optimization algorithm that allows us to generalize results in the literature as well as providing proofs for conjectured results.

We start by stating shortly the framework and definitions of the concepts we will review. Then we comment the state of the art related to stochastic optimization, which will bring us to the specific motivation of this manuscript. We finish this section with an outline of the reminder of the work.

2.1.1 Framework

When the gradient is available for the optimization process, there are algorithms developed in the literature that show a good performance: moderate numbers of function evaluations and good precision. Such is the case for Stochastic Gradient Descent, which is the stochastic version of the classic Gradient Descent Algorithm.

Nonetheless, having access to a gradient is a major assumption in real life scenario. Therefore, in this manuscript, we focus on a black-box case, i.e. we do not use any internal property of f, we only have access to function evaluations for points in the search space - and, in some cases, a slightly grey box scenario (using random seeds).

A noisy black-box optimization algorithm at iteration $m \geq 1$: (i) chooses a new point x_m in the domain and computes its objective function value $y_m = f(x_m, w_m)$, where the w_m are independent copies of w; (ii) computes an approximation $\tilde{x}_m$ of the unknown optimum x^*.

Therefore, at the end of the application of the noisy optimization algorithm, we obtain several sequences: the search points $(x_m)_{m\geq 1}$, the function value on the search points $(y_m)_{m\geq 1}$ and the approximations $(\tilde{x}_m)_{m\geq 1}$. Let us note that each search point x_m is a computable function of the previous search point and their respective function values. But the computation of the search point involves random processes: the stochasticity of the function, and/or some specific random process of the algorithm, if the latter is randomized. The point $\tilde{x}_m$ is termed *recommendation*, and it represents the current approximation of x^*, chosen by the algorithm. Even though in many cases, the recommendation and the search points are exactly the same, we will make a difference here because in the noisy case it is known that algorithms which do not

distinguish recommendations and search points can lead to poor results[1], depending on the noise level.

It is necessary to state a merely technical comment, related to the indexation of the sequences mentioned above and the exact notation used in this manuscript. Depending on the study that one is carrying, there are arguments for indexing the sequences by *iterations* or by *function evaluations*. It occurs often in the case of optimization of noisy functions, that it is more convenient to have multiple evaluations per iteration. Therefore, the best idea is to keep the iteration index, rather than indexing the sequences by the number of evaluations. We will then use $x_{m,1}, \ldots, x_{m,r_m}$ to denote the r_m *search points*[2] *at iteration* m. On the other hand, x_m^{opt}, with only one subscript, is the *recommended point at iteration* m.

For consistency in criteria in the following sections, $\tilde{x}_n$ will always denote the recommendation *after* n *evaluations*. Hence, when the approximations of the optimum are defined per iteration rather than per evaluation, the sequence of recommended points is redefined as follows, for all $n \geq 1$: $\tilde{x}_n = x_k^{opt}$, where k is maximal such that $\sum_{i=1}^{k-1} r_i \leq n$.

Now that we have defined the basic notations for the algorithms considered in this work, let us introduce the *optimization criteria* which will evaluate the *performance* of the algorithms. They allow us to compare the performance of the algorithm considering all search points or only the recommended points. The information we have on the *cost* of evaluating search points can be the dealbreaker when choosing what algorithm to use, as long as we have specialized optimization criteria to help us decide. We will consider three criteria: Uniform Rate (UR), Simple Regret (SR) and Cumulative Regret (CR), respectively defined in Equations 2.1, 2.2 and 2.3.

[1] See [Fabian, 1967, Coulom, 2012] for more on this.

[2] When we need to access to the m^{th} *evaluated search point*, we define x'_m the m^{th} evaluated search point, i.e. $x'_m = x_{i,k}$ with $m = \sum_{j=1}^{i-1} r_j + k$ and $k \leq r_i$.

$$s(UR) = \limsup_i \frac{\log(UR_i)}{\log(i)} \tag{2.1}$$

$$s(SR) = \limsup_i \frac{\log(SR_i)}{\log(i)} \tag{2.2}$$

$$s(CR) = \limsup_i \frac{\log(CR_i)}{\log(i)} \tag{2.3}$$

where UR_i is the $1-\delta$ quantile of $\|x'_i - x^*\|$, SR_i is the $1-\delta$ quantile of $\mathbb{E}_w f(\tilde{x}_i, w) - \mathbb{E}_w f(x^*, w)$, CR_i is the $1-\delta$ quantile of $\sum_{j \leq i} \left(\mathbb{E}_w f(x'_j, w) - \mathbb{E}_w f(x^*, w)\right)$. $\|.\|$ stands for the Euclidean norm, x'_i denotes the i^{th} evaluated search point and $\tilde{x}_i$ denotes the recommendation after i evaluations. We have expectation operators $\mathbb{E}_w$ above with respect to w only, therefore $\mathbb{E}_w f(\tilde{x}_i, w)$ is not deterministic. Quantiles $Q_{1-\delta}$ are with respect to all remaining stochastic parts such as noise in earlier fitness evaluations and possibly internal randomness of the optimization algorithm.

In Equations 2.1, 2.2 and 2.3, we consider the *slopes* in log-log graphs (x-axis: log of evaluation numbers; y-axis: log of UR or SR or CR). The use of the *slopes* turns out to be more convenient because it allow us to know with one single number how fast an algorithm is reaching the specific optimization criterion.

These quantities depend on the threshold δ, but in all cases below we get the same result independently of δ, therefore we will drop this dependency.

Note that, for $s(UR)$ and $s(SR)$, 0 can be trivially reached by an algorithm with constant $(x'_m, \tilde{x}_m)$. Therefore, $s(UR)$ and $s(SR)$ are only interesting when they are less than 0. And $s(CR)$ is relevant when it is less than 1.

Finally, regarding to the objective functions, we investigate three types of noise models :

$$Var(f(x,w)) = O\left([\mathbb{E}_w f(x,w) - \mathbb{E}_w f(x^*,w)]^z\right) \qquad z \in \{0,1,2\} \tag{2.4}$$

We will refer to them respectively as the case where the variance of the noise is *constant*, *linear* and *quadratic* as a function of the simple regret.

2.1.2 Local noisy optimization

Local noisy optimization refers to the optimization of an objective function in which the main problem is noise, and not local minima. Hence, diversity mechanisms as in [Jones et al., 1998] or [Auger et al., 2005], in spite of their qualities, are not relevant in this manuscript. We also restrict our work to noisy settings in which noise may not decrease to 0 around the optimum. This constrain makes our work different from [Jebalia et al., 2010]. In [Arnold and Beyer, 2006, Finck et al., 2011] we can find noise models related to ours but the results presented here are not covered by their analysis. On the other hand, in [Coulom, 2012, Coulom et al., 2011, Teytaud and Decock, 2013], different noise models (with Bernoulli fitness values) are considered, inclosing a noise with variance which does not decrease to 0 (as in the present paper). They provide general lower bounds, or convergence rates for specific algorithms, whereas we consider convergence rates for classical evolution strategies equipped with resamplings.

We classify noisy local convergence algorithms in the following 3 families:

- *Algorithms based on sampling, as far as they can, close to the optimum.* In this category, we include evolution strategies [Beyer, 2001, Finck et al., 2011, Arnold and Beyer, 2006] and EDA [Yang et al., 2005] as well as pattern search methods designed for noisy cases [Anderson and Ferris, 2001, Lucidi and Sciandrone, 2002, Kim and Zhang, 2010]. Typically, these algorithms are based on noise-free algorithms, and evaluate individuals multiple times in order to cancel (reduce) the effect of noise. Authors studying such algorithms focus on the number of resamplings; it can be chosen by estimating the noise level [Hansen et al., 2009], or using the step-size, or, as in parts of the present work, in a non-adaptive manner.

- *Algorithms which learn (model) the objective function*, sample at locations in which the model is not precise enough, and then assume that the optimum is nearly the optimum of the learnt model. Surrogate models and Gaussian pro-

cesses [Jones et al., 1998, Villemonteix et al., 2008] belong to this family. However, Gaussian processes are usually supposed to achieve global convergence (i.e. good properties on multimodal functions) rather than local convergence (i.e. good properties on unimodal functions) - in the present document, we focus on local convergence.

- *Algorithms which combine both ideas*, assuming that learning the objective function is a good idea for handling noise issues but considering that points too far from the optimum cannot be that useful for an optimization. This assumption makes sense at least in a scenario in which the objective function cannot be that easy to learn on the whole search domain. CLOP [Coulom, 2012, Coulom et al., 2011] is such an approach.

2.1.3 Black-box noisy optimization

When solving problems in real life, having access to noisy evaluations of the function to be optimized, instead of the real evaluations, can be a very common issue. If, in this context, we have access to the gradient of the function, the Stochastic Gradient Method is particularly appreciated for the optimization, given its efficiency and its moderate computational cost [Bottou and Bousquet, 2011]. However, the most general case consists in only having access to the function evaluations in certain points: this is called a *black-box setting*. This setting is specially relevant in cases such as reinforcement learning, where gradients are difficult and expensive to get [Sehnke et al., 2010]. For example, Direct Policy Search, an important tool from reinforcement learning, usually boils down to choosing a good representation [Bengio, 1997] and applying black-box noisy optimization [Heidrich-Meisner and Igel, 2009].

Among the noisy optimization methods, we find in the work of [Robbins and Monro, 1951], the ground-break proposal to face the problem of having noise in one or more stages of the optimization process. From this method derive other important methods as the type of stochastic gradient algorithms. On a similar track, [Kiefer and Wolfowitz, 1952] has also added tools based on finite difference

inside of this kind of algorithms. In a more general way, [Spall, 2000, Spall, 2003, Spall, 2009] designed various algorithms which can be adapted to several settings, with or without noise, with or without gradient, with a moderate number of evaluations per iteration.

The tools based on finite differences are classical for approximating derivatives of functions in the noise-free case. Nonetheless, the use of finite differences is usually expensive. Therefore, for instance, quasi-Newton methods also use successive values of the gradients for estimating the Hessian [Broyden, 1970, Fletcher, 1970, Goldfarb, 1970, Shanno, 1970]. And this technique has also been applied in cases in which the gradient itself is unknown, but approximated using successive objective function values [Powell, 2004, Powell, 2008]. With regards to latter method, so-called NEWUOA algorithm, it presents impressive results in the black-box noise-free case but this results do not translate into the noisy case, as reported by [Ros, 2009].

2.1.4 Optimization criteria

Studies on uniform, simple and cumulative regret. In this manuscript we refer to three optimization criteria to study the convergence of algorithms, so-called *uniform, simple and cumulative regret*, by taking into account the *slope* on the *log-log* graph of the criteria vs the number of evaluations (see $s(UR)$, $s(SR)$ and $s(CR)$ defined in Equations 2.1, 2.2 and 2.3). The literature in terms of these criteria is essentially based on stochastic gradient techniques.

[Sakrison, 1964, Dupač, 1957] have shown that $s(SR) = -\frac{2}{3}$ can be reached, when the objective function is twice differentiable in the neighborhood of the optimum. This original statement has been broadened and specified. [Spall, 2000] obtained similar results with an algorithm using a small number of evaluations per iteration, and an explicit limit distribution. The small number of evaluations per iteration makes stochastic gradient way more practical than earlier algorithms such as [Dupač, 1957, Fabian, 1967], in particular in high dimension where these earlier algorithms were

based on huge numbers of evaluations per iterations[3]. In addition, their results can be adapted to noise-free settings and provide non trivial rates in such a setting.

[Fabian, 1967] made a pioneering work with a Simple Regret arbitrarily close to $O(1/n)$ after n evaluations, i.e. $s(SR) \simeq -1$, provably tight as shown by [Chen, 1988], when higher derivatives exist. Though they use a different terminology than recent papers in the machine learning literature, [Fabian, 1967] and [Chen et al., 1996] have shown that stochastic gradient algorithms with finite differences can reach $s(UR) = -\frac{1}{4}$, $s(SR) = -1$ and $s(CR) = \frac{1}{2}$ on quadratic objective functions with additive noise; the slopes $s(SR) = -1$ and $s(CR) = \frac{1}{2}$ are optimal in the general case as shown by, respectively, [Chen, 1988] (simple regret) and [Shamir, 2013] (cumulative regret). [Shamir, 2013] also extended the analysis in terms of dependency in the dimension and non-asymptotic results - switching to $-\frac{1}{2}$ for twice differentiable functions, in the non-asymptotic setting, as opposed to $-\frac{2}{3}$ for [Dupač, 1957] in the asymptotic setting.

[Rolet and Teytaud, 2009, Rolet and Teytaud, 2010, Coulom et al., 2011] take into consideration functions with $\mathbb{E}_w f(x,w) - \mathbb{E}_w f(x^*,w) = C\|x - x^*\|^p$ ($p \geq 1$) and different intensity on the perturbation; one with noise variance $\Theta(1)$ and the second with variance $Var(f(x,w)) = O(\mathbb{E}_w f(x,w) - \mathbb{E}_w f(x^*,w))$. In the case of "strong" noise (i.e. variance $\Theta(1)$), they prove that the optimal $s(UR)$ is in $[-\frac{1}{p}, -\frac{1}{2p}]$. In the case of $z = 1$ as well, a lower bound $-\frac{1}{p}$ was obtained in [Decock and Teytaud, 2013] for algorithms matching some "locality assumption". While having weaker perturbations of the function (variance of noise decreasing linearly as a function of the simple regret, $z = 1$), intuitively as expected, they obtain better results, with $s(UR) = -\frac{1}{p}$, proved tight in [Decock and Teytaud, 2013] for algorithms with "locality assumption".

The study of noise variance decreasing quickly enough, for some simple functions, has been performed in [Jebalia et al., 2010],

[3]It must be pointed out that in the algorithms proposed in [Dupač, 1957, Fabian, 1967], the number of evaluations per iteration is constant, so that the rate $O(1/n)$ is not modified by this number of evaluations. Still, the number of evaluations is exponential in the dimension, making the algorithm intractable in practice.

where it is conjectured that one can obtain $s(UR) = -\infty$ and geometric convergence ($\|x_n\| = O(\exp(-\Omega(n)))$) - we prove this $s(UR) = -\infty$ for our algorithm.

Another branch of the state of the art involves Bernoulli random variables as objective functions: for a given x, $f(x, w)$ is a Bernoulli random variable with probability of success $\mathbb{E}_w f(x, w)$. We wish to find x such that $\mathbb{E}_w f(x, w)$ is minimum. This framework is particularly relevant in games [Chaslot et al., 2008,Coulom, 2012] or viability applications [Aubin, 1991, Chapel and Deffuant, 2006] and it is a natural framework for $z = 1$ (when the optimum value $\mathbb{E}_w f(x^*, w)$ is 0, in the Bernoulli setting, the variance at x is linear as a function of the simple regret $\mathbb{E}_w f(x, w) - \mathbb{E}_w f(x^*, w)$) or for $z = 0$ (i.e. when the optimum value is > 0). In these theoretical works, the objective function at x is usually a Bernoulli random variable with parameter depending on $a + b\|x - x^*\|^\zeta$ for some $\zeta \geq 1, a \geq 0, b > 0$. Some of the lower bounds below hold even when considering only Bernoulli random variables, while upper bounds usually hold more generally.

Notice that by definition the $s(UR)$ criterion is harder to be reached than $s(SR)$ because all search points must verify the bound, not only the recommended ones - for any problem, if for some algorithm, $s(UR) \leq c$, then for the same problem there is an algorithm such that $s(SR) \leq c$. There are also relations between $s(CR)$ and $s(UR)$, at least for algorithms with a somehow "smooth" behavior; we will give more details on this in our conclusions. In this manuscript, we propose a Hessian-based algorithm which provably covers all the rates above, including UR, SR and CR and $z = 0, 1, 2$. Our results are summarized in Table 4.1.

In the case of noise with constant variance, the best performing algorithms differ, depending on the optimization criteria (UR, SR, CR) that we choose. On the other hand, when variance decreases at least linearly, the algorithms used for reaching optimal $s(UR)$, $s(SR)$ and $s(CR)$ are the same. In all frameworks, we are not aware of differences between algorithms specialized on optimizing $s(UR)$ criterion and on $s(CR)$ criterion. An interesting remark on the differences between the criteria is the following: optimality for $s(SR)$ and $s(CR)$ can not be reached simultaneously. This so-called *tradeoff* is observed in discrete settings [Stoltz et al., 2011]

and also in the algorithm presented in [Fabian, 1967], to which we will refer as Fabian's algorithm. Fabian's algorithm is a gradient descent algorithm, using finite differences for approximating gradients. Depending on the value of a parameter, namely γ_{Fabian} [4], we get a good $s(SR)$ or a good $s(CR)$, but never both simultaneously. In the case of quadratic functions with additive noise (constant variance $z = 0$):

- $\gamma_{Fabian} \to \frac{1}{4}$ leads to $s(SR) = -\frac{1}{2}$ and $s(CR) = \frac{1}{2}$;
- $\gamma_{Fabian} \to 0$ leads to $s(SR) = -1$ and $s(CR) = 1$.

The algorithms analyzed in this manuscript, as well as Fabian's algorithm or the algorithm described in [Shamir, 2013], present this tradeoff, and similar rates. A difference between the latter algorithm and ours is that ours have faster rates when the noise decreases around the optimum and proofs are included for other criteria (see Table 4.1). The cases with variance decreasing towards zero in the vicinity of the optimum are important, for example in the Direct Policy Search method when the fitness function is 0 for a success and 1 for a failure; if a failure-free policy is possible, then the variance is null at the optimum. Another example is parametric classification, when there exists a classifier with null error rate: in such a case, variance is null at the optimum, and this makes convergence much faster ([Vapnik, 1995]).

Importantly, some of the rates discussed above are for stronger convergence criteria than ours. For example, [Fabian, 1967] gets almost sure convergence. [Shamir, 2013] gets convergence in expectation. [Spall, 2000] gets asymptotic distributions. We get upper bounds on quantiles of various regrets, up to constant factors.

2.1.5 Motivation and key ideas

This section discusses the motivations for Part II. First, to obtain new bounds and recover existing bounds within a single algorithm (Section 2.1.5.1). Second, proving results in a general approximation setting, beyond the classical approximation by quadratic

[4] [Fabian, 1967] defines a sequence $c_n = cn^{-\gamma_{Fabian}}$, which specifies the sequence of finite difference widths used for the gradient approximation.

models; this is in line with the advent of many new surrogate models in the recent years (Section 2.1.5.2).

2.1.5.1 Generalizing existing bounds for noisy optimization

We here extend the state of the art in the case of $z = 2$ for all criteria, and $z = 1$ for more general families of functions (published results were only for sphere functions), and get all the results with a same algorithm. We also generalize existing results for UR or SR or CR to all three criteria. On the other hand, we do not get Fabian's $s(SR)$ arbitrarily close to -1 on smooth non-quadratic functions with enough derivatives, which require a different schema for finite differences and assumes the existence of a large number of additional derivatives.

2.1.5.2 Hessian-based noisy optimization algorithms and beyond

We propose and study a noisy optimization algorithm, which possibly uses a Newton-style approximation, i.e. a local quadratic model. Gradient-based methods (without Hessian) have a difficult parameter, which is the rate at which gradient steps are applied. Such a problem is solved when we have a Hessian; the gradient and Hessian provide a quadratic approximation of the objective function, and we can use, as next iterate, the minimum of this quadratic approximation. There are for sure new parameters, associated to the Hessian updates, such as the widths used in finite differences; however other algorithms, without Hessians, already have such parameters (e.g. [Fabian, 1967, Shamir, 2013]). Such a model was already proposed in [Fabian, 1971], a few years after his work establishing the best rates in noisy optimization [Fabian, 1967], but without any proof of improvement. [Spall, 2009] also proposed an algorithm based on approximations of the gradient and Hessian, when using the SPSA (simultaneous perturbation stochastic approximation) method [Spall, 2000]. They provided some results on the approximated Hessian and on the convergence rate of the algorithm; [Spall, 2000] and [Spall, 2009] study the convergence rate in the search space, but their results can be converted in simple regret results and in this setting they get the same

slope of simple regret $-\frac{2}{3}$ as [Dupač, 1957]; the paper also provides additional information such as the limit distribution and the dependency in the eigenvalues. The algorithm in [Spall, 2000, Spall, 2009] work with a very limited number of evaluations per iteration, which is quite convenient in practice compared to numbers of evaluations per iteration exponential in the dimension in [Shamir, 2013].

Importantly, our algorithm is not limited to optimization with black-box approximations of gradients and Hessians; we consider more generally algorithms with *low-squared error* (LSE) (Definition 2).

2.1.6 Resampling methods

2.1.6.1 Noisy optimization with variance reduction

In standard noisy optimization frameworks, the black-box noisy optimization algorithm, for its n^{th} request to the black-box objective function, can only provide some x in a d-dimentional search domain, and receive a realization of $f(x, w_n)$. The w_n, $n \in \{1, 2, \dots\}$, are independent samples of w, a random variable with values in $D \subset \mathbb{R}$. The algorithm can not influence the w_n. Contrarily to this standard setting, we here assume that the algorithm can request $f(x, w_n)$ where w_n is:

- either an independent copy of w (independent of all previously used values), which inspires our work on adaptive and non-adaptive resampling rules;
- or a previously used value w_m for some $m < n$ (m is chosen by the optimization algorithm).

Due to the later possibility, *paired sampling* can be applied, i.e. the same w_n can be used several times, as explained in Chapter 7. In addition, we assume that we have *strata*. A stratum is a subset of D. Strata have empty intersections and their union is D (i.e. they realize a partition of D). When an independent copy of w is requested, the algorithm can decide to provide it conditionally

to a chosen stratum. Thanks to strata, we can apply *stratified sampling* (Section 2.1.6.2).

2.1.6.2 Statistics of variance reduction

Monte Carlo methods are the estimation of the expected value of a random variable owing to a randomly drawn sample. Typically, in our context, $\mathbb{E}[f(x,w)]$ can be estimated as a result of $f(x,w_1), f(x,w_2), \ldots, f(x,w_n)$, where the w_i are independent copies of w, $i \in \{1,\ldots,n\}$. Laws of large numbers prove, under various assumptions, the convergence of Monte Carlo estimates such as (see [Billingsley, 1986])

$$\hat{\mathbb{E}}f(x,w) = \frac{1}{n}\sum_{i=1}^{n} f(x,w_i) \to \mathbb{E}_w f(x,w). \tag{2.5}$$

There are also classical techniques for improving the convergence:

- *Antithetic variates* (symmetries): ensure some regularity of the sampling by using symmetries. For example, if the random variable w has distribution invariant by symmetry w.r.t 0, then, instead of Equation 2.5, we use Equation 2.6, which reduces the variance:

$$\hat{\mathbb{E}}f(x,w)=\frac{1}{n}\sum_{i=1}^{n/2}\left(f(x,w_i)+f(x,-w_i)\right). \tag{2.6}$$

  More sophisticated antithetic variables are possible (combining several symmetries).

- *Importance sampling*: instead of sampling w with density dP, we sample w' with density dP'. We choose w' such that the density dP' of w' is higher in parts of the domain which are critical for the estimation. However, this change of distribution introduces a bias. Therefore, when computing the average, we change the weights of individuals by the ratio of probability densities as shown in Equation 2.7 - which is an unbiased estimate.

$$\hat{\mathbb{E}}f(x,w)=\frac{1}{n}\sum_{i=1}^{n}\frac{dP(w_i)}{dP'(w_i)}f(x,w_i) \tag{2.7}$$

- *Quasi Monte Carlo* methods: use samples aimed at being as uniform as possible over the domain. Quasi Monte Carlo methods are widely used in integration; thanks to modern randomized Quasi Monte Carlo methods, they are usually at least as efficient as Monte Carlo and much better in favorable situations [Niederreiter, 1992, Cranley and Patterson, 1976, Mascagni and Chi, 2004, Wang and Hickernell, 2000]. There are interesting (but difficult and rather "white-box") tricks for making them applicable for time-dependent random processes with many time steps [Morokoff, 1998].

- [Dupacová et al., 2000] proposes to generate a finite sample which approximates a random process, optimally for some metric. This method has advantages when applied in the framework of Bellman algorithms as it can provide a tree representation, mitigating the anticipativity issue. But it is hardly applicable when aiming at the convergence to the solution for the underlying random process.

- *Control variates*: instead of estimating $\mathbb{E}f(x,w)$, we estimate $\mathbb{E}\left(f(x,w)-g(x,w)\right)$, using

$$\mathbb{E}f(x,w) = \underbrace{\mathbb{E}g(x,w)}_{A} + \mathbb{E}\underbrace{\left(f(x,w)-g(x,w)\right)}_{B}.$$

  This makes sense if g is a reasonable approximation of f (so that term B has a small variance) and term A can be computed quickly (e.g. if computing g is much faster than computing f or A can be computed analytically).

- *Stratified sampling* is the case in which each w_i is randomly drawn conditionally to a stratum. We consider that the domain of w is partitioned into disjoint strata $S_1,\ldots,S_N$. N is the number of strata. The stratification function $i \mapsto s(i)$ is chosen by the algorithm and w_i is randomly drawn conditionally to $w_i \in S_{s(i)}$.

$$\hat{\mathbb{E}}f(x,w) = \sum_{i=1}^{n} \frac{P(w \in S_{s(i)}) f(x,w_i)}{\text{Cardinality}\{j \in \{1,\ldots,n\}; w_j \in S_{s(i)}\}} \quad (2.8)$$

- *Common random numbers (CRN)*, or paired comparison, refer to the case where we want to know $\mathbb{E}f(x, w)$ for several x, and use the same samples $w_1, \ldots, w_n$ for the different possible values of x.

In Chapter 7, we focus on stratified sampling and paired sampling, in the context of optimization with arbitrary random processes. They are easy to adapt to such a context, which is not true for other methods cited above.

Stratified sampling Stratified sampling involves building strata and sampling in these strata. **Simultaneously building strata and sampling** There are some works doing both simultaneously, i.e. build strata adaptively depending on current samples. For example, [Lavallée and Hidiroglou, 1988,Sethi, 1963] present an iterative algorithm which stratifies a highly skewed population into a take-all stratum and a number of take-some strata. [Kozak, 2004] improves their algorithm by taking into account the gap between the variable used for stratifying and the random value to be integrated.

A priori stratification However, frequently, strata are built in an ad hoc manner depending on the application at hand. For example, an auxiliary variable $\tilde{f}(x^*, w)$ might approximate $w \mapsto f(x^*, w)$, and then strata can be defined as a partition of the $\tilde{f}(x^*, w)$. It is also convenient for visualization, as in many cases the user is interested in viewing statistics for w leading to extreme values of $f(x^*, w)$. More generally, two criteria dictate the choice of strata:

- a small variance inside each stratum, i.e. $\mathcal{V}ar_{w|S} f(x^*, w)$ small for each stratum S, is a good idea;
- interpretable strata for visualization purpose.

The sampling can be

- *proportional*, i.e. the number of samples in each stratum S is proportional to the probability $P(w \in S)$;

- or *optimal*, i.e. the number of samples in each stratum S is proportional to a product of $P(w \in S)$ and an approximation of the standard deviation $\sqrt{\mathcal{V}ar_{w|S} f(x^*, w)}$. In this case, reweighting is necessary, as in Equation 2.8.

Stratified noisy optimization Compared to classical stratified Monte Carlo, an additional difficulty when working in stratified noisy optimization is that x^* is unknown, so we can not easily sample $f(x^*, w)$. Also, the strata should be used for many different x; if some of them are very different, nothing guarantees that the variance $Var_{w|S} f(x, w)$ is approximately the same for each x and for x^*. As a consequence, there are few works using stratification for noisy optimization and there is, to the best of our knowledge, no work using optimal sampling for noisy optimization, although there are many works around optimal sampling. We will here focus on the simple proportional case. In some papers [Linderoth et al., 2006], the word "stratified" is used for *Latin Hypercube Sampling*; we do not use it in that sense in the present paper.

Common random numbers & paired sampling Common Random Numbers (CRN), also called correlated sampling or pairing, is a simple but powerful technique for variance reduction in noisy optimization problems. Consider $x_1, x_2 \in \mathbb{R}^d$, where d is the dimension of the search domain and w_i denotes the i^{th} independent copy of w:

$$\begin{aligned} & \mathcal{V}ar \sum_{i=1}^{n} (f(x_1, w_i) - f(x_2, w'_i)) \\ = \; & n\mathcal{V}ar \left(f(x_1, w_1) - f(x_2, w'_1) \right) \\ = \; & n\mathcal{V}ar f(x_1, w_1) + n\mathcal{V}ar f(x_2, w'_1) \\ & -2n\mathcal{C}ov \left(f(x_1, w_1), f(x_2, w'_1) \right). \end{aligned}$$

If $\mathcal{C}ov(f(x_1, w_i), f(x_2, w'_i)) > 0$, i.e. there is a positive correlation between $f(x_1, w_i)$ and $f(x_2, w'_i)$, the estimation errors are smaller. CRN is based on $w_i = w'_i$, which is usually a simple and efficient solution for correlating $f(x_1, w_i)$ and $f(x_2, w'_i)$; there are examples in which, however, this does not lead to a positive correlation. In Chapter 7, we will present examples in which CRN does not work.

Pairing in artificial intelligence Pairing is used in different application domains related to optimization. In games, it is a common practice to compare algorithms based on their behaviors on a finite constant set of examples [Huang et al., 2010]. The cost of labelling (i.e. the cost for finding the ground truth regarding the value of a game position) is a classical reason for this. This is different from simulating against paired random realizations (because it is usually an adversarial context rather than a stochastic one), though it is also a form of pairing and is related to our framework of dynamic optimization. More generally, paired statistical tests improve the performance of stochastic optimization methods, e.g. dynamic *Random Search* [Hamzaçebi and Kutay, 2007, Zabinsky, 2009] and *Differential Evolution* [Storn and Price, 1997]. It has been proposed [Takagi and Pallez, 2009] to use a paired comparison-based *Interactive Differential Evolution* method with faster rates. In *Direct Policy Search*, paired noisy optimization has been proposed in [Strens and Moore, 2001, Strens et al., 2002, Kleinman et al., 1999]. Our work follows such approaches and combines them with stratified sampling. This is developed in Chapter 7. In *Stochastic Dynamic Programming* (SDP) [Bellman, 1957] and its dual counterpart Dual SDP [Pereira and Pinto, 1991], the classical *Sample Average Approximation* (SAA) reduces the problem to a finite set of scenarios; the same set of random seeds is used for all the optimization run. It is indeed often difficult to do better, because there are sometimes not infinitely many scenarios available. Variants of dual SDP have also been tested with increasing set of random realizations [de Matos et al., 2012] or one (new, independent) random realization per iteration [Shapiro et al., 2013]. A key point in SDP is that one must take care of anticipativity constraints, which are usually tackled by a special structure of the random process. This is beyond the scope of the present chapter; we focus on direct policy search, in which this issue is far less relevant as long as we can sample infinitely many scenarios. However, our results on the compared benefits of stratified sampling and common random numbers suggest similar tests in non direct approaches using Bellman values.

2.2 Portfolio Algorithms

2.2.1 Algorithm selection

Combinatorial optimization is probably the most classical application domain for AS [Kotthoff, 2012]. However, machine learning is also a classical test case [Utgoff, 1988]; in this case, AS is sometimes referred to as meta-learning [Aha, 1992].

2.2.1.1 No free lunch.

[Wolpert and Macready, 1997] 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 AS can outperform existing algorithms on average on this uniform probability distribution of problems. Nonetheless, reality is very different from a uniform average of optimization problems, and AS does improve performance in many cases.

2.2.1.2 Chaining and information sharing.

Algorithm chaining [Borrett and Tsang, 1996] means switching from one solver to another during the AS run. More generally, a *hybrid* algorithm is a combination of existing algorithms by any means [Vassilevska et al., 2006]. This is an extreme case of sharing. Sharing consists in sending information from some solvers to other solvers; they communicate in order to improve the overall performance.

2.2.1.3 Static portfolios & parameter tuning.

A portfolio of solvers 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 [Xu et al., 2008]. Samulowitz and Memisevic have pointed out in [Samulowitz and Memisevic, 2007] 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. AS and parameter tuning are combined in [Xu et al., 2011]; parameter tuning can be

viewed as an AS over a large but structured space of solvers. We refer to [Kotthoff, 2012] and references therein for more information on parameter tuning and its relation to AS; this is beyond the scope of the present chapter.

2.2.1.4 Fair or unfair distribution of computation budgets.

In [Pulina and Tacchella, 2009], 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 keeping 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; [Kadioglu et al., 2011], 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. In [Gagliolo and Schmidhuber, 2005, Gagliolo and Schmidhuber, 2006], it is proposed to use 50% of the time budget for the best solver, 25% for the second best, and so on. Some AS algorithms [Gagliolo and Schmidhuber, 2006, Armstrong et al., 2006] do not need a separate training phase, and perform 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. A portfolio algorithm, namely Noisy Optimization Portfolio Algorithm (NOPA), designed for noisy optimization solvers, and which distributes uniformly the computational power among them, is proposed in [Cauwet et al., 2014]. We extend it to INOPA (Improved NOPA), which is allowed to distribute the budget in an unfair manner. It is proved that INOPA reaches the same convergence rate as the best solver, within a small (converging to 1) multiplicative factor on the number of evaluations, when there is a unique optimal solver - thanks to a principled distribution of the budget into (i) running all the solvers (ii) comparing their results

(iii) running the best performing one. The approach is anytime, in the sense that the computational budget does not have to be known in advance.

2.2.1.5 Parallelism.

We refer to [Hamadi, 2013] for more on parallel portfolio algorithms (though not in the noisy optimization case). 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[5]).

2.2.1.6 Best solver first.

[Pulina and Tacchella, 2009] point out the need for a good ordering of solvers, even if it has been decided to distribute nearly uniformly the time budget among them: this improves the anytime behavior. For example, they propose, within a given scheduling with same time budget for each optimizer, to use first the best performing solver. We will adapt this idea to our context; this leads to INOPA, improved version of NOPA.

2.2.1.7 Bandit literature.

During the last decade, a wide literature on bandits [Lai and Robbins, 1985, Auer, 2003, Bubeck et al., 2009] has 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 [Kocsis and Szepesvari, 2006b] or adversarial [Grigoriadis and Khachiyan, 1995,Auer et al., 1995]), and maybe due to strong differences such as the very non-stationary nature of bandit problems involved in optimization portfolios, these methods did not, for the moment, really find their way to AS. Another approach consists in writing this bandit algorithm as a

[5]"Resamplings" means that the stochastic objective function, also known as fitness function, is evaluated several times at the same search point. This mitigates the effects of noise.

meta-optimization problem; [St-Pierre and Liu, 2014] applies the differential evolution algorithm [Storn and Price, 1997] to some non-stationary bandit problem, which outperforms the classical bandit algorithm on an AS task. The state of the art of multi-armed bandit will be extended in Section 2.3.

The main contributions of this chapter can be summarized as follows. First, we prove positive results for a portfolio algorithm, termed NOPA, for AS in noisy optimization. Second, we design a new AS, namely INOPA, which (i) gives the priority to the best solvers when distributing the computational power (ii) approximately reaches the same performance as the best solver (iii) possibly shares information between the different solvers. We then prove the requirement of selecting the solver that was apparently the best *some time before* the current iteration - a phenomenon that we term the *lag*. Finally, we provide some experimental results.

2.2.2 Applications of portfolio algorithms

The most classical application is combinatorial, in particular with the success of SATzilla [Xu et al., 2008]. [Baudiš and Pošík, 2014] worked on online black-box algorithm portfolios for continuous optimization in the deterministic case. We here extend this work to the noisy case (Chapter 14).

2.3 Multi-Armed Bandit

The multi-armed bandit problem [Auer et al., 2002a, Katehakis and Veinott Jr, 1987] is an important model of exploration/exploitation trade-offs, aimed at optimizing the expected payoff. It is related to portfolio methods, as they are natural candidates for selecting the best algorithm in a family of solvers depending on their payoffs. The concepts defined here will be used throughout the present document: adversarial bandit algorithms for computing Nash equilibria, bandit algorithms for portfolio methods in noisy optimization; and in Monte Carlo Tree Search, used in some of our experiments.

2.3.1 Multi-armed bandit framework and notations

A multi-armed stochastic bandit can be considered as a family of unknown distributions $B = (D_1, D_2, \ldots, D_K)$, where $K \in \mathbb{N}^+$ refers to the number of arms and {1,2,...,K} denotes the set of arms. Let μ_k be the expected reward of each arm $k \in \{1, 2, \ldots, K\}$ and $k^* = \underset{k\in\{1,2,\ldots,K\}}{\operatorname{argmax}} \mu_k$ is the arm with maximal expected reward. The maximal expected reward is thus $\mu^* = \mu_{k^*}$.

2.3.2 Sequential bandit setting

There are T time steps. At each time step $t \in \{1, 2, \ldots, T\}$, the algorithm chooses $\theta_t \in \{1, 2, \ldots, K\}$ and obtains a reward r_t. $\hat{\mu}_{k,t}$ is the average reward when arm $k \in \{1, 2, \ldots, K\}$ was played at time $t' \leq t$:

$$\hat{\mu}_{k,t} = \frac{1}{N_{k,t}} \sum_{t'\leq t, \theta_{t'}=k} r_{t'}, \tag{2.9}$$

where $N_{k,t} = \sum_{t'\leq t, \theta_{t'}=k} 1$. In the case of pure exploration (also termed "simple regret") problems, the algorithm must also propose some $\tilde{\theta}_t \in \{1, 2, \ldots, K\}$. When T is known in advance (non anytime setting), only $\tilde{\theta}_T$ matters. The pseudo code for a generic sequential bandit algorithm is provided in Algorithm 1.

We use simple regret and cumulative regret as optimization criteria. The simple regret is the difference between the average reward of the best arm and the average reward obtained by the recommended arm. The simple regret at time T is thus computed by

$$SR_T = \mu^* - \mu_{\tilde{\theta}_T}. \tag{2.10}$$

The cumulative regret at time T is defined as

$$CR_T = \sum_{t\leq T} (\mu^* - r_t). \tag{2.11}$$

We here consider the expected case: the goal is to minimize the expected cumulative regret $\mathbb{E}(CR_T)$.

Algorithm 1 Generic sequential bandit algorithm. The problem is described through the *get stochastic reward*, a stochastic method and the arm sets. The *return* method is formally called the recommendation policy.

Input: $T > 0$: Computational budget
Input: $\{1, 2, \ldots, K\}$: Set of arms
Input: $\pi_{\tilde{SEL}}$: Exploration policy
1: **for** $t \in \{1, \ldots, T\}$ **do**
2: Select arm $k \in \{1, 2, \ldots, K\}$ based upon $\pi_{\tilde{SEL}}(\cdot)$, *get stochastic reward* r_t
3: Update the information of k with r_t
4: **end for**
Output: $\tilde{\theta}_T$

2.3.3 Algorithms for cumulative regret

We present some well known algorithms for exploration on sequential machines and their upper bounds on the expected cumulative regret.

2.3.3.1 Upper Confidence Bound (UCB)

The Upper Confidence Bound algorithm (**UCB**) goes back to [Lai and Robbins, 1985, Auer et al., 2002a]. On a single processor, for a given parameter $\alpha > 0$:

$$\theta_t = \begin{cases} mod(t-1, K) + 1 & \text{if } t \leq K \\ \underset{k \in \{1,2,\ldots,K\}}{\operatorname{argmax}} \ \hat{\mu}_{k,t-1} + \sqrt{\alpha \frac{\log(t)}{N_{k,t-1}}} & \text{otherwise,} \end{cases} \tag{2.12}$$

where $N_{k,t}$ is the number of times the arm $k \in \{1, 2, \ldots, K\}$ has been chosen at time $t' \leq t$ and $\hat{\mu}_{k,t}$ denotes its' empirical expectation of reward after time t. [Auer et al., 2002a] proves that, when using **UCB**, $\exists C \in \mathbb{R}$ such that

$$\mathbb{E}(CR_t) \leq \sum_{k:\mu_k < \mu^*} \frac{8 \log(t)}{\Delta_k} + C, \tag{2.13}$$

where $\Delta_k = \mu^* - \mu_k$ is the expected loss of playing the arm $k \in \{1, 2, \ldots, K\}$.

2.3.3.2 β-UCB

[Audibert et al., 2006] provides a different algorithm, **β-UCB**, which ensures that with probability at least $1 - \beta$, $\beta \in (0, 1)$, at least a subset of the optimal arms is found after a number of iterations linear (within logarithmic factors) in

$$\log(\frac{K}{\beta}) \sum_{k \in \{1,2,\ldots,K\}; \Delta_k > 0} (\frac{\sigma_k^2}{\Delta_k} + \frac{2}{\Delta_k}), \tag{2.14}$$

where σ_k denotes the variance of reward of the arm $k \in \{1, 2, \ldots, K\}$ and $\Delta_k = \mu^* - \mu_k$ is the expected loss when playing arm k. When the optimal arm is found, then the simple regret becomes zero, so this is the number of iterations necessary for reaching regret zero.

Theorem 3 in [Audibert et al., 2006] provides an upper bound for the cumulative regret of the **β-UCB** policy:

$$\mathbb{E}(CR_t) \leq C' \log(2t) \sum_{k \neq k^*} (1 + \frac{\sigma_k^2}{\Delta_k}), \tag{2.15}$$

with probability at least $1 - \beta$ for any time $t \in N^+$, where $C' \in \mathbb{R}$ is a universal constant.

2.3.3.3 UCB-tuned & UCB-V

[Auer et al., 2002a] proposes a **UCB** variant, termed UCB-tuned (**UCBt**), where the exploration parameter α is adaptive:

$$\theta_t = \begin{cases} mod(t-1, K) + 1 & \text{if } t \leq K \\ \underset{k \in \{1,2,\ldots,K\}}{\text{argmax}} \; \hat{\mu}_{k,t-1} + \sqrt{\alpha_{k,t} \frac{\log(t)}{N_{k,t-1}}} & \text{otherwise.} \end{cases} \tag{2.16}$$

where $\alpha_{k,t} = \min\{\frac{1}{4}, (\frac{1}{N_{k,t-1}} \sum_{s \leq N_{k,t-1}} \mu_{k,s}^2) - \hat{\mu}_{k,t-1}^2 + \sqrt{\frac{2\log(t)}{N_{k,t-1}}}\}$.

UCBt is aimed at exploiting the variance information; however, such a property has never been proved. [Audibert et al., 2009]

proposes a **UCB-V** algorithm as exploration policy for stochastic bandit problems, which is similar to **UCBt**, and proves an upper bound on the expected cumulative regret of **UCB-V** as follows:

$$\mathbb{E}(CR_t) \leq 10 \log(t) \sum_{k:\mu_k<\mu^*} (\frac{\sigma_k^2}{\Delta_k} + 2b), \tag{2.17}$$

where σ_k denotes the variance of reward of arm $k \in \{1, 2, \ldots, K\}$ and the rewards are bounded by $[0, b]$. Without loss of generality, we assume $b = 1$, thus Equation 2.17 becomes

$$\mathbb{E}(CR_t) \leq 10 \log(t) \sum_{k:\mu_k<\mu^*} (\frac{\sigma_k^2}{\Delta_k} + 2). \tag{2.18}$$

2.3.3.4 KL-UCB

[Garivier and Cappé, 2011] proposes a **KL-UCB** algorithm and proves that the regret of **KL-UCB** for Bernoulli distribution satisfies

$$\limsup_{t\to\infty} \frac{\mathbb{E}(CR_t)}{\log(t)} \leq \sum_{k:\mu_k<\mu^*} \frac{\Delta_k}{d(\mu^*, \mu_k)}. \tag{2.19}$$

where $d(w, v) = w \log(\frac{w}{v}) + (1-w) \log(\frac{1-w}{1-v})$ denotes the Kullback-Leibler divergence between Bernoulli distributions of parameters w and v, respectively. [Cappé et al., 2013] provides a similar result with general distributions. These results are tight as shown by [Burnetas and Katehakis, 1996].

This result leads to Table 2.1, summarizing the state-of-the-art.

Table 2.1: Upper bound on the expected cumulative regret.

Reference	Algorithm	Upper bound
[Auer et al., 2002a]	UCB	$\mathbb{E}(CR_t) \leq \sum_{k:\mu_k<\mu^*} \frac{8 \log(t)}{\Delta_k} + C$
[Audibert et al., 2006]	β-UCB	$\mathbb{E}(CR_t) \leq C' \log(2t) \sum_{k\neq k^*} (1 + \frac{\sigma_k^2}{\Delta_k})$
[Audibert et al., 2009]	UCB-V	$\mathbb{E}(CR_t) \leq 10 \log(t) \sum_{k:\mu_k<\mu^*} (\frac{\sigma_k^2}{\Delta_k} + 2)$
[Garivier and Cappé, 2011] [Cappé et al., 2013]	KL-UCB	$\limsup_{t\to\infty} \frac{\mathbb{E}(CR_t)}{\log(t)} \leq \sum_{k:\mu_k<\mu^*} \frac{\Delta_k}{d(\mu^*, \mu_k)}$

2.3.4 Adversarial bandit and pure exploration bandit

In this section we show that the parallelization of adversarial or stochastic simple regret algorithms was already studied in the literature.

- Adversarial bandit: we refer to Sections 2.5.3.3 for an overview of adversarial bandit algorithms, which are also used for computing Nash equilibria in adversarial settings.

- Parallel adversarial bandit: in a matrix setting, [Grigoriadis and Khachiyan, 1995] provides an algorithm, close to the well known EXP3 [Auer et al., 1995], and with a good parallel behavior up to p close to $\frac{K}{\log(K)}$. More precisely, [Grigoriadis and Khachiyan, 1995] shows that a pair of ϵ-optimal strategies, for a given 0-sum matrix game in $[-1,1]^{K\times K}$, can be computed in expected time $O(\frac{\log^2(K)}{\epsilon^2})$ on an $\frac{2K}{\log(2K)}$-processor machine. The product of this number of time steps and this number of processors is $O(K\log(K)/\epsilon^2)$, i.e. the sequential complexity ([Auer et al., 1995]). Therefore, the break-even point (the number of processors until which there is a significant speed-up) is at least K. To the best of our knowledge, there is no proof that this is optimal.

- Simple regret: [Bubeck et al., 2011] shows that, for T sufficiently large, the uniform exploration policy is optimal both in distribution-free and distribution-dependent terms (within logarithmic factors in the distribution-free setting and within multiplicative constants in the distribution dependent case). The uniform allocation strategy is fully parallel. Therefore, at least for T sufficiently large, a fully parallel algorithm is already available. A limitation of this positive result is that in spite of their theoretical optimality for huge values of T, algorithms with uniform allocation are not that convenient in the real (non asymptotic) world. It should however be mentioned that uniform allocation has obvious simplicity, robustness, parallelization advantages which, besides their mathematical foundation above, are preferred by practitioners in many real-world cases [Bourki et al., 2010].

2.4 Scenario-Based Decision-Making

2.4.1 Decision making in uncertain environments

Planning in power systems relies on many uncertainties. Some of them, originating in nature or in consumption, can be tackled through probabilities [RTE-ft, 2008, Pinson, 2013, Siqueira et al., 2006, Vassena et al., 2003]; others, such as technology evolution, geopolitics or CO2 penalization laws, are somewhere between stochastic and adversarial:

- Climate: The United Nations Climate Change Conference, COP21, aims at achieving a new universal agreement on climate agreement, which is an issue of cooperation and competition.

- Uranium supply: India has been using imported enriched uranium from Russia since 2001. In 2004, Russia deferred to the Nuclear Suppliers' Group and declined to supply further uranium for India's reactors. The uranium supply was not resumed until the end of 2008 (after the refurbishment was finished). Now, Russia is already supplying the India's first large nuclear power plant under a Russian-financed 3 billion contract. Moreover, in 2014, Putin agreed to help building 10 nuclear reactors in India.

- Curtailment risk: Wind and solar curtailment may occur for several reasons including transmission congestion (or local network constraints), global oversupply and operational issues [Lew et al., 2013]. Each type of curtailment occurs with variant frequency depending on the regional and local system's generation and electrical characteristics. Another example is the risk of terrorism in the congested traffic, which cannot be represented by any stochastic model.

- Geopolitical implications: Affected by the dollar, geopolitical and other factors, at the beginning of 2008 the international crude oil prices rose sharply. Another example is the Ukraine Crisis, which made Europe consider seriously

adjusting its energy policy to reduce its dependence on imported energy supply.

Handling such uncertainties is a challenge. For example, how should we modelize the risk of gas curtailment in Europe, and the evolution of oil prices ?

We discuss existing methodologies in Section 2.4.2.

2.4.2 State of the art: decision with uncertainties

The notations are as follows: K is the number of possible policies. S is the number of possible scenarios. R is the matrix of rewards and the associated reward function ($R_{k,s} = R(k,s)$), i.e. $R(k,s)$ is the reward when applying policy $k \in \mathcal{K} = \{1,\dots,K\}$ in case the outcome of uncertainties is $s \in \mathcal{S} = \{1,\dots,S\}$. The reward function is also called a utility function or a payoff function. A *strategy* (a.k.a. policy) is a random variable k with values in $\mathcal{K}$. A *mixed strategy* is a probability distribution of possible policies; this is the general case of a strategy. A *pure strategy* is a deterministic policy, i.e. it is a mixed strategy with probability 1 for one element, others having probability 0. The *exploitability* of a (deterministic or randomized) strategy k is

$$\left(\max_{k' stochastic} \min_{s\in\{1,\dots,S\}} \mathbb{E}_{k'} R(k',s)\right) - \min_{s\in\{1,\dots,S\}} \mathbb{E}_k R(k,s). \quad (2.20)$$

We refer to the choice of s as Nature's choice. This does not mean that only natural effects are involved; geopolitics and technological uncertainties are included. k is chosen by us. In fact, natural phenomena can usually be modelized with probabilities, and are included through random perturbations - they are not the point in this work - contrarily to climate change uncertainties.

2.4.2.1 Scenario-based planning

Maybe the most usual solution consists in selecting a small set $s_1,\dots,s_M$ of possible s, assumed to be most realistic. Then, for each s_j, an optimal k_i is obtained. The human then checks the matrix of the $R(k_i,s_j)$ for i and j in $\{1,\dots,M\}$. Variants of

this approach are studied in scenario planning [Saisirirat et al., 2013, Chaudry et al., 2014, Schwartz, 1996]. [Feng, 2014] provides examples with more than 1000 scenarios. When optimizing the transmission network, we must take into account the future installation of power plants, for which there are many possible scenarios - in particular, the durations involved in power plant building are not necessarily larger than constants involved in big transmission lines. The scenarios involving large wind farms, or large nuclear power plants, lead to very specific constraints depending on their capacities and locations.

2.4.2.2 Wald criterion

The Wald [Wald, 1939] criterion consists in optimizing in the worst case scenario. For a maximum problem, the *Wald-value* is

$$v = \max_{k \text{ pure strategy on } \{1,\dots,K\}} \min_{s\in\{1,\dots,S\}} R_{k,s}, \tag{2.21}$$

and the recommended policy is k realizing the max. We choose a policy which provides the best solution (maximal reward) for the worst scenario. Wald's maximin model provides a reward which is guaranteed in all cases. Implicitly, it assumes that Nature will make its decision in order to bother us, and, in a more subtle manner, Nature will make its decision while knowing what we are going to decide. It is hard to believe, for example, that the ultimate technological limit of photovoltaic units will be worse if we decide to do massive investments in solar power. Therefore, Wald's criterion is too conservative in many cases; hence the design of the Savage criterion.

2.4.2.3 Savage criterion

The *Savage-value* [Savage, 1951] is:

$$v = \min_{k \text{ pure strategy on } \mathcal{K}} \max_{s\in\mathcal{S}} regret(k,s). \tag{2.22}$$

where

$$regret(k,s) = \max_{k'\in\mathcal{K}} (R_{k',s} - R_{k,s}).$$

The Savage criterion is an application of the Wald maximin model to the regret. Contrarily to Wald's criterion, it does not focus on the worst scenario. Its interpretation is that we optimize the guaranteed loss compared to an anticipative choice (anticipative in the sense: aware of all future outcomes) of decision. On the other hand, Nature still makes its decision after us, and has access to our decision before making its decision - Nature, in this model, can still decide to reduce the technological progress of wind turbines just because we have decided to do massive investments in wind power.

2.4.2.4 Nash equilibria

The principle of the Nash equilibrium is that contrarily to what is assumed in Wald's criterion (Eq. 2.21), there is no reason for Nature (the opponent) to make a decision *after* us, and to know what we have decided. The *Nash-value* v is

$$v = \max_{k \text{ mixed strategy on } \mathcal{K}} \min_{s \in \mathcal{S}} \mathbb{E}_k R(k, s).$$

As a mixed strategy is used, the fact that the maximum is written before the min does not change the result [v. Neumann, 1928]; v is also equal to

$$\min_{s \text{ r.v. on} \mathcal{S}} - \max_{k} \mathbb{E}_k R(k, s).$$

where r.v. stands for "random variable". The exploitability (Eq. 2.20) of a (possibly mixed) strategy k is equivalent to

$$\textit{Nash-value} - \min_{s \in \mathcal{S}} \mathbb{E}_k R(k, s).$$

A Nash strategy is a strategy with exploitability equal to 0. A Nash strategy always exists; it is not necessarily unique. A Nash equilibrium, for a finite-sum problem, is a pair of Nash strategies for us and for Nature respectively. In the general case, a Nash strategy is not pure. Criteria for Nash equilibria corresponds to Nature and us making decision privately, i.e. without knowing what each other will do. In this sense, it is more intuitive than other criteria. Other elements around Nash equilibria are defined in Section 2.5.3.

Table 2.2: Comparison between several tools for decision under uncertainty.

Method	Extraction of policies	Extraction of critical scenarios	Computational cost	Interpretation
Wald	One	One per policy	$K \times S$	Nature decides later, minimizing our reward.
Savage	One	One per policy	$K \times S$	Nature decides later, maximizing our regret.
Nash	Nash-optimal	Nash-optimal	$(K+S)\times \log(K+S)$	Nature decides privately, before us.
Scenarios	Handcrafted	Handcrafted	$K' \times S'$	Human expertise

2.4.2.5 Other decision tools

Other possible tools for partially adversarial decision making are multi-objective optimization (i.e. for each s, there is one objective function $k \mapsto R(k, s)$) and possibilistic reasoning [Dubois and Prade, 2012]. These tools rely intensively on human experts, a priori (selection of scenarios) or a posteriori (selection in the Pareto set).

2.4.3 Comparison between various decision tools

Let us compare the various discussed policies, where K is the number of possible investment policies, S is the number of scenarios, K' is the number of displayed policies, S' is the number of displayed policies; we provide an overview in Table 2.2.

2.5 Games

2.5.1 AIs for games: the fully observable case

For a while, the focus in game AI research was fully observable games, and the standard algorithm for game AIs was the alpha-beta approach. The basic principle of alpha-beta is a retrograde

computation of game-theoretic values of states [Richards and Hart, 2006] (induction with min/max operators), plus two key ingredients:

- Use of approximate game-theoretic values, at some depth of the recursive analysis [Shannon, 1988];
- Pruning of useless branches of the game tree [Knuth and Moore, 1975].

Other important features of modern alpha-beta algorithms include endgame retrograde analysis (in particular in Chess), opening books, iterative deepening [Korf, 1985].

However, since [Coulom, 2006], Monte Carlo Tree Search invaded the field and outperforms alpha-beta for many games, in particular those for which a fast and reliable evaluation function is not available.

Monte Carlo simulations can be used inside tree search algorithms [Bouzy, 2004], simple Monte Carlo search [Bruegmann, 1993,Cazenave, 2006,Cazenave and Borsboom, 2007], Nested Monte Carlo [Cazenave, 2009] and the famous Monte Carlo Tree Search (MCTS) algorithm [Coulom, 2006] which is at the heart of the current revolution in computer games. A key issue is that Monte Carlo simulations are biased. A classical example in Go is the case where several semeais[6] are in progress. Monte Carlo will typically [Cazenave and Saffidine, 2011] predict that each of them is won, independently, by Black with probability 50%. If there are three such semeais, and if Black can only win the game by winning all of them, then the Monte Carlo estimate will be $\frac{1}{2^3} = 12.5\%$. However, if the semeais are, in fact, independent wins for Black, the real value of the position is 100%. Positions with several local fights are, for the same reason, prone to be poorly evaluated.

To correctly evaluate positions in games is of critical importance and a large body of work has been devoted to the subject. To palliate the bias induced by Monte Carlo simulations one can craft her own simulation policy through, for instance,

[6]Semeai, sometimes called capturing race, is the situation where neither of the two groupes can survive except capturing the other (see details in `https://en.wikipedia.org/wiki/Capturing_race`).

domain specific knowledge [Billings et al., 1999], Reinforcement Learning [Tesauro and Galperin, 1996,Gelly and Silver, 2007], Supervised Learning [Coulom, 2007] or Simulation Balancing [Silver and Tesauro, 2009,Huang et al., 2010]. Other typical examples are hill-climbing [Chaslot et al., 2008] or directly by hand [Gelly et al., 2006]. To this date, human expertise remains the most efficient solution for improving the simulations [Baudiš and Gailly, 2011].

So far, people focused mostly on improving the quality of the simulation policy. We propose a rather different approach. In Chapter 12, we put emphasis on extracting more information from the simulation results rather than improving the simulation policy. Basically the idea is to, instead of using the average outcome of many simulations to evaluate a game position, modify the way they are averaged by adjusting their relative weights in the evaluation of a position. Section 12.1 describes the approach more formally.

2.5.2 Improving AI: endgames and opening books

In this section, we discuss tools for generically improving an AI for games, without modifying "internally" this AI. Solving endgames makes the algorithm faster by storing solved positions. Classical tools are retrograde analyze [Russell and Wolfe, 2005] and Nalimov's endgame tables (EGTs) [Nalimov et al., 2000]. Building an opening book is a challenge; human expertise is usually limited, and detecting errors in an opening book is as important as adding new data. [Gaudel et al., 2010] has shown how to apply portfolios for combining, correcting, and bootstrapping (more precisely, improving by combining subsamples) opening books. We will show how to apply random seed optimization for improving the opening book (chapter 11). We will then apply random seed optimization on the fly, for given positions, in chapter 12 - somehow building an opening book specifically for a given input position. Chapter 11 also includes the combination of variants.

2.5.3 Partial Observation and Nash Equilibria

Whereas previous sections consider fully observable games (though some of the presented algorithms have extensions in the partially

observable setting), this section is devoted to partial observation. We first present matrix games, which are the typical example of partial observation, and then we discuss some algorithms for partially observable games.

2.5.3.1 Matrix games

Consider a matrix M of size $K_1 \times K_2$ with values in $\{0, 1\}$. Player 1, the row player, chooses an action $i \in [[1, K_1]]$ and player 2, the column player, chooses an action $j \in [[1, K_2]]$; both actions are chosen simultaneously. Then, player 1 gets reward $M_{i,j}$ and player 2 gets reward $1 - M_{i,j}$. The game therefore sums to 1 (we consider games summing to 1 for convenience, but 0-sum games are equivalent).

2.5.3.2 Nash equilibria

A Nash Equilibrium (Definition 1) is a pair (x^*, y^*) (in $[0, 1]^{K_1}, [0, 1]^{K_2}$ and summing to 1) such that if i is distributed according to the distribution x^* (i.e. $i = k$ with probability x^*_k) and if j is distributed according to the distribution y^* (i.e. $j = k$ with probability y^*_k) then none of the players can expect a better average reward by changing unilaterally its strategy.

Definition 1 (Nash Equilibria)**.** *A Nash Equilibrium (NE)* (x, y) *of a two-player 1-sum matrix game* $M_{K_1 \times K_2}$ *is a vector* $x \in [0, 1]^{K_1}$ *and a vector* $y \in [0, 1]^{K_2}$ *such that*

$$\sum_{i=1}^{K_1} x_i = \sum_{j=1}^{K_2} y_j = 1, \tag{2.23}$$

$$\forall x', y', \ xMy' \geq xMy \geq x'My, \tag{2.24}$$

with x' *and* y' *non negative vectors with sum* 1 *and sizes* K_1, K_2 *respectively.*

There might be several NE, but the value of the game M, given by $v = x^*My^*$, remains the same. Moreover, we define an ϵ-NE as a pair (x^*, y^*) such that

$$\inf_{y'} x^T My' > x^T My - \epsilon \quad \text{and} \quad \sup_{x'} (x')^T My < x^T My + \epsilon. \tag{2.25}$$

2.5.3.3 Algorithms for computing Nash equilibria in adversarial settings

A state-of-the-art bandit algorithm to approximate a NE (which includes matrix games) is EXP3 [Auer et al., 1995]. Here we present the version used in [Bubeck et al., 2009].

At iteration $t \in [[1, T]]$, our version of EXP3 proceeds as follows (this is for one of the players; the same is done, independently, for the other player):

- At iteration 1, S is initialized as a null vector.
- Action i is chosen with probability $p(i) = \alpha_t/K + (1 - \alpha_t) \times \exp(\eta_t S_i)/\sum_j \exp(\eta_t S_j)$ for some sequences $(\alpha)_{t\geq 1}$ and $(\eta_t)_{t\geq 1}$, e.g. in [Bubeck and Cesa-Bianchi, 2012] $\alpha_t = 0$ and $\eta_t = \log(K)/(tK)$ or in [Bubeck et al., 2009] $\eta_t = \min(\frac{4}{5}\sqrt{\frac{\log(K)}{TK}}, \frac{1}{K})$ and $\alpha_t = K\eta_t$.
- Let r be the received reward.
- Update S_i: $S_i \leftarrow S_i + r/p(i)$ (and S_j for $j \neq i$ is not modified).

This algorithm, as well as its variants, converges to the NE as explained in [Bubeck et al., 2009, Bubeck and Cesa-Bianchi, 2012] (see also [Grigoriadis and Khachiyan, 1995, Auer et al., 1995]).

The state of the art of the computation of Nash equilibria in matrix games is summarized in Table 2.3.

2.5.3.4 Others

The computation of Nash equilibria in non-adversarial settings is beyond the scope of the present document; the task is way more complex. Algorithms for computing Nash equilibria in a less black-box manner, using the tree structure of the game, are also not discussed here.

Table 2.3: Complexity of computing approximate Nash Equilibrium in matrix games. In this framework, the input is provided as an Oracle which can computes an entry in the matrix; the complexity is indeed in many cases smaller than the size of the matrix, hence a sublinear complexity. LP and [Grigoriadis and Khachiyan, 1995] require that the Oracle is deterministic whereas EXP3, Inf and our Chapter 9 do not.

Algorithm	Complexity	Exact solution?	Confidence	Time
LP [von Stengel, 2002]	$O(K^\alpha), \alpha > 6$	yes	1	constant
[Grigoriadis and Khachiyan, 1995]	$O(\frac{K \log(K)}{\epsilon^2}$	no	1	random
[Grigoriadis and Khachiyan, 1995]	$O(\frac{\log^2(K)}{\epsilon^2}$ with $\frac{K}{\log(K)}$-processor	no	1	random
EXP3 [Auer et al., 1995]	$O(\frac{K \log(K)}{\epsilon^2}$	no	$1-\theta$	constant
Inf [Bubeck et al., 2009]	$O(\frac{K \log(K)}{\epsilon^2}$	no	$1-\theta$	constant
Chapter 9	$O(k^3 K \log(K)$	yes	$1-\theta$	constant

Part II

Contributions in noisy optimization

3 Contributions to noisy optimization: outline

Noisy optimization is the optimization of fitness functions corrupted by noise. A black-box noisy optimization consists in searching for the optimum (e.g. minimum) x^* of some noisy fitness function $f : \mathbb{R}^d \mapsto \mathbb{R}$ by successive calls to fitness without using any internal property of the fitness function. f is a random process, and equivalently it can be viewed as a mapping $(x, w) \mapsto f(x, w)$ where $x \in \mathbb{R}^d$ and w is a random variable independently sampled at each call to f. There is no knowledge about the fitness function. We aim at finding a good approximation of x^*.

We refer to Section 2.1 for more on noisy optimization, optimization criteria and variance reduction methods.

Chapter 4 discusses the use of resampling for continuous noisy optimization. In such cases, the simple regret can decrease linearly with the number of evaluations - so-called slope -1. We then switch to more specific cases, where optimal rates are slower (square root of the above) - this is the price of robustness. Chapter 5 applies resampling to the specific case of evolutionary algorithms. Chapter 6 applies resampling to the case of differential evolution. In these two cases, we reach the expected rate. Chapter 7 applies "grey" box tricks, in particular common random numbers, also known as pairing, for improving the convergence.

4 Resampling in continuous noisy optimization

In this chapter, we propose a Hessian-based iterative noisy algorithm and show general results, based on some properties of optimum estimates. Our proposed algorithm recovers most existing results, except the slope of simple regret -1 obtained by [Fabian, 1967] when arbitrarily many derivatives are supposed to exist. In particular, using a noisy evaluation of the gradient and Hessian, we get at best $s(SR) = -1$ or $s(CR) = \frac{1}{2}$ (not simultaneously; the former with parameters optimized for SR and the latter with parameters optimized for CR) for constant noise variance on quadratic positive definite functions, as well as $s(SR) = -\frac{2}{3}$ and $s(CR) = \frac{1}{2}$ (also not simultaneously; the former with parameters optimized for SR and the latter with parameters optimized for CR) for functions which have positive definite second order Taylor expansion, as in [Fabian, 1967, Shamir, 2013] respectively.

We can also get $s(SR) = -\frac{2}{3}$ and $s(CR) = \frac{2}{3}$ (simultaneously) in the same setting. We get $s(SR) = -1$ and $s(CR) = 0$ (simultaneously) with linearly decreasing variance as in [Rolet and Teytaud, 2010], $s(SR) = -\infty$ and $s(CR) = 0$ (simultaneously) with quadratically decreasing variance as conjectured in [Jebalia and Auger, 2008] - for a different algorithm. In addition, our results are applicable with arbitrary surrogate models, provided that they verify the LSE assumption (Definition 2).

Our results are summarized in Table 4.1.

Section 4.1.1 presents a general iterative algorithm, which uses a *sampling tool* and an *optimum estimator*. It relies on a *LSE assumption* (Definition 2) which is central in the assumptions for the

main theorem. Section 4.1.2 provides examples of sampling tools, called Sampler(·) functions, and examples of optimum estimators, given by Opt(·) functions, which match the assumptions in Section 4.1.1.

4.1 The Iterative Noisy Optimization Algorithm (Inoa)

4.1.1 General framework

The Iterative Noisy Optimization Algorithm (Inoa) is presented in Algorithm 2. It uses a pair of functions (Sampler, Opt). Specific tasks and properties of these functions are described below and examples of such functions are given in Section 4.1.2.

Algorithm 2 Iterative Noisy Optimization Algorithm (Inoa).

Input: Step-size parameters $\alpha > 0$, $A > 0$
Input: Number of revaluations parameters $\beta \geq 0$, $B > 0$
Input: Initial points $x_1^{opt} = \tilde{x}_1$
Input: A fitness function (also termed noisy objective function)
Input: A sampler function Sampler(·)
Input: An optimizer function Opt(·)
Return approximations $(x_n^{opt})_{n\geq 1}$, recommendations $(\tilde{x}_m)_{m\geq 1}$, evaluation points $(x_{n,i})_{n\geq 1, i\in\{1,\dots,r_n\}}$, fitness evaluations $(y_{n,i})_{n\geq 1, i\in\{1,\dots,r_n\}}$
$n \leftarrow 1$
while The computation time is not elapsed **do**
    Compute step-size $\sigma_n = A/n^\alpha$
    Compute revaluations number $r_n = B\lceil n^\beta \rceil$
    for $i = 1$ to r_n **do**
        $x_{n,i} = \text{Sampler}(x_n^{opt}, \sigma_n, i)$
        $y_{n,i} =$ fitness evaluation at $x_{n,i}$
    end for
    Compute next approximation $x_{n+1}^{\text{opt}} = \text{Opt}(x_n^{opt}, (x_{n,i}, y_{n,i})_{i\in\{1,\dots,r_n\}})$
    $n \leftarrow n + 1$
end while

Sampler is the element of the algorithm that provides new search points: given a point x in the search space, Sampler provides new search points that lie on the neighborhood of x. More

precisely, $\forall i \in \{1, 2, \dots\}$, $\textsc{Sampler}(x, \sigma, i)$ outputs a point x_i such that it satisfies $\|x_i - x\| \leq 2\sigma$, with σ a given step-size. Notice that we do not make any assumptions on *how* the new search points are chosen, we only ask for them to be within a given maximal distance from the generator point x. Opt corresponds to the optimum estimator of the algorithm: given x, $x_1, \dots, x_r$ and $y_1, \dots, y_r$ with $y_i = f(x_i, w_i)$ (with w_i independent copies of w), Opt provides an estimate $x^{opt} := \textsc{Opt}(x, (x_i, y_i)_{i \in \{1,\dots,r\}})$ of x^*, the argmin of $\mathbb{E}f$. Additionally, for the sake of convergence, the pair $(\textsc{Sampler}, \textsc{Opt})$ verifies a property defined in Definition 2 and called the *Low squared error assumption (LSE)*.

The algorithm provides the sequence $(x_n^{opt})_{n \geq 1}$, indexed with the number of *iterations*, but the recommendations $(\tilde{x}_n)_{n \geq 1}$ in the definitions of Section 2.1.1 have to be indexed by the number of *evaluations*. Hence, for $m \geq 1$, the recommendation $\tilde{x}_m$ are defined by $\tilde{x}_m = x_{n(m)}$ with $n(m) = \max\{n; \sum_{i=1}^{n-1} r_i \leq m\}$, since there are r_i evaluations at iteration i.

Definition 2 (Low squared error assumption (LSE))**.** *Given a domain $D \subseteq \mathbb{R}^d$, an objective function $f : D \to \mathbb{R}$ corrupted by noise. We assume that f is such that $\mathbb{E}_w f(x, w)$ has a unique optimum x^*. Let $C > 0$, $U > 0$, and $z \in \{0, 1, 2\}$. Then, we say that* $(\textsc{Sampler}, \textsc{Opt})$ *has a $(2z-2)$-low squared error for f, C, U, S if $\forall (r, \sigma) \in S$*

$$\|x - x^*\| \leq C\sigma \implies$$
$$\text{for any positive integer } r, \quad \mathbb{E}(\|x^{opt} - x^*\|^2) \leq U\frac{\sigma^{2z-2}}{r}, \qquad (4.1)$$

where x^{opt} is provided by the Opt *function, which receives as input*

- *the given x,*
- *r search points $(x_i)_{i \in \{1,\dots,r\}}$, outputs of* Sampler*,*
- *and their corresponding noisy fitness values.*

In the latter definition, z is related to the intensity of the noise. Recall that we consider three types of noise, namely *constant*, *linear* or *quadratic* in function of the SR. More precisely, we

consider that $Var(f(x,w)) = O\left([\mathbb{E}_w f(x,w) - \mathbb{E}_w f(x^*,w)]^z\right)$ with $z \in \{0,1,2\}$.

The rate $O(1/r)$ for a squared error is typical in statistics, when estimating some parameters from r samples. We will see in examples below that the scaling with σ is also relevant, as we recover, with the LSE as an intermediate property, many existing rates.

We can work with the additional assumption that $x^* = 0$ without loss of generality. Hence from now on, examples, proofs and theorems are displayed with $x^* = 0$.

4.1.2 Examples of algorithms verifying the LSE assumption

In this section we provide two examples of pairs (Sampler, Opt) which verify Definition 2. Not only Sampler and Opt are important, but also the type of functions we consider (conditions for expectation and variance on the properties that show the verification of LSE). The first example uses an estimation of the gradient of the function to produce an approximation to the optimum. The idea is simple: if we have x, a current approximation to the optimum, we sample around it and use these points to estimate the gradient and obtain the next approximation.

Let $(e_j)_{j=1}^d$ be the canonical orthonormal basis of $\mathbb{R}^d$. Sampler will output search points $x \pm \sigma e_j$ for some $j \in \{1, \dots d\}$. Therefore, the set of points that Sampler has access to is $E' := E'_+ \cup E'_-$ where $E'_+ = (x + \sigma e_j)_{j=1}^d$ and $E'_- = (x - \sigma e_j)_{j=1}^d$, and E' is ordered[1] We assume that Sampler outputs at the end a sample of r points, all belonging to E'. Note that as soon as $r > 2d$ the search points are sampled several times. However, the values of the objective function of the same search point evaluated two or more times will differ due to the noise in the evaluation. On the other hand, Opt takes this regular repeated sample around x and its corresponding

[1] $E' = \{x + \sigma e_1, \dots, x + \sigma e_d, x - \sigma e_1, \dots, x - \sigma e_d\}$. In this example, when Sampler is queried for the i-th time it will output the i-th point of E'. For the case $i > 2d = |E'|$, to simplify the notation we define a slightly different version of the usual modulo operation, denoted "$\overline{\text{mod}}$", such that for any i, d, $i \ \overline{\text{mod}}\ d = 1 + ((i-1) \bmod d)$. Therefore, when $i > 2d = |E'|$, Sampler will output the $(i \ \overline{\text{mod}}\ 2d)$-th point of E'.

objective function values to compute an average value for each of the points in E'. Hence, the average is done over at least $\lfloor r/(2d) \rfloor$ function evaluations and it allows to reduce the noise and obtain a more confident - still noisy - evaluation. With these averaged values, Opt computes the approximated optimum. Let us consider

$$Y_{j+} = \{\text{all evaluations of } x + \sigma e_j\}$$

and

$$Y_{j-} = \{\text{all evaluations of } x - \sigma e_j\}$$

and use the notation $x^{(j)}$ to refer to the j-th coordinate of x. Also, when we use $\sum Y_{j+}$, with Y_{j+} a set, it will simply denote that we sum over all the elements of the multiset Y_{j+}.

Property 1 enunciates the fact that the pair (Sampler, Opt) defined in Example 1 satisfies the Low Squared Error assumption (Definition 2).

Property 1. ***(*Sampler, Opt*) in Example 1 satisfy*** $(2z-2)$***-LSE for the sphere function.*** *Let f be the function to be optimized, and $z \in \{0,1,2\}$. We assume that:*

$$\textbf{\textit{Framework 1}} \quad \left| \begin{array}{l} \mathbb{E}_w f(x,w) = \|x\|^2 \\ Var(f(x,w)) = O(\|x\|^{2z}) \text{ for some } z \in \{0,1,2\} \end{array} \right. \tag{4.6}$$

$$\tag{4.7}$$

Then there is $C > 0$, such that if x and σ verify $\|x\| \leq C\sigma$, then

$$\mathbb{E}(\|x^{opt}\|^2) = O(\sigma^{2z-2}/r). \tag{4.8}$$

where x^{opt} is the output of Opt$(x, (x_i, y_i)_{i\in\{1,\dots,r\}})$*, $(x_i)_{i\in\{1,\dots,r\}}$ is the output of* Sampler *and $(y_i)_{i\in\{1,\dots,r\}}$ their respective noisy fitness values.*

The method using gradients described above is already well studied, as well as improved variants of it with variable step-sizes, (see [Fabian, 1967, Chen, 1988, Shamir, 2013]).

Therefore, we now switch to the second example, including the computation of the Hessian.

Example 1 Gradient based method verifying the LSE assumption (Definition 2). Given $x \in \mathbb{R}^d$ and $\sigma > 0$, Sampler and Opt are defined as follows.

function Sampler(x, σ, i)

$$j \leftarrow i \ \overline{\mathrm{mod}}\ 2d \tag{4.2}$$

$$x_i \leftarrow \text{the } j\text{-th point in } E' \tag{4.3}$$

return x_i
end function

function Opt$(x, (x_i, y_i)_{i\in\{1,\dots,r\}})$
for $j = 1$ to d **do**

$$\hat{y}_{j+} \leftarrow \frac{1}{|Y_{j+}|}\sum Y_{j+}, \qquad \hat{y}_{j-} \leftarrow \frac{1}{|Y_{j-}|}\sum Y_{j-} \tag{4.4}$$

$$\hat{g}^{(j)} \leftarrow \frac{\hat{y}_{j+} - \hat{y}_{j-}}{2\sigma} \tag{4.5}$$

end for

$$x^{opt} \leftarrow x - \frac{1}{2}\hat{g}$$

return x^{opt}
end function

As in the Example 1, we consider a set of search points that are available for Sampler to output. Let us define $E'' = \{x \pm \sigma e_i \pm \sigma e_j; 1 \le i < j \le d\}$. And so the sample set will be E, which includes the set E'' defined above and the sample set E' defined for Example 1. Therefore, $|E| = 2d^2$ (E' has cardinal $2d$ and E'' has cardinal $2d(d-1)$). Also, we define naturally the sets of evaluations of the search points as follows:

$$Y_{j+,k+} = \{\text{all evaluations of } x + \sigma e_j + \sigma e_k\},$$

$$Y_{j+,k-} = \{\text{all evaluations of } x + \sigma e_j - \sigma e_k\},$$

$$Y_{j-,k+} = \{\text{all evaluations of } x - \sigma e_j + \sigma e_k\},$$

$$Y_{j-,k-} = \{\text{all evaluations of } x - \sigma e_j - \sigma e_k\}.$$

Note that in Example 3, the output of Sampler(x, σ, i) are equally distributed over E so that each of them is evaluated at least $\lfloor r/2d^2 \rfloor$ times. The pair (Sampler, Opt) defined in Example 3 verifies the LSE assumption (Property 2) when the noisy objective function is approximately quadratic (Eq 4.15) and the noise follows the constraint given by Equation 4.16.

Property 2. ***(*Sampler, Opt*) in Example 3 satisfy LSE.*** *Let f be the function to be optimized, $z \in \{0, 1, 2\}$. We assume that:*

Framework 2

$$\begin{cases} \mathbb{E}_w f(x,w) = \sum_{1 \le j,k \le d} c_{j,k} x^{(j)} x^{(k)} \\ \qquad\qquad + \sum_{1 \le j,k,l \le d} b_{j,k,l} x^{(j)} x^{(k)} x^{(l)} + o(\|x\|^3), \ \textit{with } c_{j,k} = c_{k,j} \\ Var(f(x,w)) = O(\|x\|^{2z}) \ \textit{where } z \in \{0,1,2\}. \end{cases} \tag{4.15}$$

(4.16)

Assume that there is some $c_0 > 0$ such that h is positive definite with least eigenvalue greater than $2c_0$, where h is the Hessian of $\mathbb{E}f$ at 0, i.e $h = (2c_{j,k})_{1 \le j,k \le d}$.

Then there exists $\sigma_0 > 0$, $K > 0$, $C > 0$, such that for all σ that satisfies i) $\sigma < \sigma_0$ and ii) $\sigma^{6-2z} \le K/r$, and for all x such that

$$\|x\| \le C\sigma, \tag{4.17}$$

we have

$$\mathbb{E} \ \|x^{opt}\|^2 = O\left(\frac{\sigma^{2z-2}}{r}\right),$$

where x^{opt} is the output of Opt$(x, (x_i, y_i)_{i=1}^r)$*, $(x_i)_{i=1}^r$ are the output of* Sampler(x, σ, i) *and the $(y_i)_{i=1}^r$ are their respective noisy fitness values.*

Remark 1. *Using the expressions of σ and r given by* Inoa*, if $(6 - 2z)\alpha \ge \beta$, and given $A > 0$, then there exists a constant $B_0 > 0$ such that if $B > B_0$ then the condition $\sigma_n^{6-2z} \le K/r_n$ is satisfied.*

Algorithm 3 Noisy-Newton method verifying the $(2z-2)$-LSE assumption. Given $x \in \mathbb{R}^d$, $\sigma > 0$ and $c_0 > 0$, Sampler and Opt are defined as follows. $t(M)$ denotes the transpose of matrix M.

Example 1 (h). **function** Sampler(x, σ, i)

$$j \leftarrow i \ \overline{\mathrm{mod}}\ 2d^2 \tag{4.9}$$

$$x_i \leftarrow \text{the } j\text{-th point in } E \tag{4.10}$$

return x_i
end function

function Opt$(x, (x_i, y_i)_{i\in\{1,\dots,r\}})$
for $j = 1$ to d **do**

$$\hat{y}_{j+} \leftarrow \frac{1}{|Y_{j+}|}\sum Y_{j+}, \quad \hat{y}_{j-} \leftarrow \frac{1}{|Y_{j-}|}\sum Y_{j-} \tag{4.11}$$

$$\hat{g}^{(j)} \leftarrow \frac{\hat{y}_{j+} - \hat{y}_{j-}}{2\sigma} \tag{4.12}$$

end for
for $1 \le j, k \le d$ **do**
$\hat{y}_{j+,k+} \leftarrow \frac{1}{|Y_{j+,k+}|}\sum Y_{j+,k+}$, $\hat{y}_{j+,k-} \leftarrow \frac{1}{|Y_{j+,k-}|}\sum Y_{j+,k-}$
$\hat{y}_{j-,k+} \leftarrow \frac{1}{|Y_{j-,k+}|}\sum Y_{j-,k+}$, $\hat{y}_{j-,k-} \leftarrow \frac{1}{|Y_{j-,k-}|}\sum Y_{j-,k-}$
$\hat{h}^{(j,k)} \leftarrow \frac{\left(\hat{y}_{j+,k+}-\hat{y}_{j-,k+}\right)-\left(\hat{y}_{j+,k-}-\hat{y}_{j-,k-}\right)}{4\sigma^2}$
end for

$$\hat{h} \leftarrow \frac{\hat{h} + t(\hat{h})}{2}$$

if $\hat{h}$ is positive definite with least eigenvalue greater than c_0 **then**

$$x^{opt} \leftarrow x - (\hat{h})^{-1}\hat{g} \tag{4.13}$$

else

$$x^{opt} \leftarrow x \tag{4.14}$$

end if
return x^{opt}
end function

4.2 Convergence Rates of INOA

Sections 4.2.1 and 4.2.2 provide, respectively, the main result and its applications, namely cumulative regret analysis and simple regret analysis for various models of noise. The special case of twice-differentiable functions is studied in Section 4.2.3.

4.2.1 Rates for various noise models

In this section, we present the main result, i.e. the convergence rates of INOA.

Theorem 1 (Rates for various noise models)**.** *Consider some $A > 0$ and consider the iterative noisy optimization algorithm (*INOA*, Algorithm 2, with parameters A, B, α, β). Assume that* (SAMPLER, OPT) *has a $(2z-2)$-low squared error assumption (LSE, Definition 2) for some f, C, U, S. Assume that $B > B_0$, where B_0 depends on α, β and A only. Let us assume that* INOA *provides (r_n, σ_n) always in S, and let us assume that*

$$1 < \beta + \alpha(2z-4), \tag{4.18}$$

Consider $\delta > 0$. Then there is $C > 0$, such that if $x_1^{opt} = \tilde{x}_1$ satisfies $\|x_1^{opt}\| \leq CA$, then with probability at least $1-\delta$,

$$\forall n, \quad \|x_n^{opt}\| \leq C\sigma_n \tag{4.19}$$

$$\forall n, \forall i \leq r_n, \quad \|x_{n,i}\| \leq (C+2)\sigma_n. \tag{4.20}$$

Remark 2. *It is assumed that given x,* SAMPLER *provides a new search point x_i such that $\|x_i - x\| \leq 2\sigma$ (see Section 4.1.1). This together with Equation 4.19 gives $\|x_{n,i}\| \leq \|x_{n,i} - x_n^{opt}\| + \|x_n^{opt}\| \leq (C+2)\sigma_n$. Hence Equation 4.20 holds if Equation 4.19 holds; we just have to show Equation 4.19.*

General organization of the proof of Equation 4.19: Assume that Equation 4.18 holds. Consider a fixed $C > 0$ and $1 > \delta > 0$. Consider hypothesis H_n: for any $1 \leq i \leq n$, $\|x_i^{opt}\| \leq C\sigma_i$ with probability at least $1 - \delta_n$, where

$$\delta_n = \sum_{i=1}^{n} c i^{-\beta-\alpha(2z-4)}. \tag{4.21}$$

c is chosen such that $\forall n \geq 1, \delta_n \leq \delta$. By Equation 4.18, $\sum_{i=1}^{\infty} i^{-\beta-\alpha(2z-4)} = \Delta < \infty$, and $c = \delta/\Delta$ is suitable. We prove that for any positive integer n, H_n holds. The proof is by induction on H_n. H_1 is true since x_1^{opt} is chosen such that $\|x_1^{opt}\| \leq CA$, i.e $\|x_1^{opt}\| \leq C\sigma_1$.

4.2.2 Application: the general case

Theorem 1 ensures some explicit convergence rates for SR and CR depending on parameters α, β and z.

Corollary 1. *Consider the context and assumptions of Theorem 1, including some* (Sampler, Opt) *which has a* $(2z-2)$*-LSE (Definition 2) for some* f, C, U, S *such that for all* n, $(r_n, \sigma_n) \in S$, *and let us assume that* $\mathbb{E}_w f(x,w) - \mathbb{E}_w f(x^*,w) = O(\|x - x^*\|^2)$.

Then, the simple regret of Inoa *of has slope* $s(SR) \leq \frac{-\alpha(2z-2)-\beta}{\beta+1}$ *and the cumulative regret has slope* $s(CR) \leq \frac{\max(0,1+\beta-2\alpha)}{1+\beta}$.

Quadratic case: in the special case $z = 0$ *and if* $\mathbb{E}f$ *is quadratic (i.e* $\mathbb{E}_w f(x,w) = \sum_{1 \leq j,k \leq d} c_{j,k} x^{(j)} x^{(k)}$, *we get* $s(SR) \leq \frac{2\alpha-\beta}{\beta+1}$.

4.2.3 Application: the smooth case

Table 4.1 presents optimal $s(SR)$ and $s(CR)$ in the more familiar case of smooth functions, with at least two derivatives. All results in this table can be obtained by Inoa with Opt and Sampler as in Example 3 and the provided parametrizations for α and β, except the result by [Fabian, 1967] assuming many derivatives.

In all cases except the quadratic case with $z = 0$, we assume $(6 - 2z)\alpha > \beta$, so that the LSE assumption holds for Inoa with Opt and Sampler as in Example 3 (see Property 2) and we assume $1 < \beta + \alpha(2z - 4)$ so that Equation 4.18 in Theorem 1 holds. Regarding the special case of $z = 0$ and quadratic function, the equation to satisfy is $1 < \beta - 4\alpha$. Please note that in this last case, the assumption $(6-2z)\alpha > \beta$ is not necessary. We then find out values of α and β such that good slopes can be obtained for CR and SR. Algorithms ensuring a slope $s(CR)$ in this table also ensure a slope $s(UR) = \frac{1}{2}(s(CR) - 1)$. It follows that the optimal parametrization for UR is the same as the optimal parametrization for CR.

We consider parameters optimizing the CR (left) or SR (right) - and both simultaneously when possible. These results are for B constant

Table 4.1: $s(SR)$ and $s(CR)$ for INOA for various values of α and β, in the case of twice-differentiable functions. The references mean that our algorithm gets the same rate as in the cited paper. No reference means that the result is new.

z	optimized for CR		optimized for SR	
	$s(SR)$	$s(CR)$	$s(SR)$	$s(CR)$
0 (constant var)	$\alpha \simeq \infty, \beta \simeq 4\alpha + 1^+$		$\beta = 6\alpha, \alpha = \infty$	
	$-1/2$	$1/2$ [Fabian, 1967] [Shamir, 2013]	$-2/3$ [Dupač, 1957]	$2/3$
0 and ∞-differentiable.			-1 [Fabian, 1967]	
0 and "quadratic"			$\alpha = 0, \beta \simeq \infty$	
			-1 [Dupač, 1957]	
1 (linear var)	$\alpha \simeq \infty, \beta \simeq 2\alpha + 1^+$			
	-1 [Rolet and Teytaud, 2010]	0	-1	0
2 (quadratic var)	$\alpha \simeq \infty, \beta > 1$			
	$-\infty$	0	$-\infty$	0

but large enough. Infinite values mean that the value can be made arbitrarily negatively large by choosing a suitable parametrization. X^+ denotes a value which should be made arbitrarily close to X by superior values, in order to approximate the claimed rate.

Results are not adaptive; we need a different parametrization when $z = 0$, $z = 1$, $z = 2$. Also, for $z = 0$, we need a different parametrization depending on whether we are interested in CR or SR.

4.3 Conclusion and further work

We have shown that estimating the Hessian and gradient can lead to fast convergence results. In fact, with one unique algorithm we obtain many of the rates presented by

- [Spall, 2009, Shamir, 2013] in the case of a constant variance noise for simple regret and cumulative regret respectively.
- [Rolet and Teytaud, 2010, Coulom et al., 2011] ($z = 1$) and [Jebalia and Auger, 2008] ($z = 2$) for a larger space of functions than in these papers, where sphere functions are considered.

In summary, we observe on the Table 4.1 that results obtained here recover most previous results discussed in the introduction. And also the results presented here cover all the analyzed criteria: simple regret, cumulative regret, uniform rates.

Compared to [Spall, 2009], our algorithm uses more evaluations per iteration. This has advantages and drawbacks. The positive part is that it is therefore more parallel. For example, for $z = 0$, and an algorithm optimized for SR, we get $s(SR) = -2/3$; this rate is the same as the one in [Spall, 2009] in terms of number of evaluations, i.e. the number of evaluations is proportional to $(1/sr)^{2/3}$ for a simple regret sr, but our evaluations are grouped into a small number of iterations. On the other hand, it is far less convenient in a sequential setting as the optimization process starts only after an iteration is complete, which takes a significant time in our case. Our algorithm is proved for $z = 1$, $z = 2$; these cases are not discussed in [Shamir, 2013,Fabian, 1967,Spall, 2009].

Our algorithm is not limited to functions with quadratic approximations; quadratic approximations are a natural framework, but the success of various surrogate models in the recent years suggests that other approximation frameworks could be used. Our theorems are not specific for quadratic approximations and only require that the LSE approximation holds. The LSE assumption is natural in terms of scaling with respect to r - the $1/\sqrt{r}$ typical deviation is usual in e.g. maximum likelihood estimates, and therefore the method should be widely applicable for general surrogate models.

More generally, our results show a fast rate as soon as the estimator of the location of the optimum has squared error $O(\sigma^{2z-2}/r)$, when using r points sampled adequately within distance $O(\sigma)$ of the optimum.

Further work. In the theoretical side, further work includes writing detailed constants, in particular depending on the eigenvalues of the Hessian of the expected objective function at the optimum and the dimension of the search space. In the case of infinite slope (see Table 4.1, $z = 2$), we conjecture that the convergence is log-linear, i.e. the logarithm of the simple regret decreases as a function of the number of evaluations. In the other hand, future study consists of extensive experiments - but we refer to [Cauwet et al., 2014] for significant artificial experiments and [Liu and Teytaud, 2014] for the application which motivated this work.

Part of the agenda is to extend the algorithm by providing other examples of estimators to be used for approximating the location of the optimum (other than Examples 1 and 3, but verifying the LSE assumption); in particular, classical surrogate models, and applications to piecewise linear strongly convex functions as in [Rolet and Teytaud, 2010]. A way to improve the algorithm is to use quasi-Newton estimates of the Hessian, from the successive gradients, rather than using directly finite differences. Last, making algorithms more adaptive by replacing the constants by adaptive parameters depending on noise estimates is under consideration.

5 Resampling in evolution strategies

Log-linear scale and log-log scale: uniform and non-uniform rates. To ensure the convergence of an algorithm and analyze the rate at which it converges are part of the main goals when it comes to the study of optimization algorithms.

In the noise-free case, evolution strategies typically converge linearly in log-linear scale, this is, the logarithm of the distance to the optimum typically scales linearly with the number of evaluations (see Section 5.1.1 for more details on this). The case of noisy fitness values leads to a log-log convergence [Teytaud and Decock, 2013]. We investigate conditions under which such a log-log convergence is possible. In particular, we focus on uniform rates. Uniform means that all points are under a linear curve in the log-log scale. Formally, the rate is the infimum of C such that with probability $1-\delta$, for m sufficiently large, all iterates after m fitness evaluations verify $\log ||x_m|| \leq -C \log m$, where x_m is the m^{th} evaluated individual. This is, all points are supposed to be "good" (i.e. satisfy the inequality); not only the best point of a given iteration. In contrast, a non-uniform rate would be the infimum of C such that $\log ||x_{k_m}|| \leq -C \log k_m$ for some increasing sequence k_m.

The state of the art in this matter exhibits various results. For an objective function with expectation $\mathbb{E}[f(x)] = ||x - x^*||^2$, when the variance is not supposed to decrease in the neighborhood of the optimum, it is known that the best possible slope in this log-log graph is $-\frac{1}{2}$ (see [Fabian, 1967]), but without uniform rate. When optimizing $f(x) = ||x||^p + \mathcal{N}$, this slope is provably limited to $-\frac{1}{p}$ under locality assumption (i.e. when sampling far from the optimum does not help, see [Teytaud and Decock, 2013] for a formalization of this assumption), and it is known that some ad hoc EDA can reach $-\frac{1}{2p}$ (see [Rolet and Teytaud, 2009]).

For evolution strategies, the slope is not known. Also, the optimal rate for $\mathbb{E}[f(x)] = ||x - x^*||^p$ for $p \neq 2$ is unknown; we show that

our evolution strategies with simple revaluation schemes have linear convergence in log-log representation in such a case.

Algorithms considered in this chapter. We here focus on simple revaluation rules, in evolution strategies, based on choosing the number of resamplings. We start with rules which decide the number of revaluations only depending on n. This is, independently of the step-size σ_n, the parents x_n and fitness values. To the best of our knowledge, these simple rules have not been analyzed so far. Nonetheless, they have strong advantages: on one hand, rules based on numbers of resamplings defined as a function of σ_n have a strong sensitivity to parameters, whereas we get a linear slope in log-log curve with simple rules $r_n = K\lceil n^\zeta \rceil$. Also evolution strategies, contrarily to algorithms with good non-uniform rates, have a nice empirical behavior from the point of view of uniform rates, as shown mathematically by [Rolet and Teytaud, 2009].

Overview of the chapter. We get mathematical proofs only with an exponential number of resamplings. Essentially, the algorithms for which we get a proof have the same dynamics as in the noise-free case, they just use enough resamplings for canceling the noise. This is consistent with the existing literature, in particular [Rolet and Teytaud, 2009] which shows a log-log convergence for an Estimation of Distribution Algorithm with exponentially decreasing step-size and exponentially increasing number of resamplings. In our experiments, we see that another solution is a polynomially increasing number of resamplings (independently of σ_n; the number of resamplings just smoothly increases with the number of iterations, in a non-adaptive manner), leading to a slower convergence when considering the progress rate per iteration, but the same log-log convergence when considering the progress rate per evaluation.

After analyzing such non-adaptive rules in the scale invariant case (Section 5.1.2), we study adaptive rules (Section 5.1.3), in which the number of evaluations depend on the step size only; we also get rid of the scale invariant assumption. We then investigate experimentally (Section 5.2) the polynomial case. We could get positive experimental results even with the non-proved polynomial number of revaluations (non-adaptive); maybe those results are the most satisfactory (stable) results. We could also get convergence with adaptive rules (number of resamplings depending on the step-size), however results are seemingly less stable than with non-adaptive methods.

5.1 Theoretical analysis: exponential non-adaptive rules can lead to log/log convergence.

Section 5.1.1 is devoted to some preliminaries. Section 5.1.2 presents results in the scale invariant case, for an exponential number of resamplings and non-adaptive rules. Section 5.1.3 will focus on adaptive rules, with numbers of resamplings depending on the step-size.

5.1.1 Preliminary: noise-free case

In the noise-free case, for some evolution strategies, we know the following results, almost surely (see e.g. Theorem 4 in [Auger, 2005], where, however, the negativity of the constant is checked by Monte-Carlo simulations): $\log(\sigma_n)/n$ converges to some constant $(-A) < 0$ and $\log(||x_n||)/n$ converges to some constant $(-A') < 0$. This implies that for any $\rho < A$, $\log(\sigma_n) \leq -\rho n$ for n sufficiently large. So, $\sup_{n\geq 1} \log(\sigma_n) + \rho n$ is finite. With these almost sure results, now consider V the quantile $1 - \delta/4$ of $\exp\left(\sup_{n\geq 1} \log(\sigma_n) + \rho n\right)$. Then, with probability at least $1 - \delta/4$, $\forall n \geq 1, \sigma_n \leq V \exp(-\rho n)$. We can apply the same trick for lower bounding σ_n, and upper and lower bounding $||x_n||$, all of them with probability $1-\delta/4$, so that all bounds hold true simultaneously with probability at least $1 - \delta$.

Hence, for any $\alpha < A'$, $\alpha' > A'$, $\rho < A$, $\rho' > A$, there exist $C > 0$, $C' > 0$, $V > 0$, $V' > 0$, such that with probability at least $1 - \delta$

$$\forall n \geq 1, \quad C' \exp(-\alpha' n) \leq \quad ||x_n|| \quad \leq C \exp(-\alpha n); \tag{5.1}$$

$$\forall n \geq 1, \quad V' \exp(-\rho' n) \leq \quad \sigma_n \quad \leq V \exp(-\rho n). \tag{5.2}$$

We will first show,in Section 5.1.2, our noisy optimization result (Theorem 2):

1. in the scale invariant case
2. using Equation 5.1 (supposed to hold in the noise-free case)

We will then show similar results in Section 5.1.3:

1. without scale-invariance
2. using Equation 5.2 (supposed to hold in the noise-free case)
3. with other resamplings schemes

5.1.2 Scale invariant case, with exponential number of resamplings

We consider Algorithm 20, a version of multi-membered Evolution Strategies, the (μ,λ)-ES with resamplings. μ denotes the number of parents and λ the number of offspring ($\mu \leq \lambda$). In every generation, the selection takes place among the λ offspring, produced from a population of μ parents. Selection is based on the ranking of the individuals $fitness(x_n)$ taking the μ best individuals. Here x_n denotes the parent at iteration n.

We now state our first theorem, under log-linear convergence assumption (the assumption in Equation 5.4 is just Equation 5.1).

Theorem 2. *Consider the fitness function*

$$f(z) = ||z||^p + \mathcal{N} \tag{5.3}$$

over $\mathbb{R}^d$ *and* $x_1 = (1, 1, \ldots, 1)$.
Consider an evolution strategy with population size λ*, parent population size* μ*, such that without resampling, for any* $\delta > 0$*, for some* $\alpha > 0$*,* $\alpha' > 0$*, with probability* $1 - \delta/2$*, with objective function* $fitness(x) = ||x||$*,*

$$\exists C, C'; \qquad C' \exp(-\alpha' n) \leq ||x_n|| \leq C \exp(-\alpha n). \tag{5.4}$$

Assume, additionally, that there is scale invariance:

$$\sigma_n = C''||x_n|| \tag{5.5}$$

for some $C'' > 0$.
Then, for any $\delta > 0$*, there is* $K_0 > 0, \zeta_0 > 0$ *such that for* $K \geq K_0$*,* $\zeta > \zeta_0$*, Equation 5.1 also holds with probability at least* $1 - \delta$ *for fitness function as in Equation 5.3 and resampling rule*

$$N_{resample} = \lceil K \zeta^n \rceil.$$

Proof. In all the proof, $\mathcal{N}$ denotes a standard Gaussian random variable (depending on the context, in dimension 1 or d). Consider an arbitrary $\delta > 0$. Consider some $n \geq 1$. Consider $\delta_n = \exp(-\gamma n)$ for some $\gamma > 0$.

Define p_n the probability that two generated points, e.g. i_1 and i_2, are such that $\mid ||i_1||^p - ||i_2||^p \mid \leq \delta_n$.

Step 1: Using $i_j = x_{n,mod(j-1,\mu)+1} + \sigma_j \mathcal{N}$ (Equation 17.2) to compute the coordinate j of individual i and Equation 5.5, we show that

$$p_n \leq B' \exp(-\gamma' n) \tag{5.6}$$

for some $B' > 0, \gamma' > 0$ depending on γ, d, p, C', C'', α'.

Proof of step 1: with $\mathcal{N}_1$ and $\mathcal{N}_2$ two d-dimensional independent standard Gaussian random variables,

$$p_n \leq P(\mid ||1 + C''\mathcal{N}_1||^p - ||1 + C''\mathcal{N}_2||^p \mid \leq \delta_n / ||x_n||^p). \tag{5.7}$$

Define $densityMax$ the supremum of the density of $\mid ||1 + C''\mathcal{N}_1||^p - ||1 + C''\mathcal{N}_2||^p \mid$ we get

$$p_n \leq densityMax C'^{-p} \exp((p\alpha' - \gamma)n),$$

hence the expected result with $\gamma' = \gamma - p\alpha'$ and $B' = densityMax(C')^{-p}$. Notice that $densityMax$ is upper bounded.

In particular, γ' is arbitrarily large, provided that γ is sufficiently large.

Step 2: Consider now p'_n the probability that there exists i_1 and i_2 such that $\mid ||i_1||^p - ||i_2||^p \mid \leq \delta_n$. Then, $p'_n \leq \lambda^2 p_n \leq B' \lambda^2 \exp(-\gamma' n)$.

Step 3: Consider now p''_n the probability that $|\mathcal{N}/\sqrt{K\zeta^n}| \geq \delta_n/2$. First, we write $p''_n = P(\mathcal{N} \geq \frac{\delta_n}{2}\sqrt{K\zeta^n})$. So by Chebychev inequality, $p''_n \leq B'' \exp(-\gamma'' n)$ for $\gamma'' = \log(\zeta) - 2\gamma$ arbitrarily large, provided that ζ is large enough, and $B'' = 4/K$.

Step 4: Consider now p'''_n the probability that $|\mathcal{N}/\sqrt{K\zeta^n}| \geq \delta_n/2$ at least once for the λ evaluated individuals of iteration n. Then, $p'''_n \leq \lambda p''_n$.

Step 5: In this step we consider the probability that two individuals are misranked due to noise. Let us now consider p''''_n the probability that at least two points i_a and i_b at iteration n verify

$$||i_a||^p \leq ||i_b||^p \tag{5.8}$$

$$\text{and} \qquad noisyEvaluation(i_a) \geq noisyEvaluation(i_b) \tag{5.9}$$

where $noisyEvaluation(i)$ is the average of the multiple evaluations of individual i. Equations 5.8 and 5.9 occur simultaneously if either two points have very similar fitness (difference less than δ_n) or the noise is big (larger than $\delta_n/2$). Therefore, $p''''_n \leq p'_n + p'''_n \leq \lambda^2 p_n + \lambda p''_n$ so $p''''_n \leq (B' + B'')\lambda^2 \exp(-\min(\gamma', \gamma'')n)$.

Step 6: Step 5 was about the probability that at least two points at iteration n are misranked due to noise. We now consider $\sum_{n \geq 1} p''''_n$,

which is an upper bound on the probability that in at least one iteration there is a misranking of two individuals.

If γ' and γ'' are large enough, $\sum_{n\geq 1} p_n'''' < \delta$.

This implies that with probability at least $1-\delta$, provided that K and ζ have been chosen large enough for γ and γ' to be large enough, we get the same rankings of points as in the noise free case - this proves the expected result. □

Remark 3. *(i) Informally speaking, our theorem shows that if a scale invariant algorithm converges in the noise-free case, then it also converges in the noisy case with the exponential resampling rule, at least if parameters are large enough (a similar effect of constants was pointed out in [Jebalia et al., 2010] in a different setting).*
(ii) We assume that the optimum is in 0 *and the initial* x_1 *at* 1*. Note that these assumptions have no influence when we use algorithms invariant by rotation and translation.*
(iii) We show a log-linear convergence rate as in the noise-free case, but at the cost of more evaluations per iteration. When normalized by the number of function evaluations, we get $\log ||x_n||$ *linear in the logarithm of the number of function evaluations, as detailed in Corollary 2.*

The following corollary shows that this is a log-log convergence.

Corollary 2 (log-log convergence with exponential resampling)**.** *With* e_n *the number of evaluations at the end of iteration* n*, we have* $e_n = K\zeta\frac{\zeta^n-1}{\zeta-1}$*. We then get, from Equation 5.1,*

$$\log(||x_n||)/\log(e_n) \to -\frac{\alpha}{\log\zeta} \tag{5.10}$$

with probability at least $1-\delta$*.*

Equation 5.10 is the convergence in log/log scale.

We have shown this property for an exponentially increasing number of resamplings, which is indeed similar to R-EDA [Rolet and Teytaud, 2009], which converges with a small number of iterations but with exponentially many resamplings per iteration. In the experimental section 5.2, we will check what happens in the polynomial case.

5.1.3 Extension: adaptive resamplings and removing the scale invariance assumption

We have assumed above a scale invariance. This is obviously not a nice feature of our proof, because scale invariance does not correspond to anything real; in a real setting we do not know the distance to the optimum. We show below an extension of the result above using the assumption of a log-linear convergence of σ_n as in Equation 5.2.

In the corollary below, we also get rid of the non-adaptive rule with exponential number of resamplings, replaced by a number of resamplings depending on the step-size σ_n only, as in the following equation:

$$R_{sample} = \lceil Y {\sigma_n}^{-\eta} \rceil.$$

In one corollary, we switch to both (i) adaptive resampling rule and (ii) no scale invariance; each change can indeed be proved independently of the other.

Corollary 3 (Adaptive resampling and no scale-invariance)**.** *The proof also holds without scale invariance, under the following assumptions:*

- *For any $\delta > 0$, there are constants $\rho > 0, V > 0, \rho' > 0, V' > 0$ such that with probability at least $1 - \delta$, Equation 5.2 holds.*
- *The number of revaluations is*

$$Y \left(\frac{1}{\sigma_n} \right)^{\eta} \tag{5.11}$$

  with Y and η sufficiently large.

- *Individuals are still randomly drawn using $x_n + \sigma_n \mathcal{N}$ for some random variable $\mathcal{N}$ with bounded density.*

Proof. Two steps of the proof are different, namely step 1 and step 2. We here adapt the proofs of these two steps.

Adapting step 1: Equation 5.7 becomes Equation 5.12:

$$p_n \leq P(\mid ||1 + C''_n \mathcal{N}_1||^p - ||1 + C''_n \mathcal{N}_2||^p \mid \leq \delta_n / ||x_n||^p). \tag{5.12}$$

where $C''_n = \sigma_n / ||x_n|| \geq t' \exp(-tn)$ for some $t > 0, t' > 0$ depending on ρ, ρ', V, V' only. Equation 5.12 leads to

$$p_n \leq (C''_n)^{-d} densityMax C'^{-p} \exp((p\alpha' - \gamma)n),$$

hence the expected result with $\gamma' = \gamma - p\alpha' - dt$. $densityMax$ is upper bounded due to the third condition of Corollary 3.

Adapting step 2: It is sufficient to show that the number of resamplings is larger (for each iteration) than in the Theorem 2, so that step 2 still holds.

Equation 5.11 implies that the number of revaluations at step n is at least $Y\left(\frac{1}{V}\right)^\eta \exp(\rho\eta n)$. This is more than $K\zeta^n$, at least if Y and η are large enough. This leads to the same conclusion as in the Theorem 2, except that we have probability $1-2\delta$ instead of $1-\delta$ (which is not a big issue as we can do the same with $\delta/2$). □

Remark 4. *The last remark is here for cases like self-adaptive algorithms, in which we do not use directly a Gaussian random variable, but a Gaussian random variable multiplied e.g. by* $\exp(\frac{1}{\sqrt{d}})Gaussian$*, with Gaussian a standard Gaussian random variable. For example, SA-ES algorithms as in [Auger, 2005] are included in this proof because they converge log-linearly as explained in Section 5.1.1.*

The following corollary is here for showing that our result leads to the log-log convergence.

Corollary 4 (log-log convergence for adaptive resampling)**.** *With* e_n *the number of evaluations at the end of iteration* n*, we have* $e_n = Y\left(\frac{1}{V}\right)^\eta \exp(\rho\eta)\frac{\exp(\rho\eta n)-1}{\exp(\rho\eta)-1}$*. We then get, from Equation 5.1,*

$$\log(||x_n||)/\log(e_n) \to -\frac{\alpha}{\rho\eta} \tag{5.13}$$

with probability at least $1-\delta$*.*

Equation 5.13 is the convergence in log/log scale.

5.2 Polynomial number of resamplings: experiments

We here consider a polynomial number of resamplings, as in Algorithm 20. Experiments are performed in a "real" setting, without scale invariance. Importantly, our mathematical results hold only log-log convergence under the assumption that constants are large enough. We present results with fitness function $fitness(x) = ||x||^p + \mathcal{N}$ with $p = 2$ in Figure 5.1.

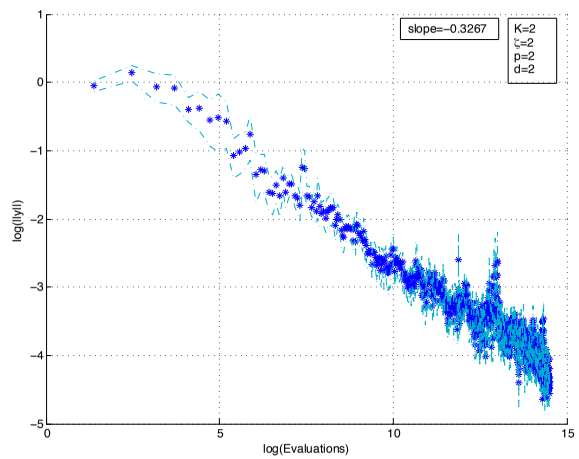
We get slopes close to $-\frac{1}{2p}$, with $\zeta = 1$ or $\zeta = 2$. Non-presented experiments show that $\zeta = 0$ performs very poorly; other experiments with $p = 1$ or $p = 4$ and dimension $2, 3, 4, 5$, with $\zeta = 1, 2, 3$ and with $\mu = \min(d, \lceil \lambda/4 \rceil)$, $\lambda = \lceil d\sqrt{d} \rceil$ as recommended in [Beyer, 2001, Fournier and Teytaud, 2011]; slopes are usually better than $-1/(2p)$ for $\zeta = 2$ or $\zeta = 3$ and worse for $\zeta = 1$; seemingly results for ζ large are farther from the asymptotic regime. We conjecture that the asymptotic regime is $-1/(2p)$ but that it is reached later when ζ is large. R-EDA [Rolet and Teytaud, 2009] reaches $-\frac{1}{2p}$; we seemingly get slightly better but this might be due to a non-asymptotic effect. Figure 5.1 provides results with high numbers of evaluations.

5.3 Experiments with adaptivity: $Y\sigma_n^{-\eta}$ revaluations

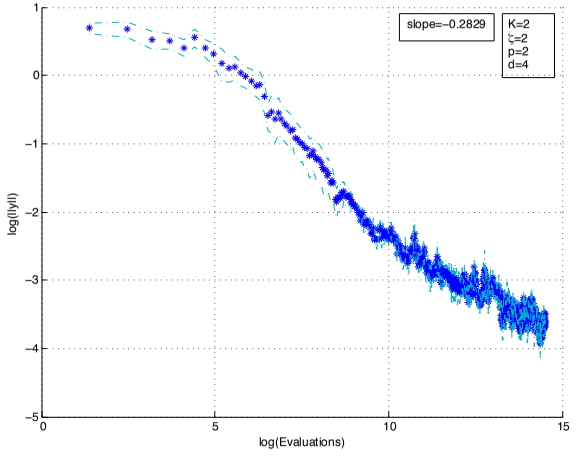
We here experiment Algorithm 20. The algorithm should converge linearly in log-log scale as shown by Corollary 4, at least for large enough values of Y and η. Notice we consider values of μ, λ for which log-linear convergence is proved in the noise-free setting (see Section 5.1.1). In all this section, $\mu = \min(d, \lceil \lambda/4 \rceil)$, $\lambda = \lceil d\sqrt{d} \rceil$.

Slopes as estimated on the case $\eta = 2$ (usually the most favorable, and an important case naturally arising in sufficiently differentiable problems) are given in Table 5.1 (left) for dimension $d = 100$. In this case we are far from the asymptotic regime. We get results close to $-\frac{1}{2}$ in all cases. $-\frac{1}{2}$ is reachable by algorithms which learn a model of the fitness function, as e.g. [Coulom, 2012]. A point is that we have high dimension in this case and maybe the $-\frac{1}{2p}$ is for asymptotic results, and in high dimension we are far from asymptotic results. This is suggested by experiments in dimension 10, summarized in Table 5.1 (right). We also point out that the known complexity bounds is $-\frac{1}{p}$ (from [Teytaud and Decock, 2013]), so maybe the slope can reach $-\frac{1}{p}$ in some cases.

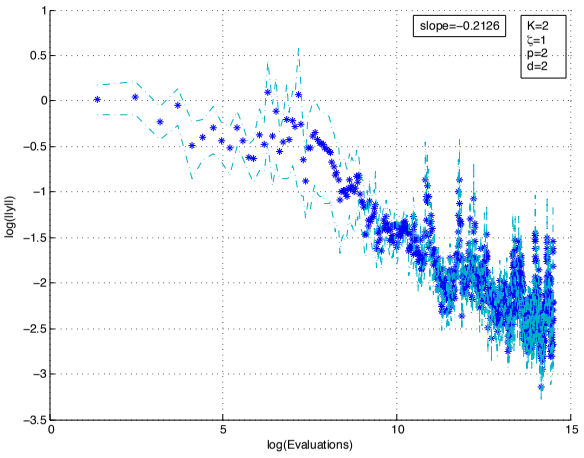
Results with $Y\left(\frac{1}{\sigma}\right)^\eta$ are moderately stable (impact of Y, in particular), hence our preference for stable rules such as non-adaptively choosing n^2 revaluations per individual at iteration n. Slopes as estimated on the case $\eta = 2$ are given in Table 5.1 (right) for dimension $d = 10$; we also tested $20(\frac{1}{\sigma_n})^2$ (i.e. $Y = 20$). Results with $\left(\frac{1}{\sigma_n}\right)^\eta$ are moderately stable, hence our preference for stable rules such as non-adaptively choosing n^2 revaluations per individual at iteration n.



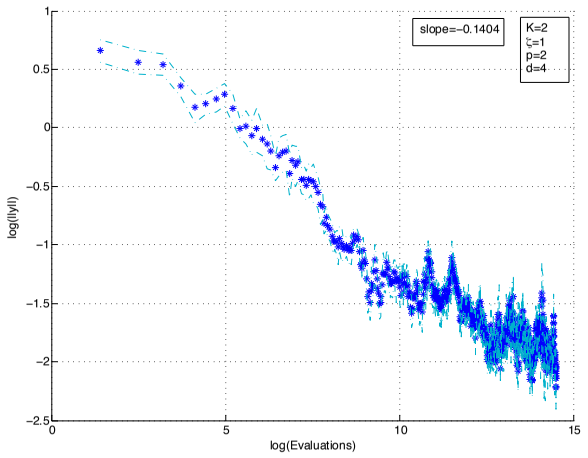
(a) $d = 2, \zeta = 2. Slope = -0.3267$.



(b) $d = 4, \zeta = 2. Slope = -0.2829$.



(c) $d = 2, \zeta = 1. Slope = -0.2126$.



(d) $d = 4, \zeta = 1. Slope = -0.1404$.

Figure 5.1: Experiments in dimension 2, 3, 4 with $\zeta = 1, 2$ (number of evaluations shown by x-axis) for $r_n = K\lceil n^\zeta \rceil$ (i.e. polynomial, non-adaptive) with $\mu = 2$, $\lambda = 4$, $p = 2$ and $K = 2$. The slope is evaluated on the second half of the iterations. We get slopes close to $-1/(2p)$. All results are averaged over 20 runs.

Table 5.1: **Left: Dimension 100.** Estimated slope for the adaptive rule with $r_n = \lceil \left(\frac{1}{\sigma_n}\right)^2 \rceil$ resamplings at iteration n. Slopes are estimated on the second half of the curve. **Right: Dimension 10.** Estimated slope for the adaptive rule with $r_n = \lceil Y \left(\frac{1}{\sigma_n}\right)^2 \rceil$ resamplings at iteration n ($Y = 1$ as in previous curves, and $Y = 20$ for checking the impact of convergence; the negative slope (apparent convergence) for $Y = 20$ is stable, as well as the divergence or stagnation for $Y = 1$ for $p = 4$). Slopes are estimated on the second half of the curve.

$d = 100$	
p	slope for $Y = 1$
1	-0.52
2	-0.51
4	-0.45

$d = 10$		
p	slope for $Y = 1$	slope for $Y = 20$
1	-0.51	-0.50
2	-0.18	-0.17
4	>0	-0.08

5.4 Comparison of resampling rules for Evolutionary Noisy Optimization on a Unimodal Function

We compare 8 resampling rules that can be separated into 3 families. The first family is polynomial. It includes a linear, quadratic and cubic resampling rules. The second family is exponential. It contains 2 simple exponential resampling rules. The third family consists of 3 new exponential resampling rules, which vary with both the current generation number n and the dimension d as mentioned in Section 5.4.4. All studied functions are given in Table 5.2.

Table 5.2: Resampling rules and sub-index in figures. d: dimension of search domain; n: current generation number.

Notation	*linear*	*quad.*	*cubic*	2*exp*	*exp*/10
r_n	n	n^2	n^3	2^n	$\lceil exp(\frac{n}{10}) \rceil$
Sub-index	(a)	(b)	(c)	(d)	(e)

Notation	*math*1	*math*2	*math*3
r_n	$\lceil exp(\frac{4n}{5d})/d^2 \rceil$	$\lceil exp(\frac{n}{10})/d^2 \rceil$	$\lceil exp(\frac{n}{\sqrt{d}})/d^2 \rceil$
Sub-index	(f)	(g)	(h)

5.4.1 Experimental setup

We consider a noisy sphere function $fitness(x) = ||x - x^*||^2 + \mathcal{N}$, where $\mathcal{N}$ denotes some independent standard Gaussian random variable. x and x^* are points in search space $\mathbb{R}^d$ and x^* is the problem optimum. Without loss of generality, we assume $x^* = 0$. Thus, $fitness(x) = ||x||^2 + \mathcal{N}$ and $\mathbb{E}fitness(x^*) = 0$.

Thanks to $\mathbb{E}fitness(x^*) = 0$, Equation 2.10, i.e. the simple regret after m evaluations, is:

$$SR = \mathbb{E}fitness(x_m). \tag{5.14}$$

We will investigate if some non-adaptive rules for choosing resampling numbers lead to

$$slope(SR) \sim A''' < 0, \tag{5.15}$$

and which resampling rules lead to the best A''' in Equation 5.15.

For SA-(μ/μ,λ)-ES, we consider two choices of offspring size [Beyer, 2001]: (i) $\lambda = 4 + 4d$; (ii) $\lambda = \lceil d\sqrt{d} \rceil$ and three choices of parent size [Fournier and Teytaud, 2011]: (i) $\mu = \min(d, \lceil \lambda/4 \rceil)$; (ii) $\mu = \min(2d, \lceil \lambda/4 \rceil)$; (iii) $\mu = 1$, termed as (1,λ)-SAES. In all figures, the y-axis is the logarithm of the Simple Regret and the x-axis is the logarithm of number of evaluations, where $fitness(x) = ||x||^2 + \mathcal{N}$, the initial $||x_{initial}|| = 1$ and the initial step-size is $\sigma_{initial} = 1$. In order to check what happens in the long run, we use a large budget $T = 5e12$ function evaluations. All experiments are repeated 10 times.

Section 5.4.2 presents the results for linear, quadratic and cubic resampling rules. Section 5.4.3 shows the results for 2 simple exponential resampling rules, which only depend on the current generation number n. Section 5.4.4 focuses on 3 new exponential resampling rules, which vary with both the current generation number n and the dimension d, whereas the functions presented in Section 5.4.2 and 5.4.3 vary solely on the current generation number n. All functions are given in Table 5.2.

Figs. 5.2, 5.3, 5.4 and 5.5 present results for dimension 2, 10, 100 and 500.

5.4.2 Experiments with polynomial number of resamplings

Figs. 5.2(a), 5.3(a) and 5.4(a) show the 6 different combinations of μ and λ defined previously when the resampling number is linear $r_n =$

n. Figs. 5.2(b), 5.3(b) and 5.4(b) present the situation where the resampling number is quadratic $r_n = n^2$ whereas Figs. 5.2(c), 5.3(c) and 5.4(c) depict a cubic resampling function $r_n = n^3$. In these 3 cases, n represents the current generation number.

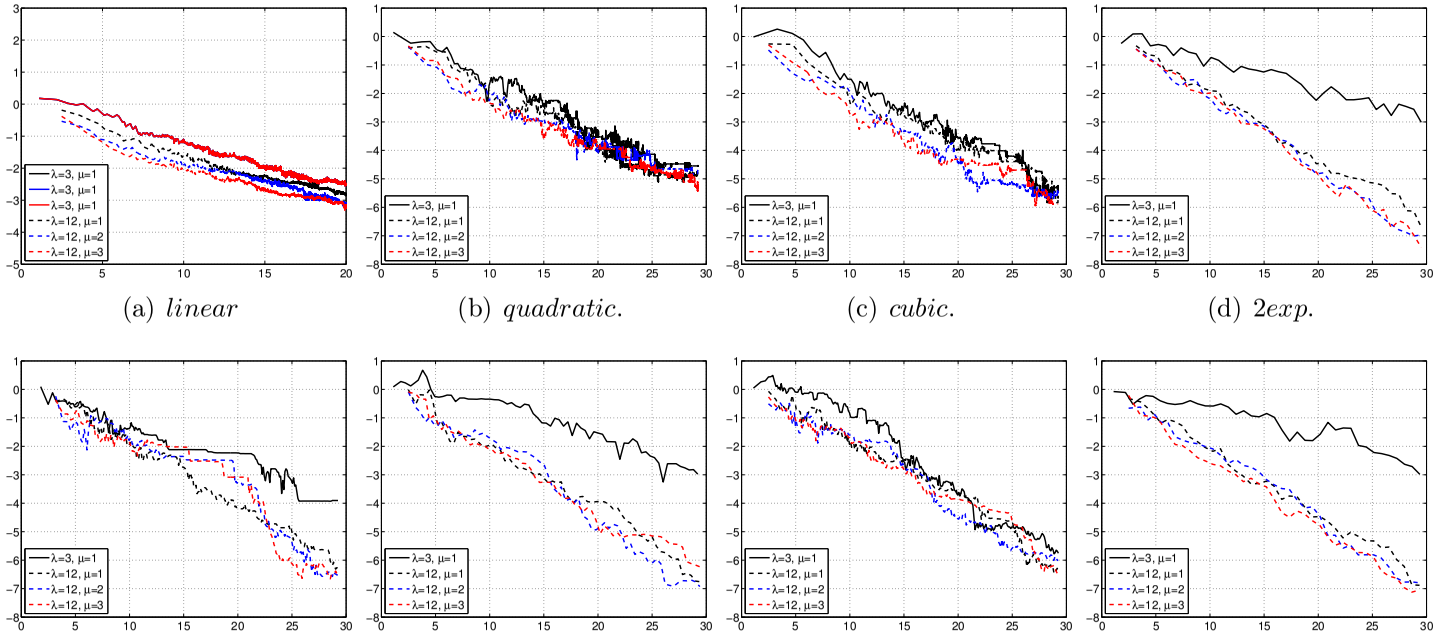
Regardless of the dimension and unsurprisingly, $\mu > 1$ yields better results. For each dimension $d = 2$ (Figure 5.2), $d = 10$ (Figure 5.3), $d = 100$ (Figure 5.4), $\mu = \min(d, \lceil \lambda/4 \rceil)$ (blue curves) and $\mu = \min(2d, \lceil \lambda/4 \rceil)$ (red curves) provide better performance than using $\mu = 1$ parent (black curves) [Arnold and Beyer, 2000]. Such results are classical in noise-free optimization and are confirmed here for polynomial resamplings in a noisy setting. The choice of λ, among these two natural choices, does not seem to have a significant influence on the outcome for dimension $d = 10$, 100 and provides a minor improvement for $d = 2$.

5.4.3 Experiments with exponential number of resamplings

Figs. 5.2(d), 5.3(d) and 5.4(d) show the 6 different combinations of μ and λ defined previously for the resampling function $2exp$. As a reminder, this function is given by $r_n = 2^n$. Figs. 5.2(e), 5.3(e) and 5.4(e) present the results for the resampling rule $exp/10$ ($r_n = \lceil exp(\frac{n}{10}) \rceil$). n represents the current generation number.

Figure 5.5 show similar behaviors. We observe that $\mu = \min(d, \lceil \lambda/4 \rceil)$ (blue) and $\mu = \min(2d, \lceil \lambda/4 \rceil)$ (red) yield better results than $\mu = 1$ which is in line with the state of the art in noise-free optimization and confirms findings in Section 5.4.2, here in the case of exponential resampling rules and noisy optimization.

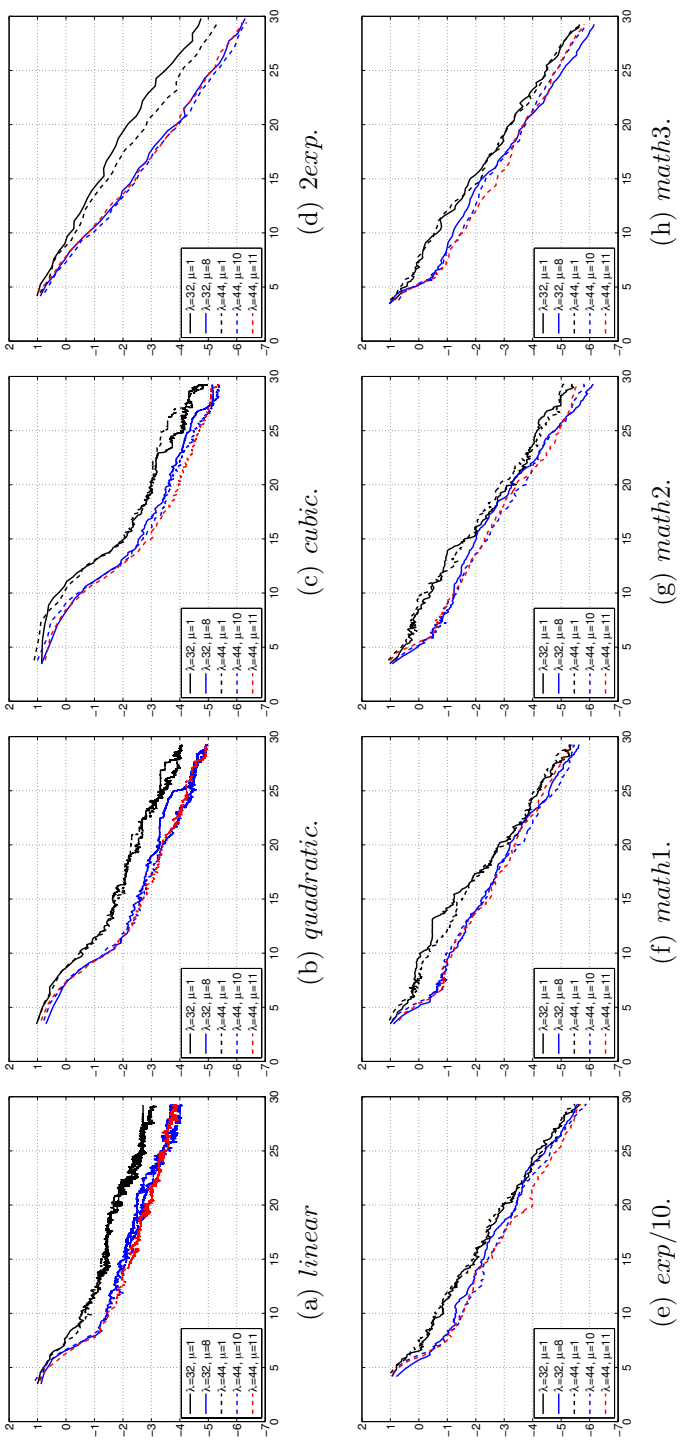
$exp/10$ and $2exp$ seem particularly suited for small dimension $d = 2, 10$, but $2exp$ does not yield good results in dimension $d = 100$. $exp/10$ is more constant in its performance with regard to the dimension. For the 4 exponential resampling functions, we conclude that μ is a significant parameter to tune and $\mu = \min(d, \lceil \lambda/4 \rceil)$ and $\mu = \min(2d, \lceil \lambda/4 \rceil)$ yield the best results. In dimension $d = 2$, the best exponential resampling function is $exp/10$.



(a) *linear* (b) *quadratic.* (c) *cubic.* (d) 2*exp.*

(e) *exp*/10. (f) *math*1. (g) *math*2. (h) *math*3.

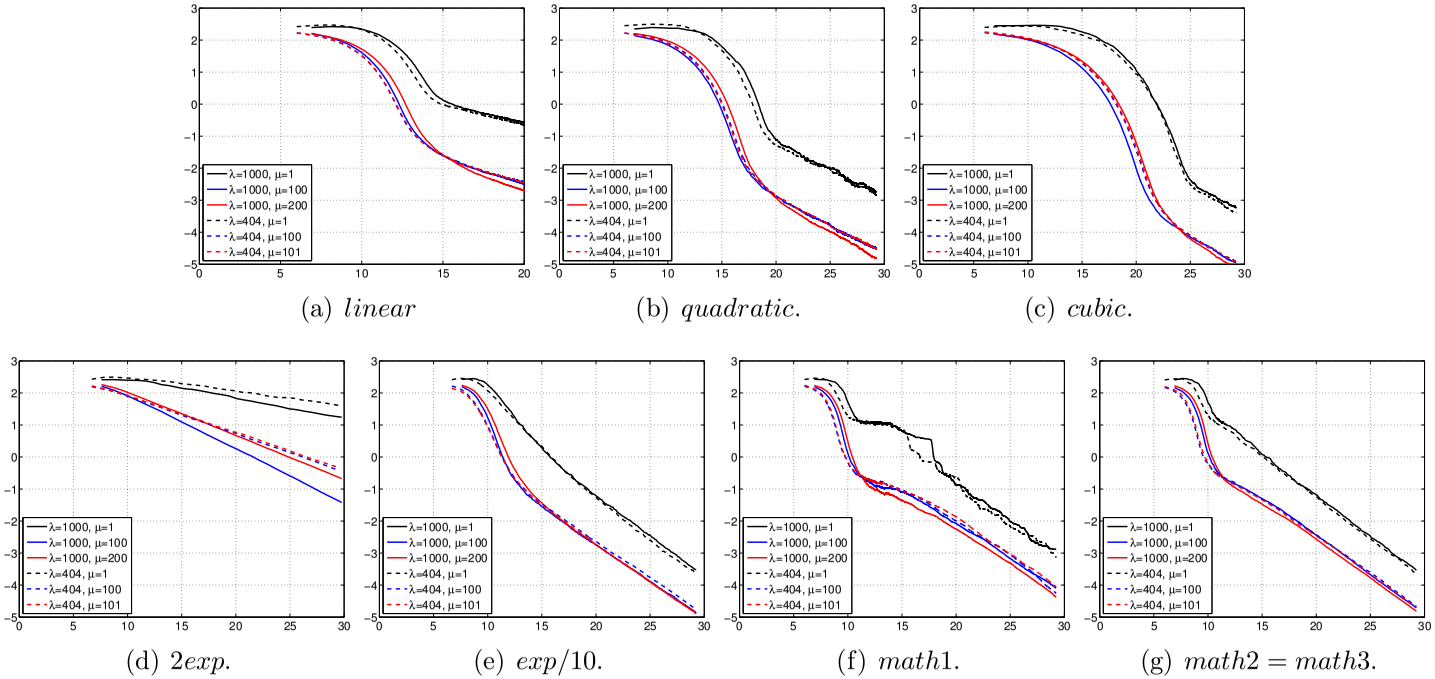
Figure 5.2: Results for dimension $d = 2$ with linear, quadratic, cubic and exponential resampling numbers. Logarithm of Simple Regret (y-axis) with respect to the logarithm of the number of evaluations (x-axis), where $fitness(x) = ||x||^2 + \mathcal{N}$, $||x_{initial}|| = 1$ and $\sigma_{initial} = 1$. Solid curves are the results for $\lambda = \lceil d\sqrt{d} \rceil = 3$, dashed curves are the results for $\lambda = 4 + 4 \times d = 12$. Here, the number of parents is set as $\mu = 1$ (black), $\mu = \min(d, \lceil \lambda/4 \rceil)$ (blue) and $\mu = \min(2d, \lceil \lambda/4 \rceil)$ (red). All exponential formulas perform well, including the *math*2 formula which performs best.



(a) *linear* (b) *quadratic.* (c) *cubic.* (d) $2exp.$

(e) $exp/10.$ (f) $math1.$ (g) $math2.$ (h) $math3.$

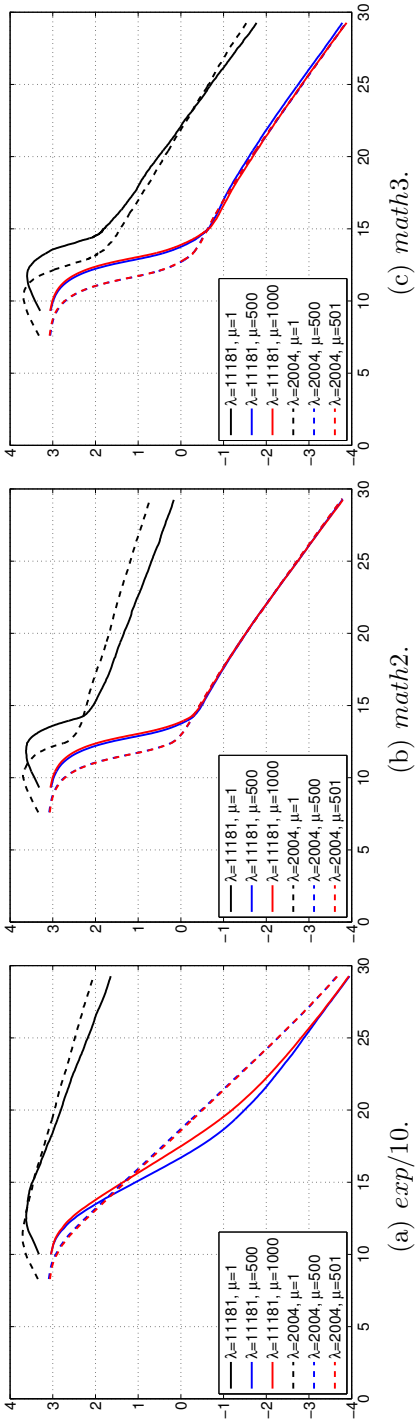
Figure 5.3: Results for dimension $d = 10$ with linear, quadratic, cubic and exponential resampling numbers. Logarithm of Simple Regret (y-axis, $[-7, 2]$) with respect to the logarithm of the number of evaluations (x-axis, $[0, 30]$), where $fitness(x) = ||x||^2 + \mathcal{N}$, $||x_{initial}|| = 1$ and $\sigma_{initial} = 1$. Solid curves are the results for $\lambda = \lceil d\sqrt{d} \rceil = 32$, dashed curves are the results for $\lambda = 4 + 4 \times d = 44$. Here, the number of parents is set as $\mu = 1$ (black), $\mu = \min(d, \lceil \lambda/4 \rceil)$ (blue) and $\mu = \min(2d, \lceil \lambda/4 \rceil)$ (red). All exponentially formulas perform well.



(a) *linear* (b) *quadratic.* (c) *cubic.*

(d) $2exp.$ (e) $exp/10.$ (f) $math1.$ (g) $math2 = math3.$

Figure 5.4: Results for dimension $d = 100$ with linear, quadratic, cubic and exponential resampling numbers. Logarithm of Simple Regret (y-axis, $[-5, 3]$) with respect to the logarithm of the number of evaluations (x-axis, $[0, 30]$), where $fitness(x) = ||x||^2 + \mathcal{N}$, $||x_{initial}|| = 1$ and $\sigma_{initial} = 1$. Solid curves are the results for $\lambda = \lceil d\sqrt{d} \rceil = 1000$, dashed curves are the results for $\lambda = 4 + 4 \times d = 404$. Here, the number of parents is set as $\mu = 1$ (black), $\mu = \min(d, \lceil \lambda/4 \rceil)$ (blue) and $\mu = \min(2d, \lceil \lambda/4 \rceil)$ (red). Many functions performed well; actually, squared, cubic and all exponential functions are satisfactory.



(a) *exp*/10. (b) *math*2. (c) *math*3.

Figure 5.5: Results for dimension $d = 500$ with exponential resampling numbers. Logarithm of Simple Regret (y-axis, $[-4, 4]$) with respect to the logarithm of the number of evaluations (x-axis, $[0, 30]$), where $fitness(x) = ||x||^2 + \mathcal{N}$, $||x_{initial}|| = 1$ and $\sigma_{initial} = 1$. Solid curves are the results for $\lambda = \lceil d\sqrt{d} \rceil = 11181$, dashed curves are the results for $\lambda = 4 + 4 \times d = 2004$. Here, the number of parents is set as $\mu = 1$ (black), $\mu = \min(d, \lceil \lambda/4 \rceil)$ (blue) and $\mu = \min(2d, \lceil \lambda/4 \rceil)$ (red). *math*2 and *math*3 perform well; combined with other curves, we conclude that *math*2 is a stable formula for choosing the number of resamplings.

5.4.4 Experiments with resampling number depending on d and n

In this section we discuss the performance of a resampling function generated from empirical observations. Figs. 5.2(f), 5.3(f) and 5.4(f) show the 6 different combinations of μ and λ defined previously when the number of resamplings is $\lceil exp(\frac{4n}{5d})/d^2 \rceil$, noted $math1$ in Table 5.2. Figs. 5.2(g), 5.3(g) and 5.4(g) show the results when the number of resamplings is $\lceil exp(\frac{n}{10})/d^2 \rceil$, noted $math2$ in Table 5.2. Figs. 5.2(h), 5.3(h) and 5.4(h) show the results when the number of resamplings is $\lceil exp(\frac{n}{\sqrt{d}})/d^2 \rceil$, noted $math3$ in Table 5.2.

First, we observe that in Figs. 5.2(f-h), 5.3(f-h) and 5.4(f-h) the results are more constant throughout the different dimensions. As it was concluded in previous sections, μ is a significant parameter to tune and $\mu = \min(d, \lceil \lambda/4 \rceil)$ (blue curves) and $\mu = \min(2d, \lceil \lambda/4 \rceil)$ (red curves) yield the best results. This is, as in previous sections, consistent with the state of the art in noise-free cases. The classical rules for choosing μ are validated in a noisy setting.

5.4.5 Discussion and conclusion: choosing a non-adaptive resampling rule

We compare 3 polynomial and 5 exponential resampling rules presented above.

For **s**mall dimension, (i) $math3$ converges fastest when λ is large; (ii) $r_n = n$ ($linear$) is the slowest; (iii) $r_n = n^3$ ($cubic$) and $math2$ are fast for all the values of λ and μ; (iv) $r_n = \lceil 2^n \rceil$ ($2exp$), $math1$ and $math3$ converge faster than $r_n = n^3$ ($cubic$) when λ is large. For **m**oderate dimension, (i) exponential numbers of resamplings have similar performance for all the values of λ and μ, but $math3$ has $slope(SR)$ slightly faster than others'; (ii) the choices of λ and μ do not change a lot the results for dimension 10. For **l**arge dimension, (i) $r_n = \lceil exp(\frac{n}{10}) \rceil$ ($exp/10$) converges the most slowly when $\mu = 1$ and converges slowly at the beginning when $\mu > 1$; (ii) $r_n = n^3$ ($cubic$) does not converge at the beginning; (iii) $math3$ has faster $slope(SR)$ for all values of λ and μ; $math2$ is almost as efficient.

In high dimension, a "S" shape in the curve becomes obvious. At the beginning of the optimization, the curve converges slowly, because the r_n is not yet relevant (regime 1). Then the convergence becomes

fast due to the good signal/noise ratio (regime 2). Then, the algorithm reaches is asymptotic regime (regime 3). $math2$ has a short slow initial regime (regime 1) and a good asymptotic rate (regime 3), compared to other rules.

Conclusion.

In all cases, the $math2$ formula, $r_n = \lceil exp(\frac{n}{10})/d^2 \rceil$ evaluations for each individual at the n^{th} generation, performs nearly optimally among our non-adaptive rules. The scaling with d is seemingly correct for saving up function evaluations: $2exp$ performs badly in large dimension and polynomial functions perform badly in small dimension. It seems that $math2$ has the best of both worlds in the considered setting (noise standard deviation of the same order as fitness values). We do not claim that this conclusion holds in more general cases compared to adaptive rules or different noise models; we just propose a conclusive answer for the simple case under work.

Limitations.

This work is restricted to non-adaptive rules. Such rules have robustness advantages: (i) we do not need bounds on fitness values (whereas Bernstein methods do), (ii) we have no problem with equal expected fitness values (whereas Bernstein rules can fall in infinite loops when expected values are equal or rules based on empirical standard deviations have such problems [Heidrich-Meisner and Igel, 2009]), (iii) no problem with step-size stagnation as in resampling rules based on the step-size [Astete-Morales et al., 2013]. But the results (both theoretical and experimental) are not relevant for easy cases, in which the noise standard deviation is very small.

Further work.

Adaptive rules have their weaknesses, as they are sensitive to parameters and special cases. However, they can save up fitness evaluations. A natural further work is to use a combination of non-adaptive and adaptive rules:

- Adaptive condition: If a rigorous statistical test concludes that there is no point in keeping resampling, we can stop.

- Non-Adaptive limit: Never apply more than the non-adaptive rule, which is conservative.

Such a combination is visible in [Heidrich-Meisner and Igel, 2009]. The non-adaptive part might benefit from the scaling proposed in our rule $math2$.

Another possible further work is a different point of view, between adaptive methods (using Bernstein races or resampling numbers depending on step-sizes) and non-adaptive methods (as those studied in this chapter). Results here suggest that our exponential formulas are asymptotically good. However, both the mathematical derivation and the experiments are based on the fact that the standard deviation of the noise is of the right order. The asymptotic behavior was sometimes reached very late. Maybe an exponential rule such as $math2$ or $math3$ but with adaptive constants make sense: keeping the scaling with n demonstrated in this chapter, but adapting the parameters, in particular during early stages of the run. Instead of using, as proposed above, the minimum between the number of resamplings proposed by the adaptive rule and the number of resamplings proposed by the non-adaptive rule, we might introduce adaptivity in the parameters of the $math2$ formula. As a further work, we propose comparisons and combinations between these rules and adaptive rules such as [Hansen et al., 2009, Heidrich-Meisner and Igel, 2009].

5.5 Conclusion: which resampling number in noisy evolution strategies ?

Resampling is a simple tool for ensuring convergence in a noisy optimization problem; we have proposed resampling rules and checked/compared their performance.

5.5.1 What is a good noisy optimization algorithm?

There are subtle issues in noisy optimization. Whereas evolutionary algorithms usually just consider individuals, some algorithms distinguish *exploration* individuals, at which the noisy fitness function is applied, and *recommendation* individuals, which are not necessarily evaluated and which are current approximations of the optimum. What is guaranteed or expected is that recommended individuals are good, and ex-

ploration individuals can be very weak. It is known (see [Coulom, 2012] for a clean discussion of that, see also [Fabian, 1967]) that, in settings in which supervised learning of the objective function is efficient (e.g. when the objective function is assumed to belong to a given parametric family of fitness functions), the best algorithms (from the point of view of the recommendation) are very weak in the sense that they use very bad individuals for exploration. For some researchers, this is a bad property because it means that the algorithm is not robust. A more subtle issue is that sometimes, we do not want one good individual; we want hundreds of individuals, for computing statistics on a big sample of points. This is in particular usual in noisy optimization, where risk criteria are often crucial and where heavy simulations are often used for estimating many side parameters. This is why criteria emphasizing convergence of all points and not only recommended individuals, like cumulative regret or the locality assumption, make sense.

Evolution strategies have good properties in terms of cumulative regret, as [Rolet and Teytaud, 2009] has shown that the cumulative regret is $O(\sqrt{n})$, which is known optimal since [Chen, 1988, Shamir, 2013].

5.5.2 Overview of our results

We have shown mathematically log-log convergence results and studied experimentally the slope in this convergence. These results were shown for evolution strategies, which are known for having good uniform rates, rather than good simple regret. We summarize these two parts below and give some research directions.

Log-log convergence. We have shown that the log-log convergence (i.e. linear convergence with x-axis the log of the number of evaluations and y-axis the log of the distance to the optimum) occurs in various cases:

- non-adaptive rules, with number of resamplings exponential in the iteration counter (here we have a mathematical proof); as shown by Corollary 3, this can be extended to non scale-invariant algorithms;
- adaptive rules, with number of resamplings polynomial in $1/\sigma_n$ with σ_n the step-size (here we have a mathematical proof; however, there is a strong sensitivity to constants Y and η which participate in the number of resamplings per individual, $Y\left(\frac{1}{\sigma_n}\right)^{\eta}$);

- non-adaptive rule, with polynomial number of resamplings; this case is a quite convenient scheme experimentally but we have no proof in this case.

Slope in log-log convergence. Experimentally, the best slope in the log-log representation is often close to $-\frac{1}{2p}$ for fitness function $||x||^p + \mathcal{N}$. It is known that under modeling assumptions (i.e. the function is regular enough for being optimized by learning), it is possible to do better than that (the slope becomes $-1/2$ for parametric cases, see [Coulom, 2012] and references therein), but $-\frac{1}{2p}$ is the best known exponent under locality assumption. Basically, locality assumption ensures that most points are reasonably good, whereas some specialized noisy optimization algorithms sample a few very good points and essentially sample individuals far from the optimum (see e.g. [Coulom, 2012]).

6 Resampling in differential evolution

This chapter is devoted to noisy optimization in case of a noise with standard deviation as large as variations of the fitness values, specifically when the variance does not decrease to zero around the optimum. We focus on comparing methods for choosing the number of resamplings. Experiments are performed on the differential evolution algorithm. By mathematical analysis, we design a new rule for choosing the number of resamplings for noisy optimization, as a function of the dimension, and validate its efficiency compared to existing heuristics.

6.1 Differential evolution & noisy differential evolution

6.1.1 Differential evolution

The Differential Evolution (DE) algorithm [Storn and Price, 1997] is an optimization algorithm which operates in continuous search spaces. DE belongs to the family of Evolutionary Algorithms (EAs). It does not need the fitness function to be differentiable. It is based on the four main steps of EAs : initialization, mutation, recombination and selection. This process is iterated until an acceptable solution is found or until a certain time limit is reached. We use Differential Evolution as described in Algorithm 4. D is the dimension of the search space.

Many variants exist, including self-adaptive parameters [Brest et al., 2006, Liu and Lampinen, 2005, Yang et al., 2010, Price et al., 2006] and meta-heuristics for choosing parameters [Pedersen, 2010]. The mutation step is a crucial point and several types exist. Most well known are:

- DE/best/1 [Storn, 1996] : $p'_i = p_{best} + F(p_b - p_c)$;
- DE/best/2 [Storn, 1996] : $p'_i = p_{best} + F(p_b - p_c) + F(p_d - p_e)$;

Algorithm 4 DE/rand/2: Pseudo-code of Differential Evolution. For $j \in \{1, \ldots, D\}$, $(x)_j$ denotes the j^{th} coordinate of a vector $x \in \mathbb{R}^D$.

Input: $F \in [0, 2]$: Differential weight
Input: $Cr \in [0, 1]$: Crossover probability
Input: λ: Population size
Initialize $p_1, \ldots, p_\lambda$ uniformly in the bounded search space
while not finished **do**
  for $i \in \{1, \ldots, \lambda\}$ **do**
    Randomly draw a, b, c, d and e distinct in $\{1, \ldots, i-1, i+1, \ldots, \lambda\}$
    Define

$$p'_i = p_a + F(p_b - p_c) + F(p_d - p_e) \tag{6.1}$$

    Randomly draw $R \in \{1, \ldots, D\}$
    for $j \in \{1, \ldots, D\}$, **do**
      if $rand < Cr$ or $j == R$ **then**
        $(p''_i)_j = (p'_i)_j$
      else
        $(p''_i)_j = (p_i)_j$
      end if
    end for
    $p_i = best(p_i, p''_i)$ (keep p_i in case of tie)
  end for
end while

- DE/rand/1 [Storn, 1996] : $p'_i = p_a + F(p_b - p_c)$;

- DE/rand/2 (Equation 6.1);

where p_{best} is the best point in the current population, p_a, p_b, p_c, p_d and p_e are distinct points randomly chosen in the current population. In this chapter we focus on "DE/rand/2". We use $\lambda = 100$ particles, $F = 0.7$ and $Cr = 0.5$.

6.1.2 Noisy differential evolution: state of the art

DE is simple, handles ill conditioning correctly, and spends very little time in internal computation. However, the performance of DE is unstable when the fitness function is corrupted by noise [Krink et al., 2004].

Resamplings and threshold can be used to deal with noise. [Das et al., 2005] studied an improved DE (DE/rand/1) algorithm where the scalar factor used to weigh the difference vector has been randomized, called Differential Evolution with Random Scale Factor (DE-RSF). A threshold based Selection Strategy, aimed at overcoming the noise through resampling, has been combined into the DE-RSF, namely DE-RSF-TS. Another variant is DE-RSF with Stochastic Selection, namely DE-RSF-SS, with which the offspring is selected as the parent of next generation with probability calculated by the ratio of the fitness of parent to the one of the offspring. [Das et al., 2005] showed that both DE-RSF-TS and DE-RSF-SS performed worse than DE for noise-free functions. But in noisy cases, these two improved algorithms (i) can more efficiently find the global optimum and (ii) are more efficient than the DE/Rand/1/Exp, the canonical PSO (Particle Swarm Optimization) and the standard real coded EA on classical benchmarks corrupted by zero-mean Gaussian noise.

[Shahryar et al., 2006,Shahryar et al., 2008] proposed a new Opposition-Based DE (ODE) algorithm, which uses Opposition-Based Optimization for population initialization, generation jumping and improving population's best individual. ODE has a concordant performance for both noise-free case and noisy case with constant variance of noise.

[Liu et al., 2008] combined the Optimal Computing Budget Allocation (OCBA) technique and the Simulated Annealing (SA) algorithm into differential evolution. Their hybrid algorithm is designed for noisy and uncertain environments inspired from real world applications. [Liu et al., 2008] concludes that, by incorporating both SA and OCBA into DE, the performance is improved for fitness functions corrupted by large noise.

We refer to [Lacca et al., 2012] for more on noisy optimization and differential evolution.

6.1.3 Non-adaptive and adaptive noisy differential evolution

6.1.3.1 Non-adaptive numbers of resamplings

We tested various rules for non-adaptive numbers of resamplings: $N_{linear} = n$; $N_{square\ root} = \sqrt{n}$; $N_{2exp} = 2^n$; $N_{constant} = 1$; $N_{scale} = \lceil D^{-2} \exp(\frac{4n}{5D}) \rceil$; where n is the generation index. After preliminary testing, we removed some of them for the clarity of graphs. The N_{scale} formula was derived in [Liu et al., 2014] based on scale analysis.

6.1.3.2 Adaptive numbers of resamplings

We propose here methods inspired from [Heidrich-Meisner and Igel, 2009, Hansen et al., 2009], adapted to differential evolution. It might make sense to resample until some statistical test becomes positive. For example, we might do a test based on standard deviation after each batch of 1000 resamplings. Such a batch size makes sense for parallelization, and for reducing the cumulative risk of false positive. We therefore define, for a pair of search points x, x' to be compared:

$$
\begin{aligned}
y_m &= \sum_{i=1}^{1000m} f(x, w^{(i)}), \\
y'_m &= \sum_{i=1}^{1000m} f(x', (w')^{(i)}) \\
\delta_m &= y_m - y'_m, \quad \mu_m = \frac{1}{m} \sum_{i=1}^{m} \delta_i, \\
\sigma_m^2 &= \frac{1}{m} \sum_{i=1}^{m} (\delta_i - \mu_m)^2 \\
N_{adaptive} &= \min_{m \in \{2,3,4,\dots\}} \{1000m; |\mu_m| > \frac{\sigma_m}{\sqrt{m-1}}\} \qquad (6.2) \\
N_{\substack{enhanced \\ adaptive}} &= \min\{ \min_{m \in \{2,3,4,\dots\}} \{10^3 m; |\mu_m| > \frac{\sigma_m}{\sqrt{m-1}}\}, \\
&\quad \lceil \frac{2^n}{10^3} \rceil \times 10^3 \} \qquad (6.3)
\end{aligned}
$$

This means that we evaluate points until there is a statistically significant difference. However, this is not a rigorous test, because we test repeatedly, after each group of 1000 resamplings. Even if each test is guaranteed (e.g. with confidence 95%), the whole set of tests is not

guaranteed with confidence 95% but with confidence $\max(0, 100-5P)\%$ after P tests. In particular, even if x and x' are equal, the procedure will eventually stop (more details below on this).

It is for sure possible to do tests differently. One can repeat the test and decrease the risk threshold so that the overall probability of misranking is always less than a fixed δ. But in such a case, we possibly get infinite loops when we meet a plateau. We keep our version above, and consider it as an adaptation ensuring almost sure halting. Indeed, [Heidrich-Meisner and Igel, 2009] also includes a rule for ensuring the finiteness of the number of resamplings.

Let us formalize below the almost sure halting property.

Property 3 (Almost sure halting of this procedure)**.** *If $f(x, w)$ and $f(x', w)$ have finite positive variance, then $N_{adaptive}$ is almost surely finite.*

Proof. If $\mathbb{E}f(x, w) \neq \mathbb{E}f(x', w)$, the result is immediate by the law of large numbers, applicable due to finite variance.

If $\mathbb{E}f(x, w) = \mathbb{E}f(x', w)$, then by the law of the iterated logarithm, the supremum over $i \in \{2, \dots, N\}$ of $\frac{\sqrt{i-1}\mu_i}{\sigma_i}$ is almost surely going to infinity as $N \to \infty$ - therefore, at some point Equation 14.10 holds, so almost surely N is finite. □

Let us formalize the concept of resampling rule. A resampling rule evaluates several times the fitness of two search points x and x'. $\hat{\mathbb{E}}f(x, w)$ is the average fitness value for x, and $\hat{\mathbb{E}}f(x', w)$ is the average fitness value for x'. At some point, the rule decides to stop reevaluating. It then outputs a result, which is either "x is better than x'", or "x' is better than x", or "x and x' have the same expected fitness", depending on the sign of $\hat{\mathbb{E}}f(x, w) - \hat{\mathbb{E}}f(x', w)$. We point out the following counterpart of Proposition 3 if a rigorous Bernstein race [Mnih et al., 2008] was applied. We consider a rule for choosing the number N of resamplings (it could be something else than Bernstein races), ensuring that the error rate is less than $1 - \delta$.

The error rate is defined as follows. When comparing search points x and x', it is the probability of concluding that $\mathbb{E}f(x, w) > \mathbb{E}f(x', w)$ (or, respectively, $\mathbb{E}f(x', w) > \mathbb{E}f(x, w)$) whereas it is wrong.

Property 4 (Races can have infinite loops on plateaus)**.** *Consider $i \in \{1, \dots, \lambda\}$ and $n \geq 1$. Consider $x = p_i$ and $x' = p_i''$ at iteration n of a DE algorithm. Assume that the resampling number N ensures that the*

error rate is less than $1-\delta$ *for some* $\delta > 0$*. Then, there is an objective function* f *and two search points* x *and* x' *such that with probability at least* $1-\delta$*,* N *is infinite.*

Proof. Let us assume that the fitness and the search points are such that $f(x,w)$ and $f(x',w)$ have the same probability distribution, with bounded density. Let us assume that the error rate is less than $1-\delta$ for some $\delta > 0$, and let us show that N is infinite with probability at least $1-\delta$.

First, due to the bounded density, the probability that $\hat{\mathbb{E}}f(x,w) = \hat{\mathbb{E}}f(x',w)$ is zero. This is because $f(x,w) - f(x',w)$ has a bounded density, hence $\hat{\mathbb{E}}f(x,w) - \hat{\mathbb{E}}f(x',w)$ has a bounded density, hence it is null with probability 0.

Therefore, the resampling rule, if it stops for a finite number N of resamplings, will conclude, with probability 1, that x is better than x', or that x' is better than x. It can not conclude that x and x' have the same expected fitness.

But, by assumption, the only correct answer is that x and x' have the same expected fitness.

Therefore it will conclude erroneously as there is no significant difference to find: the error rate $e \geq P(N < \infty)$.

But, by assumption, the error rate should be $e \leq \delta$: therefore $P(N < \infty) \leq e \leq \delta$. Therefore $P(N = \infty) \geq 1 - \delta$. □

This explains why upper bounds on numbers of resamplings are necessary when applying Bernstein races. Bernstein races without such a trick can lead to an infinite number of resamplings.

Our rules $N_{adaptive}$ and $N_{enhanced\ adaptive}$ verify the almost sure halting property, which is a good piece of news. However, associated drawbacks (which can't be avoided, by properties above) are

- the resampling loop will eventually stop and conclude that there is a difference whenever there is no significant difference and
- the resampling loop might fail (in terms of detecting the best), with probability arbitrarily close to $\frac{1}{2}$, in case of very close fitness values.

The *enhanced* version is pragmatically designed for avoiding uselessly long computations at the early stages. These elements show the difficult compromises when designing resampling rules based on statistical

Table 6.1: Notation of resampling rules used in the presented experiments (Figures 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7 and 6.8).

Rule	Formula	Notation in the figures
Constant	1	N_{cnst}
Linear	n	N_{lin}
Square root	$\lceil\sqrt{n}\rceil$	N_{sqrt}
Scale	$\lceil\frac{1}{D^2}exp(\frac{4n}{5D})\rceil$	N_{scale}
Adaptive	Equation14.10	N_{adap}
Enhanced adaptive	Equation14.11	N_{eadap}
Exponential	2^n	$2exp$
Exponential	$\lceil 1.1^n\rceil$	$1.1exp$
Exponential	$\lceil 1.01^n\rceil$	$1.01exp$

testing. Controlling the misranking probability leads to a risk of infinite resampling loop. When using empirical variance in tests, or when the noise has rare large values, another risk is the underestimation of the noise variance. Rigorous Bernstein races need bounds on possible values (the range parameter which is usually denoted by R in Bernstein races), which are hardly available in practice.

6.2 Experiments

Experimental results using different resampling rules are presented in this section. We refer to Table 6.1 for a description of the resampling rules and notations used in the presented experiments.

6.2.1 The CEC 2005 testbed

The CEC-2005 testbed [Suganthan et al., 2005] contains 25 benchmark functions. These functions are grouped into two main categories : *Unimodal functions* and *Multimodal functions*. There are 5 Unimodal functions $F_1, \ldots, F_5$ and 20 Multimodal functions grouped into 3 sub-categories. The first subcategory corresponds to 7 basic functions, named $F_6, \ldots, F_{12}$. The second sub-category are 2 expanded functions F_{13} and F_{14}. The last 11 Multimodal functions $F_{15}, \ldots, F_{25}$ are hybrid composite functions. F_1 is the translated sphere function. F_3 is an ill-

conditioned elliptic function. F_5, F_8 and F_{20} have their global optimum on the frontier of the domain.

6.2.2 Parameter selection for DE on the CEC 2005 testbed

[Tvrdık, 2006] used a portfolio with competition of parameter settings on DE/best/2 and DE/rand/1. Different F and Cr are used in the mutation and crossover steps with a probability updated at each iteration. A competition between different parameter settings was tested on two unimodal functions (first and second De Jong functions) and four multimodal functions (Rastrigin function, Schwefel function, Griewangk function and Ackley function) in dimension 5, 10 and 30. In this chapter, focusing on the extension to the strong noise case, we use parameters tuned in the noise-free case for DE/rand/2 and add a resampling tool as explained in Section 6.1.3.

6.2.3 Adding strong noise in the CEC 2005 testbed

As pointed out in [Beyer and Finck, 2008], troubles in noisy optimization by evolutionary algorithms start when the search points are close enough to the optimum. When the search points are close enough to the optimum, the noise standard deviation is close to the difference between fitness values. Then, convergence becomes difficult. [Arnold and Beyer, 2000,Jebalia and Auger, 2008] use a multiplicative noise model in which this never occurs because the variance decreases to zero around the optimum. [Auger et al., 2010] focuses on either multiplicative noise models, or constant noise models with smaller constants than our strong noise requirement. Strong noise models (constant variance, with magnitude significant compared to fitness variations) have been considered both theoretically [Fabian, 1967] and experimentally [Fabian, 1971, Coulom, 2012]. We work on DE, which is invariant by addition of a constant to the objective values; so we simplify graphs by considering functions with optimum fitness equal to 0, as follows:

$$f_{\substack{index_{CEC05},\\ noise\ free,\\ translated}}(x) = f_{\substack{index_{CEC05},\\ noise\ free,}}(x) - \inf_x f_{\substack{index_{CEC05},\\ noise\ free,}}(x),$$

where the index $index_{CEC05}$ is the function number as defined in the classical CEC 2005 testbed [Suganthan et al., 2005]. In this work, we consider the (noise free, translated) CEC 2005 testbed, and create a

strongly noisy counterpart as follows:

$$f_{\substack{index_{CEC05},\\ noisy,\\ translated}}(x) = f_{\substack{index_{CEC05},\\ noise\ free,\\ translated}}(x) + f_{\substack{index_{CEC05},\\ noise\ free,\\ translated}}(0) \times \mathbb{N}$$

where $\mathbb{N}$ is a standard Gaussian noise. The standard deviation of the noise is

$$f_{\substack{index_{CEC05},\\ noise\ free,}}(0) - \inf_x f_{\substack{index_{CEC05},\\ noise\ free,}}(0).$$

This noise model is easy to reproduce (just adding a Gaussian noise, using the fitness at 0 as standard deviation), and scales with the fitness values, leading to a strong noise model.

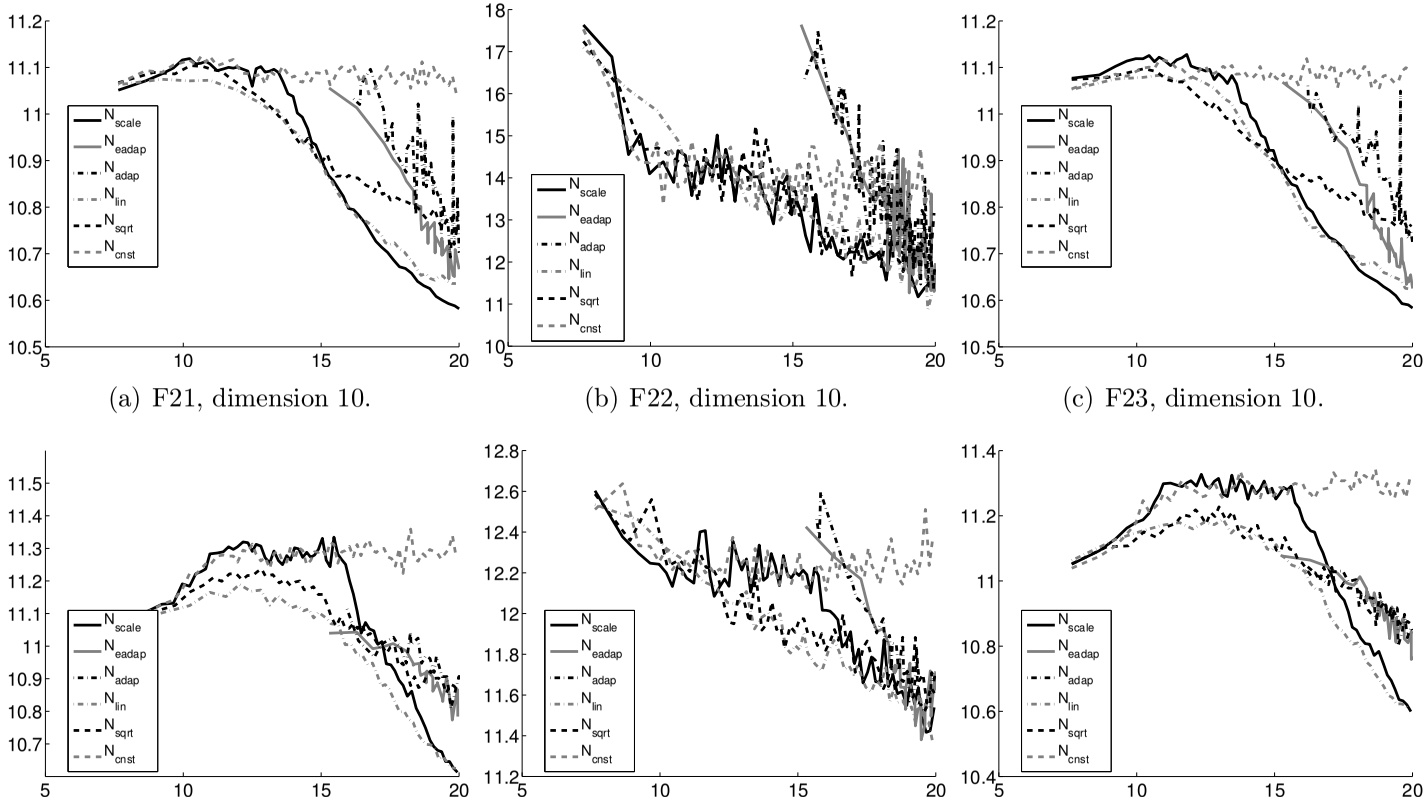
6.2.4 Experimental results

We do experiments in dimension 2, 10 and 30. Figures 6.1 and 6.2 present results on the "CEC 2005+Strong Noise" testbed described in Section 6.2.3, in dimension 10 (left) and 30 (right), for functions F_{21} to F_{25}. Figures 6.3 and 6.4 present results on the "CEC 2005+Strong Noise" testbed described in Section 6.2.3, in dimension 30, for functions F_{11} to F_{20}. Essentially, N_{lin} is usually the best or among the best; N_{scale} is sometimes better but there is no clear advantage for this more sophisticated formula. Experimental results for functions F_1 to F_{10} are not presented. Basically, F_8, F_9, F_{10} are poorly handled by all algorithms; for F_1-F_7, N_{scale} and N_{lin} perform best.

An advantage of N_{scale} is that [Astete-Morales et al., 2013] has proved that exponential rules, for a relevant choice of parameters, is consistent, i.e. guarantees some convergence rates when the original algorithm (without resampling) converges in the noise-free case. Also, the scaling analysis in [Liu et al., 2014] suggests some values of the parameters. Therefore, we also compared the heuristically derived formula N_{scale} to other exponential rules, $N_{2exp} = 2^n$, $N_{1.1exp} = 1.1^n$ and $N_{1.01exp} = 1.01^n$, and could check that

- the best coefficient in the exponential varies with the dimension;
- the heuristically derived equation N_{scale} approximately finds the right exponent, but some other formulas have approximately the same results.

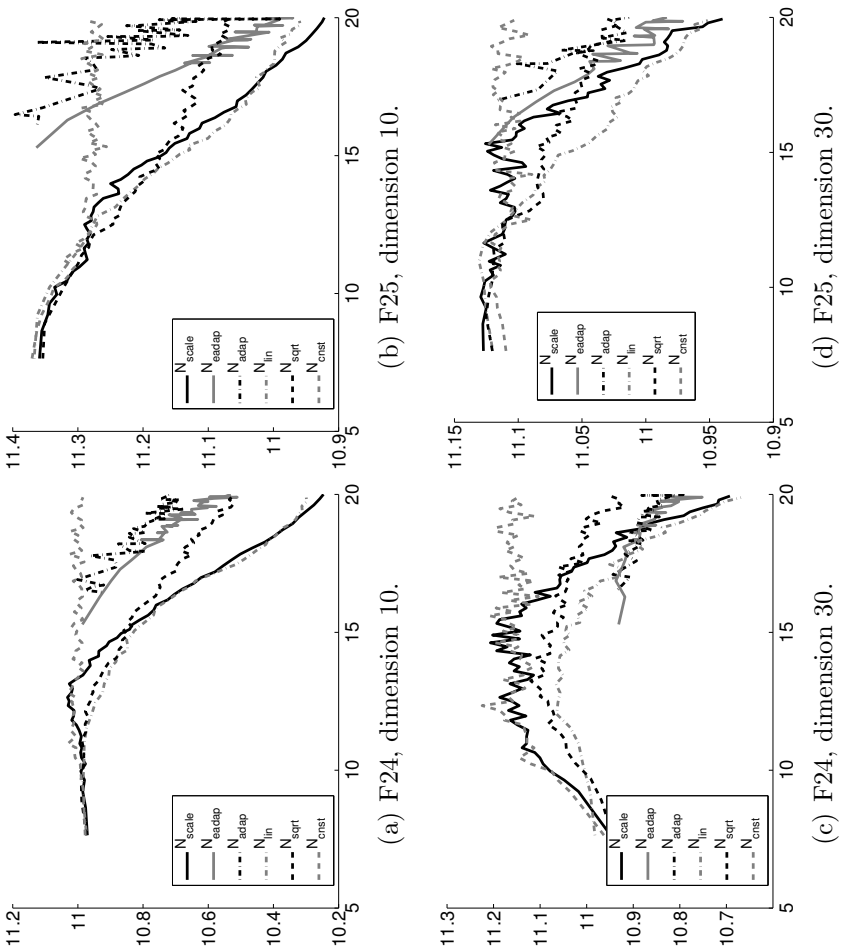
These results are presented in Figures 6.6, 6.7 and 6.8 for functions F_1-F_5.



(a) F21, dimension 10. (b) F22, dimension 10. (c) F23, dimension 10.

(d) F21, dimension 30. (e) F22, dimension 30. (f) F23, dimension 30.

Figure 6.1: (*x-axis:* $\log_2(\textit{number of evaluations})$, *y-axis:* $\log_2(\textit{simple regret})$) Multimodal functions F_{21} to F_{23} for dimension 10 (left) and dimension 30 (right). Results similar to results for F_1 to F_{20}, i.e. the "*scale*" and the linear formulas dominate. Standard deviations are present but tiny and almost invisible.



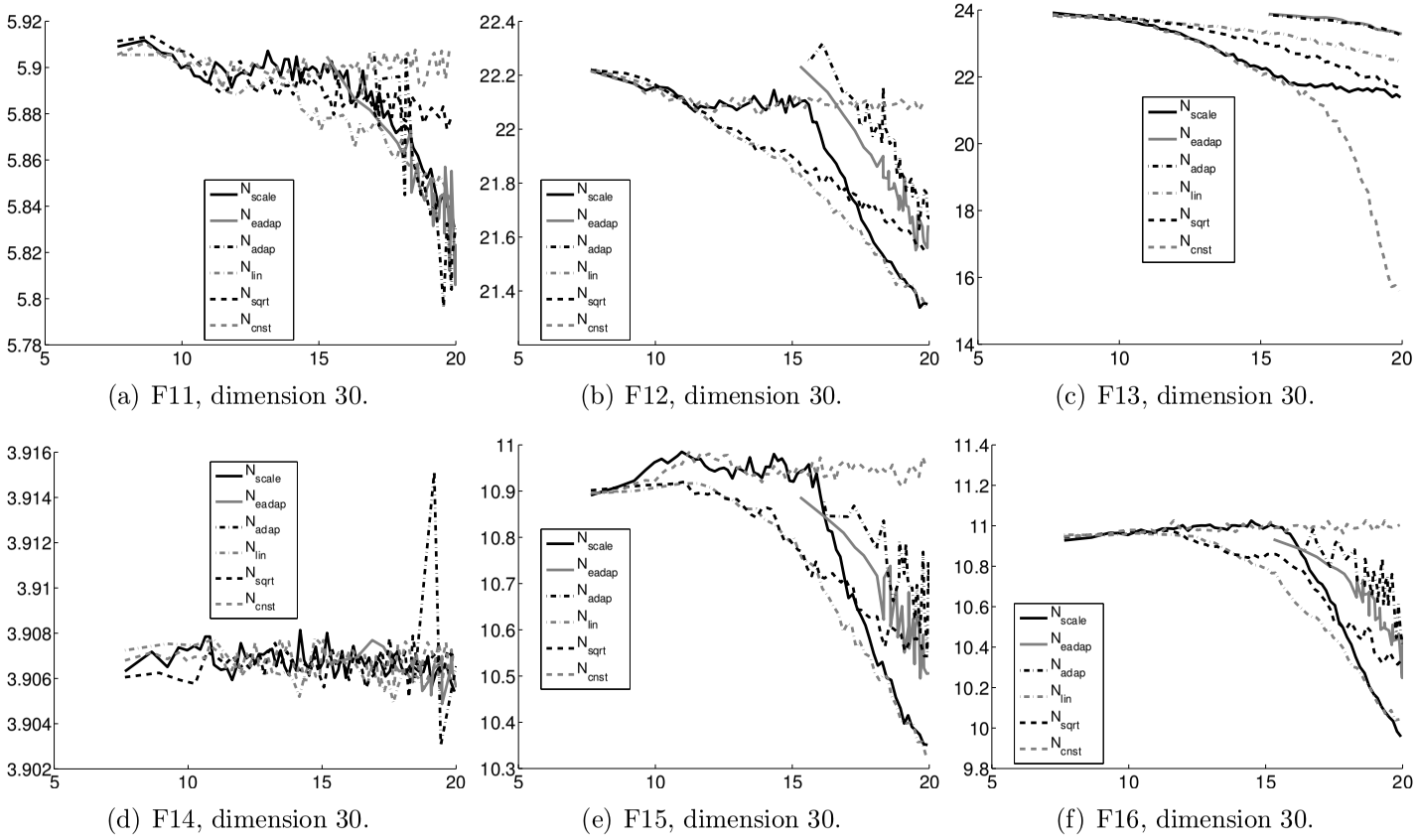
(a) F24, dimension 10.

(b) F25, dimension 10.

(c) F24, dimension 30.

(d) F25, dimension 30.

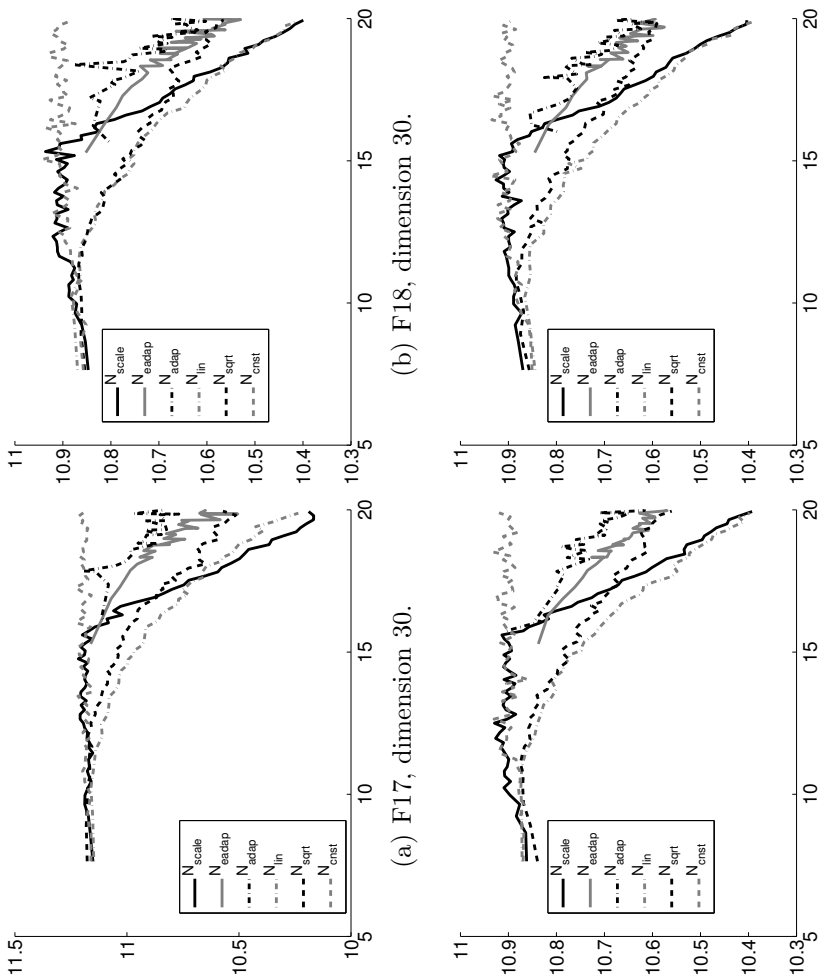
Figure 6.2: (*x-axis:* $\log_2(\textit{number of evaluations})$, *y-axis:* $\log_2(\textit{simple regret})$) Multimodal functions F_{24} and F_{25} for dimension 10 (left) and dimension 30 (right). Results similar to results for F_1 to F_{20}, i.e. the "*scale*" and the linear formulas dominate. Standard deviations are present but tiny and almost invisible.



(a) F11, dimension 30. (b) F12, dimension 30. (c) F13, dimension 30.

(d) F14, dimension 30. (e) F15, dimension 30. (f) F16, dimension 30.

Figure 6.3: (*x-axis:* $\log_2(\textit{number of evaluations})$, *y-axis:* $\log_2(\textit{simple regret})$) Multimodal functions F_{11} to F_{16} for dimension 30. Results on F_{13} are quite special; see discussion in Section 6.3.2. Linear and "*scale*" numbers of resamplings dominate. Standard deviations are present but tiny and almost invisible.



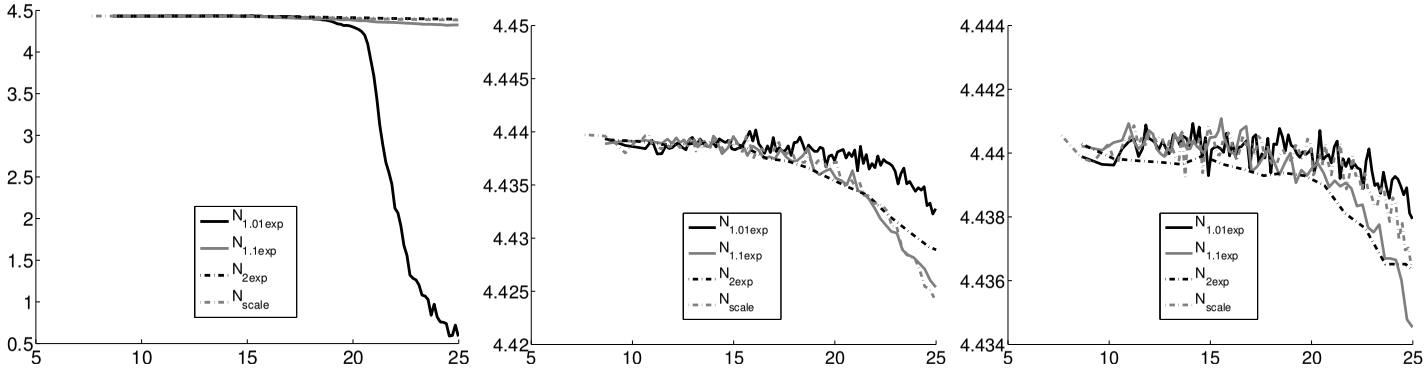
(a) F17, dimension 30.

(b) F18, dimension 30.

(c) F19, dimension 30.

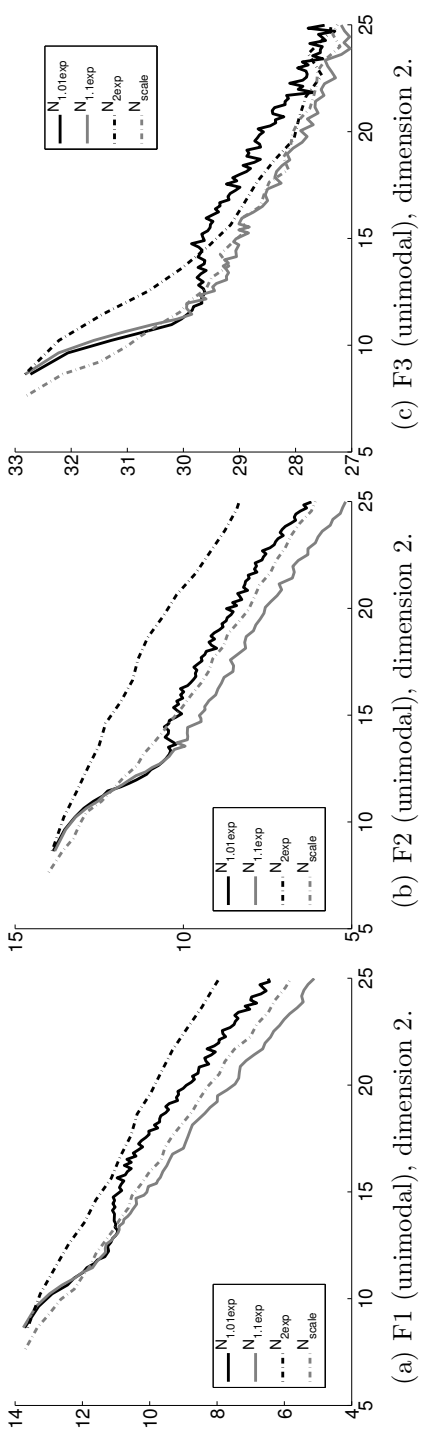
(d) F20, dimension 30.

Figure 6.4: (*x-axis:* $\log_2(number\ of\ evaluations)$, *y-axis:* $\log_2(simple\ regret)$) Multimodal functions F_{17} to F_{20} for dimension 30. Results on F_{13} are quite special; see discussion in Section 6.3.2. Linear and "*scale*" numbers of resamplings dominate. Standard deviations are present but tiny and almost invisible.



(a) F8, dimension 2. (b) F8, dimension 10. (c) F8, dimension 30.

Figure 6.5: (*x-axis:* $\log_2(\textit{number of evaluations})$, *y-axis:* $\log_2(\textit{simple regret})$) Test of exponential numbers of resamplings, including the heuristically derived formula "*scale*" and with larger numbers of function evaluations for F_8 (Shifted Rotated Ackley's Function with Global Optimum on Bounds). Standard deviations are tiny and almost invisible. Only the exponential rule with small exponent 1.01 finds a reasonable approximation of the optimum for F_8 in dimension 2. No algorithm finds a reasonable approximation of the optimum in dimension 10 or dimension 30.



(a) F1 (unimodal), dimension 2.

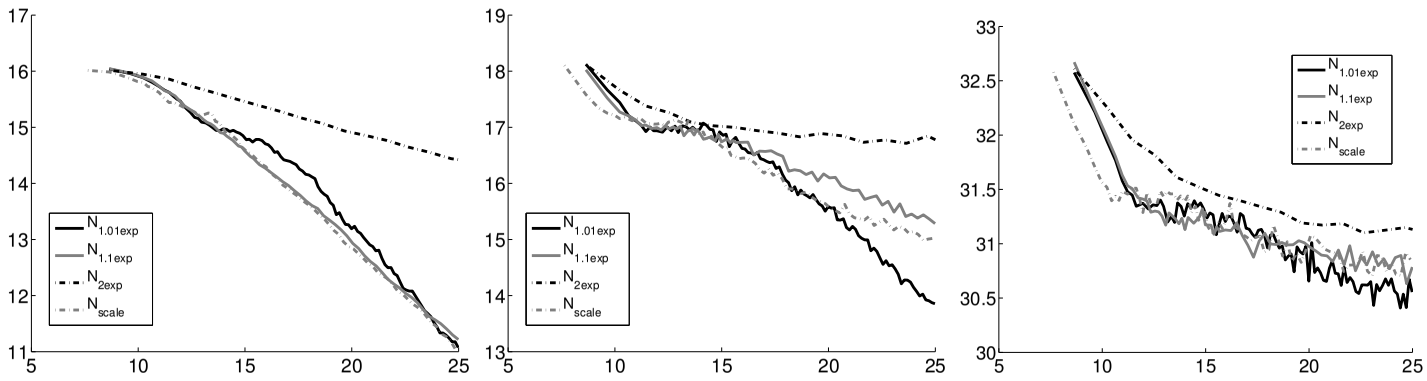
(b) F2 (unimodal), dimension 2.

(c) F3 (unimodal), dimension 2.

(d) F4 (unimodal), dimension 2.

(e) F5 (unimodal), dimension 2.

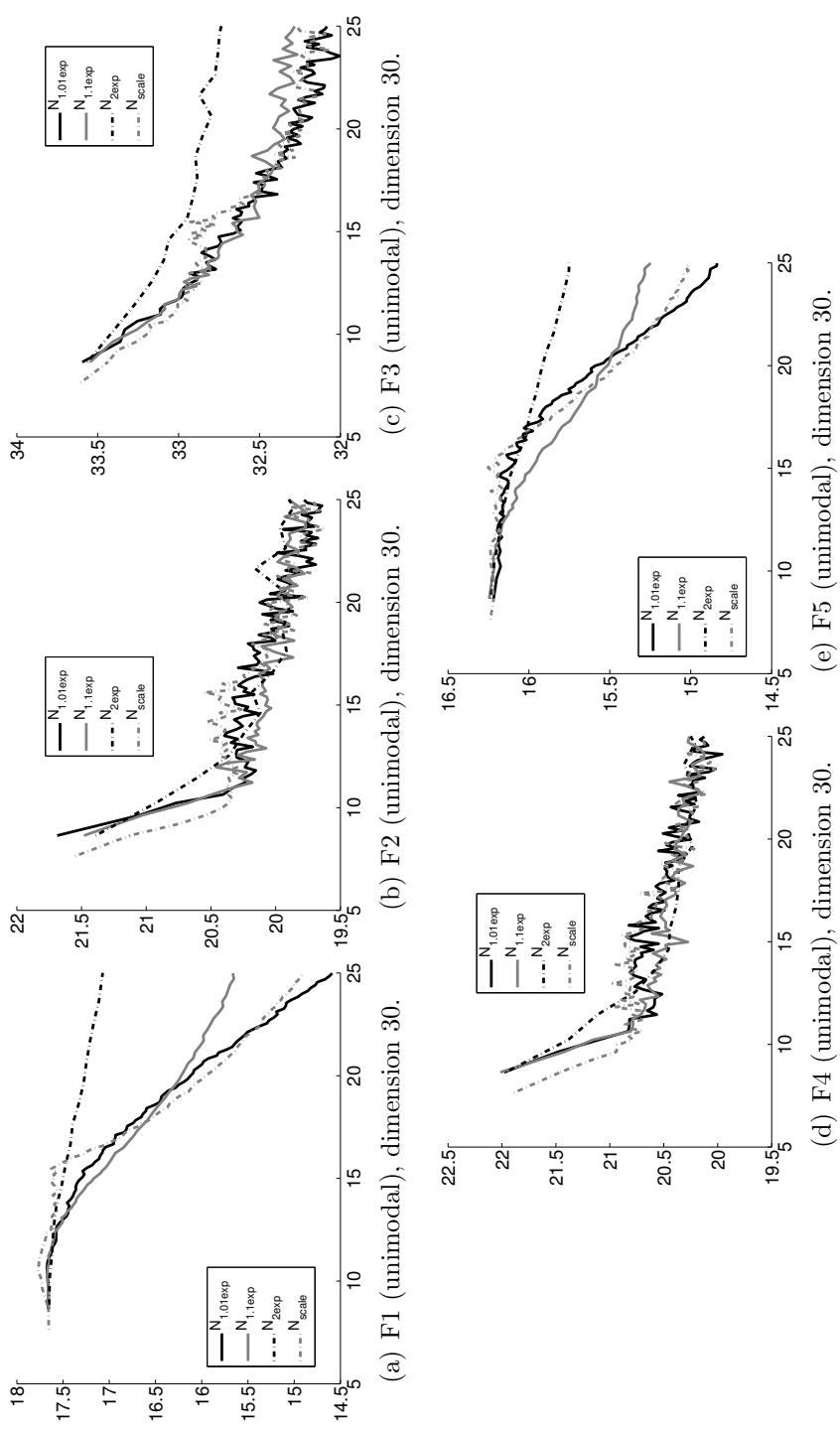
Figure 6.6: (*x-axis:* $\log_2(\textit{number of evaluations})$, *y-axis:* $\log_2(\textit{simple regret})$) Test of exponential numbers of resamplings, including the heuristically derived formula "*scale*" and with larger numbers of function evaluations. Standard deviations are tiny and almost invisible. $N_{1.1exp}$ outperforms $N_{1.01exp}$ and N_{2exp} in dimension 2.



(a) F1 (unimodal), dimension 10. (b) F2 (unimodal), dimension 10. (c) F3 (unimodal), dimension 10.

(d) F4 (unimodal), dimension 10. (e) F5 (unimodal), dimension 10.

Figure 6.7: (*x-axis:* $\log_2$(*number of evaluations*), *y-axis:* $\log_2$(*simple regret*)) Test of exponential numbers of resamplings, including the heuristically derived formula "*scale*" and with larger numbers of function evaluations. Standard deviations are tiny and almost invisible. $N_{1.01exp}$ is dominant for F_2-F_5 in dimension 10, $N_{1.1exp}$ and N_{scale} perform similarly for F_1 in dimension 10.



(a) F1 (unimodal), dimension 30. (b) F2 (unimodal), dimension 30. (c) F3 (unimodal), dimension 30.

(d) F4 (unimodal), dimension 30. (e) F5 (unimodal), dimension 30.

Figure 6.8: (*x-axis:* $\log_2(number\ of\ evaluations)$, *y-axis:* $\log_2(simple\ regret)$) Test of exponential numbers of resamplings, including the heuristically derived formula "*scale*" and with larger numbers of function evaluations. Standard deviations are tiny and almost invisible. $N_{1.01exp}$ is dominant in dimension 30, N_{scale} has sometimes similar performance. In small dimension, the slope in the log-log representation is close to $-\frac{1}{2}$; this is the optimal possible rate for a wide class of algorithms as discussed in [Astete-Morales et al., 2015].

6.3 Discussion & conclusions

Our experiments are performed in the strongly noisy case, i.e. standard deviation of noise approximately equal to the range of fitness values.

6.3.1 Main observations

The simple linear resampling method performs well in general. However, the adaptive methods have a similar or better slope at the end, which suggests that they might be better asymptotically - though we have already reached huge numbers of fitness evaluations. The enhanced adaptive method performs better than its non-adaptive counterpart. It just restricts the number of evaluations to 2^n, with n the generation index. The N_{scale} formula performs well and in particular performs always nearly as well as the best of other formulas. However, N_{lin} or $N_{1.01exp}$ perform very similarly.

6.3.2 Special cases

F_8 (Shifted Rotated Ackley's Function with Global Optimum on Bounds) and F_{14} (Shifted Rotated Expanded Scaffer's Function (F_6)) are very hard in the strongly noisy case; no algorithm finds a reasonable approximation of the optimum except F_8 in dimension 2 where only the "*scale*" formula succeeds (see Figure 6.5).

On F_{13} (Expanded Extended Griewank's plus Rosenbrock's Function ($F_8 + F_2$)), just one sampling ($r_n = 1$) is enough for the early iterations; but, asymptotically, increasing the number of resamplings becomes necessary in dimension 10 - not in dimension 30 for the considered budget.

6.3.3 Conclusion: resampling in differential evolution

The contributions of this work are as follows. We propose a strong noise model. This model is more difficult, but close to many real world problems. In many problems, the standard deviation of the noise is close to the differences between fitness values. The literature proposes various non-adaptive rules, and provides theoretical guarantees, for evolution strategies. We here transfer these rules to differential evolution.

Our rules were able to match the theoretical limit $-\frac{1}{2}$ for a class of algorithms described in [Astete-Morales et al., 2015]. It is not proved that DE is in the scope of [Astete-Morales et al., 2015]. Still, the agreement between Figure 6.8 and theory is striking.

We pointed out problems with adaptive rules: difficulties with equal fitness values (e.g. plateaus), leading to possibly infinite loops in case of resamplings until statistically significant differences.

The main take-home messages of this work on resampling numbers in evolution strategies are: (i) $N_{scale} = \lceil D^{-2} \exp(\frac{4n}{5D}) \rceil$, heuristically derived in [Liu et al., 2014], performs reasonably well but it does not clearly outperform other formulas and in particular the simple and robust $N_{lin} = n$ and other exponential formulas with small coefficients, e.g. $N_{1.01exp} = 1.01^n$. Smaller numbers of resamplings or faster exponential (e.g. $N_{1.1exp} = 1.1^n$) do not provide satisfactory results. Therefore, we might recommend $N_{1.01exp}$ for example.

(ii) Adaptive methods might be merged with bounds on resampling numbers (as shown by the compared performance of the enhanced adaptive method, compared to the default adaptive method); in our results non-adaptive methods such as N_{scale} improve adaptive methods by setting a limit on the numbers of resamplings, avoiding wasted evaluations in early stages.

(iii) Non-adaptive bounds added on top of adaptive methods, improve these adaptive methods, but the fact that adaptive methods improve non-adaptive methods is unclear in this setting. The adaptive method do not bring clear improvements compared to simple non-adaptive schemes and were indeed usually worse. This is consistent with e.g. [Astete-Morales et al., 2015] which finds better results for an evolution strategy with a simple non-adaptive rule than with uncertainty handling version such as UH-CMA [Hansen et al., 2009]. Still, these combinations (a good non-adaptive formula plus an adaptive rule based on statistical testing) might be good for easier models of noise when a transient regime in which noise is negligible can be improved by reducing resamplings in the early stages - but on the long run there is no improvement for the additive noise model.

7 Grey box noisy optimization

7.1 Grey box noisy optimization

Black-box and white-box cases correspond to two extreme cases: when there is no access to any internal property of the objective function, and when strong properties of the objective function can be used. The grey box case is an intermediate case. We here focus on the case where a random seed can be used.

The state-of-the-art of variance reduction methods are presented in Section 2.1.6.

7.2 Algorithms

7.2.1 Different forms of pairing

For each request x_n to the objective function oracle, the algorithm also provides a set $Seed_n$ of random seeds; $Seed_n = \{seed_{n,1}, \ldots, seed_{n,m_n}\}$. $\mathbb{E}f(x_n, w)$ is then approximated as $\frac{1}{m_n}\sum_{i=1}^{m_n} f(x_n, seed_{n,i})$.

One can see in the literature different kinds of pairing. The simplest one is as follows: all sets of random seeds are equal for all search points evaluated during the run, i.e. $Seed_n$ is the same for all n. The drawback of this approach is that it relies on a sample average approximation: the good news is that the objective function becomes deterministic; but the approximation of the optimum is only good up to the relevance of the chosen sample and we can not guarantee convergence to the real optimum. Variants consider m_n increasing and nested sets $Seed_n$, such as $\forall(n \in \mathbb{N}^+, i \leq m_n)$, $m_{n+1} \geq m_n$ and $seed_{n,i} = seed_{n+1,i}$. A more sophisticated version is that all random seeds are equal inside an offspring, but they are changed between offspring (see discussion above). We will test this, as an intermediate step between CRN and no

pairing at all. In Section 7.2.2, we explain on an illustrative example why in some cases, pairing can be detrimental. It might therefore make sense to have partial pairing. In order to have the best of both worlds, we propose in Section 7.2.3 an algorithm for switching smoothly from full pairing to no pairing at all.

7.2.2 Why common random numbers can be detrimental

The phenomenon by which common random numbers can improve convergence rates is well understood; correlating the noise between several points tends to transform the noise into a constant additive term, which has therefore less impact - a perfectly constant additive term has (for most algorithms) no impact on the run. Setting $\alpha = 1$ in Equation 7.1 (below), modelizing an objective function, provides an example in which pairing totally cancels the noise.

$$f(x, w) = ||x||^2 + \alpha w' + 20(1 - \alpha)w'' \cdot x \tag{7.1}$$

We here explain why CRN can be detrimental on a simple illustrative example. Let us assume (toy example) that

- We evaluate an investment policy on a wind farm.
- A key parameter is the orientation of the wind turbines.
- A crucial part of the noise is the orientation of wind.
- We evaluate 30 different individuals per generation, which are 30 different policies - each individual (policy) has a dominant orientation.
- Each policy is evaluated on 50 different simulated wind events.

With CRN: If the wind orientation (which is randomized) was on average more East than it would be on expectation, then, in case of pairing (i.e. CRN), this "East orientation bias" is the same for all evaluated policies. As a consequence, the selected individuals are more East-oriented. The next iterate is therefore biased toward East-oriented.

Without CRN: Even if the wind orientation is too much East for the simulated wind events for individual 1, such a bias is unlikely to occur for all individuals. Therefore, some individuals will be selected with a East orientation bias, but others with a West orientation bias or other biases. As a conclusion, the next iterate will incur an average of many uncorrelated random biases, which is therefore less biased.

7.2.3 Proposed intermediate algorithm

We have seen that pairing can be efficient or detrimental depending on the problem. We will here propose an intermediate algorithm (Algorithm 5), somewhere in between the paired case ($g(r) = r$) and the totally unpaired case ($g(r) >> r$).

Algorithm 5 One iteration of a population-based noisy optimization algorithm with pairing.

Input: A population-based noisy optimization algorithm (in particular, rule for generating offspring)
Input: n: current iteration number
Input: $r \in \mathbb{N}^+$: a resampling rule
Input: λ: a population size
Input: $g : \mathbb{N}^+ \to \mathbb{N}^+$: a non-decreasing mapping such that $g(r) \geq r$
1: Generate λ individuals $i_1, \ldots, i_\lambda$ to be evaluated at this iteration
2: Compute the resampling number r by the resampling rule
3: Generate $P_{r,g(r)} = (w_{r,1}, \ldots, w_{r,g(r)})$ a set of $g(r)$ random seeds (we will see below different rules)
4: Each of these λ individuals is evaluated r times with r distinct random seeds randomly drawn in the family $P_{r,g(r)}$.

The $P_{r,g(r)}$ can be

- Nested, i.e. $\forall (i,r),\ g(r) \geq i \Rightarrow w_{r,i} = w_{r+1,i}$. The $(w_{r,i})_{i \leq g(r)}$ for a fixed r are then independent.
- Independent, i.e. all the $w_{r,i}$ are randomly independently identically drawn.

SAA is equivalent to the nested case with $n \mapsto r(n)$ constant, i.e. we always use the same set of random seeds. [de Matos et al., 2012] corresponds to the nested case. Classical CRN consists in $g(r) = r$ and independent sampling.

We will design, in Section 7.3, an artificial testbed which smoothly (parametrically depending on α in Equation 7.1) switches

- from an ideal case for pairing (testbed in which pairing cancels the noise, as $\alpha = 1$ in Equation 7.1);

- to worst case for pairing (counterexample as illustrated above, Section 7.2.2).

and which (depending on $g(\cdot)$) switches from fully paired to fully independent. We will compare stratified sampling and paired sampling on this artificial testbed. Later, we will consider a realistic application (Section 7.4).

7.3 Artificial experiments

We consider a $(\mu/\mu, \lambda)$-Self-Adaptive Evolution Strategy, with $\lambda = 8d^2$, $\mu = min(2d, \lambda/4)$ and some resampling rule $r(n) = \lceil n^d \rceil$, where n is the current iteration number. We apply Algorithm 5 with $g : \mathbb{N}^+ \mapsto \mathbb{N}^+$ defined by

$$g(r) = round(r^\beta),$$

where $\beta \geq 1$ is a parameter which regulates the pairing level. When $\beta = 1$, the function evaluations are fully paired; when $\beta \to \infty$, the function evaluations are fully independent. All experiments are performed with 10000 function evaluations and are reproduced 9999 times.

7.3.1 Artificial testbed for paired noisy optimization

With $w = (w', w'')$, let us define

$$f(x, w) = ||x||^2 + \alpha w' + 20(1 - \alpha) w'' \cdot x \tag{7.1}$$

where $\cdot$ denotes the scalar product. Two different cases are considered for the random processes:

- *Continuous case:* w' is a unidimensional standard Gaussian random variable and w'' is a d-dimensional standard Gaussian random variable.
- *Discrete case:* w' is a Bernoulli random variable with parameter $\frac{1}{2}$ and w'' is a vector of d independent random variables equal to 1 with probability $\frac{1}{2}$ and -1 otherwise. For the stratified sampling, in case of 4 strata, we use the 2 first components of w'', which lead to 4 different cases: one for $(-1, -1)$, one for $(1, 1)$, one for $(-1, 1)$ and one for $(1, -1)$.

The motivations for this testbed are as follows:

- It is a generalization of the classical sphere function.
- The case $\alpha = 1$ is very easy for pairing (just a *Sample Average Approximation* (SAA) is enough for fast convergence as in the noise-free case - $\beta = 1$, i.e. $g(r(n)) = r(n)$, leads to canceling noise, even with resampling number $r(n) = 1$).
- The case $\alpha = 0$ is very hard for pairing; the case $\beta = 1$ (full pairing) means that the noise has the same bias for all points.
- For the discrete framework, the stratified sampling directly reduces the dimension of the noisy case: the two first components have no more noise in the stratified case.

7.3.2 Experimental results

We study

$$\mathbb{E}\frac{\log ||x||^2}{\log n_e} \tag{7.2}$$

(the lower the better), where x is the estimate of the optimum after $n_e = 10000$ function evaluations and the optimum is 0. Experiments are reproduced 9999 times. The continuous case leads to results in Table 7.1. Standard deviations are ± 0.0015 for the worst cases and are not presented. Essentially, the results are:

- When α is close to 1, small β (more pairing) is better.
- When α is close to 0, large β (nearly no pairing) is better.

In the discrete case, it is easy to define pairing: we can use strata correspond to distinct values of the two first components of w''. Using the four strata corresponding to the 2 possible values of each of the two first components of w'', we get results presented in Table 7.2. We still see that pairing is good or bad depending on the case (sometimes leading to no convergence whereas the non-paired case converges, see row $\alpha = 0$ in dimension 5) and never brings huge improvements; whereas stratified sampling is always a good idea in our experiments.

Table 7.1: Efficiency of pairing (i.e. case β small) in the continuous case. Left hand side columns (β small) have more pairing than right hand side columns. Pairing is efficient for the "gentle" noise $\alpha = 1$, up to a moderate 50% faster; but it is harmful when $\alpha = 0$ (correlated noise). Next results will investigate stratification. Bold font shows best performance and significant improvements. Positive numbers correspond to no convergence; they are never in bold. Intermediate values of β (intermediate levels of pairing) were never significantly better than others and not clearly more robust to changes in α.

α	$\beta = 1.0$ (paired)	$\beta = 1.16$	$\beta = 1.35$	$\beta = 1.57$	$\beta = 1.82$	$\beta = 2.12$	$\beta = 2.46$ ($\simeq$ unpaired)
dimension 2 (bold for best tested algorithm)							
0	-0.07435	-0.06654	-0.07670	-0.08581	-0.09219	**-0.09603**	-0.09344
0.8	-0.34475	-0.34661	-0.35921	-0.36253	-0.36565	-0.36709	**-0.36917**
1	**-**0.75048	-0.52772	-0.50544	-0.49794	-0.49109	-0.49339	-0.49182
dimension 3 (bold for best tested algorithm)							
0	-0.06258	-0.06373	-0.07978	-0.09489	-0.10463	-0.10931	**-0.10977**
1	**-0.47681**	-0.43320	-0.41439	-0.41004	-0.40880	-0.40202	-0.39641
dimension 5 (bold for best tested algorithm)							
0	0.02965	0.03964	0.04409	0.04394	0.04680	0.04826	0.04823
0.8	-0.15077	-0.15977	-0.16369	-0.16687	-0.16770	-0.16793	**-0.16920**
1	**-0.23235**	-0.23188	-0.23174	-0.23125	-0.23225	-0.23232	-0.23182

Dimension 10 (bold for best tested algorithm)		
	$\beta = 1$ (paired)	$\beta = \infty$ ($\simeq$ unpaired)
0	0.097	**-0.033**
0.8	0.038	**-0.053**
1.0	**-0.057**	-0.054

7.4 Real world experiments

7.4.1 Paired noisy optimization for dynamic problems

Paired statistical tests (e.g. Pegasus [Dowell and Jarratt, 1972]) convert a stochastic optimization problem into a deterministic and easier one. Although Pegasus can cause excessive "overfitting" (specialization to the set of considered seeds) when using a fixed number of scenarios, several methods, e.g. using *Wilcoxon signed rank sum test* or changing the scenarios during learning, can reduce the "overfitting" [Strens and Moore, 2001, Strens et al., 2002]. *Wilcoxon signed rank sum test* pays more attention to small improvements across all scenarios rather than large

Table 7.2: Table of results (slope as in Equation 7.2; the lower, the better) depending on α (defining the problem) and β (defining the level of pairing; $\beta = 1$ means full pairing, β large means no pairing). We see that pairing can have a positive or a negative effect. We include results with stratified sampling; which are better or much better depending on the cases. Negligible standard deviations are not presented. Numbers in the stratified case are in bold when they outperform the non stratified setting.

α	$\beta = 1.0$	$\beta = 1.16$	$\beta = 1.35$	$\beta = 1.57$	$\beta = 1.82$	$\beta = 2.12$	$\beta = 2.46$
dimension 2, no stratified sampling (bold for signif. best)							
0	-0.07200	-0.06392	-0.07926	-0.08873	-0.09539	-0.09443	-0.09382
1	-0.74716	-0.52659	-0.50665	-0.49758	-0.49383	-0.49402	-0.49310
dimension 3, no stratified sampling (bold for signif. best)							
0	-0.00802	-0.00519	-0.01246	-0.01672	-0.01750	-0.01660	-0.01635
0.4	-0.09327	-0.10422	-0.11704	-0.12771	-0.13248	-0.13375	-0.13138
0.8	-0.25365	-0.27016	-0.28168	-0.29045	-0.29341	-0.29459	-0.29474
1	-0.39480	-0.38398	-0.37981	-0.37504	-0.37562	-0.37646	-0.37653
dimension 3, stratified sampling (bold if better than no stratification)							
0	-0.01931	**-0.01396**	**-0.02585**	**-0.03590**	**-0.04430**	**-0.04836**	-0.04744
0.8	-0.26548	**-0.28079**	**-0.29481**	**-0.30133**	**-0.30797**	**-0.30761**	-0.30763
1	-0.39714	-0.38346	**-0.38021**	**-0.37749**	-0.37411	-0.37614	-0.37442
dimension 5, no stratified sampling (bold for signif. best)							
0	0.03285	0.04253	0.04896	0.04962	0.05125	0.05336	0.05412
1	-0.23188	-0.23207	-0.23265	-0.23080	-0.23219	-0.23148	-0.23042
Dimension 5, stratified sampling (bold if better than no stratification)							
0	0.00197	-0.00880	-0.02657	-0.04158	-0.04991	**-0.05404**	-0.04617
1	-0.23294	-0.23146	-0.23161	-0.23150	-0.23228	-0.23158	-0.23198
Dimension 10, no stratified sampling (bold for signif. best)							
	$\beta = 1$ (paired)						$\beta = \infty$ ($\simeq$ unpaired)
0	0.108						-0.105
0.8	0.012						-0.072
1	-0.056						-0.055
Dimension 10, stratified sampling (bold if better than no stratification)							
0	0.047						**-0.106**
0.8	**-0.033**						-0.072
1	**-0.057**						**-0.056**

changes over the return of an individual one, so that it can reduce the "overfitting" caused by a few extreme (good or bad) scenarios. [Strens et al., 2002] also shows that using an adaptive number of trials for each policy can speed-up learning in such a CRN framework. In the present work, we use new scenarios for each generation - we assume that there is no constraint on the availability of possible realizations w. Another related existing work is [Kleinman et al., 1999]. It compares *Independent*

Random Numbers (IRN), *Common Random Numbers* (CRN) and *Partial Common Random Numbers* (PCRN, which use pairing in the sense that the same pseudo-random numbers are used several times but in different orders) for *Simultaneous Perturbation Stochastic Approximation* and *Finite Differences Stochastic Approximation*. Both algorithms are faster when using CRN. The present work is dedicated to evolution strategies.

7.4.2 Unit commitment problem

For real world experiments, we consider the following sequential decision making problem in the *Markov Decision Processes* (MDP) framework, using discrete time steps: 10 batteries are managed to store energy bought and sold on the electricity market and 10 decision variables have to be made at each time step (i.e. the quantity of energy to buy or to sell for each battery) in order to maximize profits. We apply *rolling planning*, also known as *shrinking horizon*, i.e. new forecasts are used for updating the decisions. There are 168 time steps, i.e. 7 days with one hour per time step. We use an *operational horizon* $o = 5$ time steps, i.e. decisions are made by groups of 5 time steps. When a decision is made, it covers 5 decisions and there is no recourse on these decisions. We have a *tactical horizon* $h = 10$ time steps, i.e. we optimize over the 10 next time steps to speed up computations instead of doing it for all remaining time steps.

7.4.3 Testbed

We define the following variables: x is the vector of the weights of a neural network; x parametrizes the energy policy described in Equations 7.3 and 7.4 and d is the dimension of x. w is a random process modeling the market price. The policy (Equation 7.3) uses a neural network to decide the parameters (Equation 7.4) of the valorization function. The valorization function provides an estimate of the marginal value of each stock; that is, it provides, for each stock, how much (on the reward over the tactical horizon) we are willing to pay for increasing this stock by one unit.

$$\begin{aligned} d_t &= \arg\max(\ \textit{reward over}\ (t, \ldots, t+h)\ ) \\ &+ \sum_{i=1}^{d'} \zeta_i s_{t+h,i}. \end{aligned} \tag{7.3}$$

Each state variable corresponds to a stock. We see in Equation 7.3 a compromise between the current reward (first term) and the sum $\sum_{i=1}^{10} \zeta_i \times s_{t+h,i}$ over stocks (second term). The ζ_i are estimates of the marginal values of each stock by the neural network. In Equation 7.3, d_t is the vector of decisions to apply from the current time step t to time step $t+h$; $s_{t+h} = (s_{t+h,1}, \ldots, s_{t+h,d'})$ is the state at the end of the tactical horizon (the quantity of energy contained in each of the 10 batteries); d' is the number of outputs of the neural network. It is equal to the number of stocks, as we have one marginal value per stock. ζ_i is the i^{th} output of the neural network:

$$(\zeta_1, \ldots, \zeta_{d'}) = neuralNetwork(x, s_t). \tag{7.4}$$

$s_{t+h,i}$ depends on the random process and the decision:

$$(reward_t, s_{t+h}) = transition(s_t, d_t, random\ process). \tag{7.5}$$

$reward_t$ is the reward over the operational horizon, i.e. from time t to $t+o$, i.e. $t+5$. The *transition* function describes the problem. We use a (μ, λ)-evolution strategy to optimize x according to the objective function $f(x, w)$. $f(x, w)$ is the simulation function: it applies repeatedly the policy (Equation 7.3) and the *transition* function (Equation 7.5) from an initial state s_0 to a final state s_{168}. The returned value is the cumulative reward, i.e. the sum of the $reward_t$. The following setup is used: $d = 60$; $\lambda = 4(d+1) = 244$; $\mu = \lambda/4$; $r(n) = \lceil 10\sqrt{n+1} \rceil$. We define paired optimization (a.k.a common random numbers) and stratified sampling in such a case:

- We apply an evolutionary algorithm for optimizing the parameters (i.e. the weights) $x = (x_1, \ldots, x_{60})$ of the neural network controller.
- Each evaluation is a Monte Carlo average reward for a vector of parameters; a Monte Carlo evaluation is a call to $f(x, w)$ above.
- These evaluations are either pure Monte Carlo, paired Monte Carlo, stratified Monte Carlo or paired stratified Monte Carlo.

Common random numbers for energy policies: In the case of CRN (also known as pairing) for the specific case of energy policies, we apply $g(r(n)) = r(n)$, i.e. the same random outcomes $w_1, \ldots, w_{r(n)}$ are used for all individuals of a generation. The random outcomes $w_1, \ldots, w_{r(n)}$ are independently drawn for each new generation.

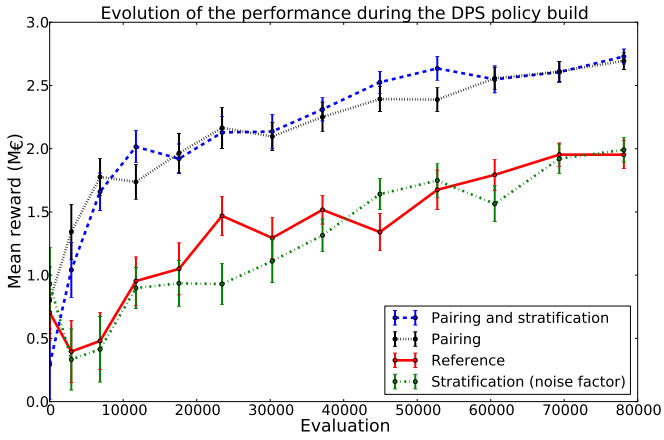


Figure 7.1: X-axis: evaluation index. Y-axis: reward (the higher the better). We see that pairing is very efficient whereas stratification provides no clear improvement.

Stratified sampling for energy policies: Stratification in the general case was defined earlier; we here discuss the application to our specific problem. It is very natural, as far as possible, to ensure that points are equally sampled among the 25% best cases, the 25% worst cases, the second quartile and the third quartile.

Even if these categories can only be approximately evaluated, this should decrease the variance. It is usually a good idea to stratify according to quantiles of a quantity which is as related as possible to the quantity to be averaged, i.e. $f(x, w)$. The four strata are the four quantiles on the annual average of an important scalar component of the noise.

Experimental results in Figure 7.1 show that pairing provides huge improvement in the realistic case. Stratification has a minor impact.

7.5 Conclusions: common random numbers usually work but there are dramatic exceptions

We tested, in an artificial test case and a Direct Policy Search problem in power management, paired optimization (a.k.a common random numbers) and partial variants of it. We also tested stratified sampling.

Both algorithms are easy to implement, "almost" black-box and applicable for most applications. Paired optimization is unstable; it can be efficient in simple cases, but detrimental with more difficult models of noise, as shown by results on $\alpha = 1$ (positive effect) and $\alpha = 0$ (negative effect) in the artificial case (Equation 7.1). We provided illustrative examples of such a detrimental effect (Section 7.2.2). Stratification had sometimes a positive effect on the artificial test case and was never detrimental. Nonetheless, on the realistic problem, pairing provided a great improvement, much more than stratification. Pairing and stratification are not totally black box; however, implementing stratification and pairing is usually easy and fast and we could do it easily on our realistic problem. We tested an intermediate algorithm with a parameter for switching smoothly from fully paired noisy optimization to totally unpaired noisy optimization. However, this parametrized algorithm (intermediate values of β) was not clearly better than the fully unpaired algorithm ($\beta = \infty$). It was not more robust in the case $\alpha = 0$, unless β is so large that there is essentially no pairing at all. As a conclusion, we firmly recommend common random numbers for population-based noisy optimization. Realistic counter-examples to CRN's efficiency would be welcome - we had such detrimental effects only in artificially built counter-example. There are probably cases (e.g. problems with rare critical cases) in which stratification also helps a lot, though this was not established in our application (which does not have natural strata).

Part III

Contributions in adversarial portfolios

8 Contributions to adversarial portfolios: outline

Adversarial problems, by nature, lead to combined solutions; a Nash equilibrium is typically a combination of pure strategies. This makes a deep difference with portfolios in the case of optimization or noisy optimization. Chapter 9 discusses the computation of the optimal combination when the solution is sparse. Chapter 10 proposes an application of this work to power systems. Chapter 11 then presents a solution for generating several algorithms, namely the random seed trick, to be combined by a portfolio. Chapter 12 extends the approach, in the sense that the combination are computed in a position-specific manner.

9 Sparse Nash equilibria

9.1 Algorithms for sparse bandits

In this section we define the problem and our proposed approach. Section 2.5.3 has presented the notion of NE in matrix games. Section 2.5.3.3 defines a classical bandit algorithm framework that aims at solving such a problem and Section 9.1.1 introduces two proposed algorithms adapted for sparse problems.

Throughout the chapter, $[[a, b]]$ denotes $\{a, a+1, a+2, \ldots, b-1, b\}$ and $lcm(x_1, \ldots, x_n)$ denotes the least common multiple of $x_1, \ldots, x_n$, where $a, b, x_1, \ldots, x_n$ are integers.

9.1.1 Rounded bandits.

[Flory and Teytaud, 2011] proposes a generic way of adapting bandits to the sparse case. Basically, the bandit runs as usual, and then the solution is pruned. We here propose a variant of their algorithm, as explained in Algorithm 6. Our definition of sparsity does not assume that the matrix M is sparse; rather, we assume that the two vectors of the NE x^* and y^* have support of moderate size k, i.e. $k << K$. This assumption certainly holds for many games.

The version in [Flory and Teytaud, 2011] is based on truncations (components less than a given ϵ are removed), instead of roundings; such a version is presented in Algorithm 7 (the exact solving after truncation was not in [Flory and Teytaud, 2011]). The drawback is that for an exact solving we need a second step of exact solving, but we will see that the resulting complexity is better than the complexity of the rounded bandit (Algorithm 6) because in case of rounding we need a higher precision in the bandit part.

Algorithm 6 RBANDIT, the rounded bandit algorithm. $\delta \in]0,1[$ is a parameter.

Input: A $K \times K$ matrix M, defined by a mapping $(i,j) \mapsto M_{i,j}$.

1: **Sparsity assumption:** We assume that the NE (x^*, y^*) is unique and sparse, i.e. x^* and y^* have support of size $\leq k$.
2: Define $p = 1/(4ck^k)$, where $c = lcm(1, 2, \ldots \lfloor k^k/2 \rfloor)$.
3: Let q be such that on $K \times K$ matrices the bandit algorithm with q iterations provides a p-NE with probability $1 - \delta$.
4: Run the bandit algorithm for q iterations.
5: Let (x, y) be the resulting approximate NE.
6: Approximate by multiples of $1/c$ as follows:

$$\begin{aligned} x'_i &= \lfloor cx_i + \frac{1}{2} \rfloor / c \\ y'_i &= \lfloor cy_i + \frac{1}{2} \rfloor / c. \end{aligned}$$

7: Renormalize: $x'' = x' / \sum_i x'_i$; $y'' = y' / \sum_i y'_i$.
8: Output (x'', y'') as an approximate NE.

Algorithm 7 TBANDIT, the truncated bandit algorithm. $\delta \in]0,1[$ is a parameter.

Input: A $K \times K$ matrix M, defined by a mapping $(i,j) \mapsto M_{i,j}$.

1: **Sparsity assumption:** We assume that the Nash equilibrium (x^*, y^*) is unique and sparse, i.e. x^* and y^* have support of size $\leq k$.
2: Define $p = 1/(4ck^k)$, where $c = \lceil k^{k/2} \rceil$.
3: Let q be such that on $K \times K$ matrices the bandit algorithm with q iterations provides a p-NE with probability $1 - \delta$.
4: Run the bandit algorithm for q iterations.
5: Consider the game restricted to rows i such that $x_i > 1/(2c)$ and to columns j such that $y_j > 1/(2c)$.
6: Let (x', y') be the exact Nash equilibrium of this restricted game (computed in polynomial time by linear programming).
7: Output (x', y') as approximate Nash equilibrium of the complete game (complete with 0's for missing coordinates).

[Flory and Teytaud, 2011] gives no proof and does not provide any tool for choosing c. The aim of this chapter is to show that our versions (Algorithm 6 and 7, i.e. rounded/truncated bandits) find exact NE, faster than existing methods when the solution is sparse.

9.2 Analysis

This section is devoted to the mathematical analysis of sparse bandit algorithms. Section 9.2.1 introduces the useful notations. Section 9.2.2 shows some properties of supports of Nash equilibria. Section 9.2.3 gives some useful results on denominators of rational probabilities involved in Nash equilibria. Section 9.2.4 presents stability results (showing that in sparse bandits, good strategies are also close to the Nash equilibrium). Section 9.2.5 concludes by properties on sparse bandits algorithms.

9.2.1 Terminology

We use the classical terminology of game theory. We consider a Matrix Game M of size $K \times K$ as above. A pure strategy is an index $i \in [[1, K]]$. A mixed strategy is a vector of size K with non-negative coefficients summing to 1.

Let e_i be the vector $(0, 0, \dots, 0, 1, 0 \dots, 0)^T$ with a one only at the i^{th} position; by a slight abuse of notation we will use this notation independently of the dimension of the vector (i.e. e_i can be used both in $\mathbb{R}^{10}$ and $\mathbb{R}^{50}$).

Let Δ denote the set of probability vectors, that is, $\Delta = \{y : \sum_j y_j = 1 \text{ and } \forall j, y_j \geq 0\}$; this implicitly depends on the dimension of the vectors, we do not precise it since there will be no ambiguity. The *support* of a vector $y \in \Delta$ is the set of indices j such that $y_j > 0$. For short, $\sup_x$ (or $\sup_y$, $\inf_x$, $\inf_y$ equivalently) means supremum on $x \in \Delta$. The *value* of $y \in \Delta$ for M is $V(y) = \sup_x x^T M y$.

Recall that v denotes the value of the game M, that is, it satisfies

$$\forall x, y \in \Delta,\ x^T M y^* \leq v \leq (x^*)^T M y, \tag{9.1}$$

and $v = (x^*)^T M y^* = V(y^*)$ if (x^*, y^*) is a Nash equilibrium.

9.2.2 Supports of Nash equilibria

Here we consider general matrices A, M with real coefficients. The following lemma is well known (see [Gale and Tucker, 1951, Lemma 1 page 318]).

Lemma 1 (Farkas Lemma)**.** *There exists $y \geq 0$ satisfying $Ay = b$ if and only if there is no x such that $A^T x \geq 0$ and $x^T b < 0$.*

The following lemma is adapted from [Dantzig and Thapa, 2003]. We give the proof for the sake of completeness.

Lemma 2. *Let I be the set of column indices i such that for all optimal solutions x^* of the row player we have $(x^*)^T M e_i = v$. Then there exists an optimal solution y^* for the column player whose support is exactly I.*

Proof. First, we show that it is sufficient to prove that for any $i \in I$ there is an optimal solution $y^i \in \Delta$ such that $y^i_i > 0$. Then one considers any strictly convex combination of the y^i, for instance $y^* = \frac{1}{|I|} \sum_{i \in I} y^i$. The vector $y^* \in \Delta$ has support including I (because $y^i_i > 0$). On the other hand, y^* has support included in I (because by construction any optimal solution has support included in I).

So, it is sufficient to show that for any $i \in I$ there is an optimal y^i such that $y^i_i > 0$. We now prove this. Without loss of generality fix $i = 1 \in I$. Let us suppose that no optimal solution y of the column player has a positive coordinate $y_1 > 0$. In other words, the system

$$\begin{cases} \sum_i y_i & = & 1 \\ My & \leq & v\mathbb{1} \\ y_1 & > & 0 \\ y & \geq & 0 \end{cases}$$

has no solution, where $\mathbb{1}$ is the vector with all coefficients equal to 1. Equivalently, this means that the following system has no solution

$$\begin{cases} \sum_i y_i & = & 1 \\ (M - v\mathbb{1}_{K \times K})y & \leq & 0 \\ y_1 & > & 0 \\ y & \geq & 0 \end{cases}$$

where $\mathbb{1}_{K \times K}$ is the $K \times K$ matrix with all coefficients equal to 1. Introducing the variable $z = \frac{y}{y_1}$ and a slack variable w of size $K \times 1$, this

is equivalent to saying that the system

$$\begin{cases} (M - v\mathbb{1}_{K\times K})z + w & = & 0 \\ z_1 & = & 1 \\ z, w & \geq & 0 \end{cases}$$

has no solution. By Lemma 1, applied with the concatenation (z, w) as y, $b = (0, 0, \ldots, 0, 1)$ and

$$A = \begin{pmatrix} M - v\mathbb{1}_{K\times K} & Id_K \\ 1\ 0\ \cdots \quad 0 & 0 \end{pmatrix},$$

we deduce the existence of a vector x and a real number ϵ such that

$$\begin{cases} x^T M_{\cdot 1} - v\sum_i x_i + \epsilon & \geq & 0 \\ x^T(M - v\mathbb{1}_{K\times K}) & \geq & 0 \\ x & \geq & 0 \\ \epsilon & < & 0 \end{cases}$$

where $M_{\cdot 1}$ denotes the first column of M.

By the first equation above, x is not zero. Thus we can normalize x to get a vector in Δ, and we infer the existence of an optimal strategy $x^* \in \Delta$ for the row player such that

$$(x^*)^T M_{\cdot 1} > v;$$

this implies that we cannot have $1 \in I$, a contradiction. □

Corollary 5. *If M admits a unique Nash Equilibrium (x^*, y^*) with support $J \times I$ then:*

- *for all $i \notin I$ and $j \notin J$ we have*

$$(x^*)^T M e_i > v \text{ and } e_j^T M y^* < v; \tag{9.2}$$

- *the submatrix M' of M with rows and columns respectively in J and I has a unique Nash Equilibrium which is the projection of x^*, y^* on $J \times I$.*

Proof. The first part is a consequence of Lemma 2. Indeed, if $(x^*)^T M e_i = v$ with $i \notin I$, then there exists an optimal solution y' whose support contains i, which contradicts the uniqueness of y^*. Thus $(x^*)^T M e_i > v$ by Equation (9.1). The statement for $j \notin J$ is symmetric.

For the second part, the projection is clearly a Nash equilibrium. Then, suppose that there is another Nash equilibrium for M' and let (x', y') be the only $2K$ vector whose projection on $J \times I$ is equal to this other equilibrium (in other words add zero coordinates for $i \notin I$ and $j \notin J$).

Consider now $(1-t)y^* + ty'$ with $t > 0$ and a row index j; then if $j \in J$

$$e_j^T M((1-t)y^* + ty') = (1-t)v + tv = v$$

and if $j \notin J$, then by the first part of this corollary the left part is at most v for t small enough. Since we have a finite number of rows this implies that by choosing t small enough we obtain a vector $(1-t)y^*+ty'$ which is another optimal solution for the column player in M, which contradicts the uniqueness. □

9.2.3 Denominators of Nash equilibria

Consider a matrix M, with coefficients in $\{0, 1\}$, of size $k_1 \times k_2$ with $k_1 \geq 2$ and $k_2 \geq 2$.

Lemma 3. *Assume that the Nash equilibrium (x^*, y^*) is unique and that $\forall i, j$, $x_i^* > 0$ and $y_j^* > 0$.*
a) Then $k_1 = k_2$ and x^* and y^* are rational vectors which can be written with common denominator at most $k^{k/2}$ with $k = k_1 = k_2$.
b) Moreover, for all $x, y \in \Delta$, $x^T M y^* = (x^*)^T M y = (x^*)^T M y^* = v$.

Proof. The Nash equilibrium y^* verifies the following properties:

- the sum of the probabilities of all strategies for the "column" player is 1, i.e.
$$\sum_i y_i^* = 1. \tag{9.3}$$
- the expected reward for the "row" player playing strategy i against y^* is independent of i, i.e.
$$\forall i, \sum_j M_{i,j} y_j^* = \sum_j M_{1,j} y_j^*. \tag{9.4}$$

If there is another solution $y \neq y^*$ to Equations (9.3), (9.4), then $y' = y^* - \alpha(y - y^*)$ is another strategy for the column player. If α is small enough, then $y' \geq 0$ and $\sum y_i' = 1$; it is a correct mixed strategy, and

its value is $x^*My^* - \alpha x^*M(y-y^*)$ which is less than or equal to x^*My^* if α has the same sign as $x^*M(y-y^*)$. This contradicts the uniqueness of y^* as a Nash strategy for the column player.

As a consequence, Equations (9.3) and (9.4) are a characterization of the unique Nash equilibrium. Thus y^* can be computed by solving Equations (9.3) and (9.4); this is a linear system $Zy^* = (1, 0, 0, \ldots, 0)$ with one single solution and Z a matrix with k_1 rows and k_2 columns, and where Z has values in $\{-1, 0, 1\}$. The solution is unique, therefore $k_1 = k_2$ and Z is invertible.

Z^{-1} can be computed as

$$Z^{-1} = \frac{1}{\det(Z)}(cofactor(Z))^T \quad \text{where} \quad (cofactor(Z))_{ij} = (-1)^{i+j} \det\left((Z_{i'j'})_{i' \neq i,\, j' \neq j}\right).$$

The matrix Z has coefficients in $\{-1, 0, 1\}$; therefore, by Hadamard's maximum determinant problem: $|\det(Z)| \leq k^{k/2}$ ([Hadamard, 1893], see e.g. [Brenner and Cummings, 1972]). Moreover, the matrix $cofactor(Z)$ has integer coefficients. This concludes the proof of the fact that y^* is rational with denominator $D = |\det(Z)|$ at most $k^{k/2}$, where $k = k_1 = k_2$. The same arguments using x^* instead of y^* show that x^* can also be written as a rational with the same denominator D.

To show b), notice that Equation (5.9) can be rewritten as follows: $\forall i, (My^*)_i = (My^*)_1$. If $x \in \Delta$, then

$$x^T My^* = \sum_i x_i (My^*)_i = \sum_i x_i (My^*)_1 = (My^*)_1 \text{ because } \sum_i x_i = 1.$$

Thus $x^T My^*$ is independent of $x \in \Delta$, which implies that $x^T My^* = (x^*)^T My^* = v$. By symmetry, one also has: $\forall y \in \Delta$, $(x^*)^T My = (x^*)^T My^* = v$. □

Please note that $k^{k/2}$ is known nearly optimal for matrices with coefficients in $\{0, 1\}$ (by Hadamard's work) for any k of the form 2^m. Also there are examples of matrices for which $V(y) = V(y^*) + \frac{1}{|\det(Z)|}\|y - y^*\|$ for y arbitrarily close to y^*.

9.2.4 Stability of Nash equilibria

In the general case of a zero-sum game, two mixed strategies can be far from each other, whenever both of them are very close, in terms of performance, to the performance of the (assumed unique) Nash equilibrium. However, with a matrix M with values in $\{0, 1\}$, this is not true anymore, as explained by the two lemmas below.

Lemma 4. *Let $k \geq 2$. Consider a $k \times k$ matrix M with elements in $\{0,1\}$ such that the Nash equilibrium (x^*, y^*) is unique and no pure strategy has a null weight. Then for all $y \in \Delta$ we have*

$$V(y) \geq V(y^*) + \frac{1}{k^{k/2}} \|y - y^*\|_\infty. \tag{9.5}$$

Proof. By convexity of V, it is sufficient to prove that

$$\min_{u;\sum_i u_i=0,\, \|u\|_\infty=1} \lim_{t\to 0,\, t>0} \frac{V(y^*+tu) - V(y^*)}{t} \geq 1/k^{k/2}.$$

This is equivalent to

$$\min_{u;\sum_i u_i=0,\, \|u\|_\infty=1} \max_i M_i.u \geq 1/k^{k/2}, \tag{9.6}$$

with M_i the i^{th} row of M as previously.

Let $\widetilde{u}$ be a vector in which the minimum is reached (it exists by compactness). Since $\|\widetilde{u}\|_\infty = 1$, there exists i_0 such that $|\widetilde{u}_{i_0}| = 1$. Let us assume without loss of generality, that $\widetilde{u}_{i_0} = 1$. The proof is the same if $\widetilde{u}_{i_0} = -1$. Thus, in Equation (9.6), we can restrict to the vectors u such that $u_{i_0} = 1$, $\sum_i u_i = 0$ and $\forall i \in \{1,\dots,k\}$, $-1 \leq u_i \leq 1$.

This is indeed a linear programming problem, as follows:

$$\min_{u\in\mathbb{R}^k,\, w\in\mathbb{R}} w$$

under constraints

$$\begin{aligned}
\forall i \in \{1,\dots,k\}, \quad & -1 \leq u_i \leq 1,\\
\forall i \in \{1,\dots,k\}, \quad & M_i.u \leq w,\\
& u_{i_0} = 1,\\
& \sum_{1\leq i\leq k} u_i = 0.
\end{aligned}$$

It is known that when a linear problem in dimension $k+1$ has a non infinite optimum, then there is a solution (u, w) with $k+1$ linearly independent active constraints. Let us pick such a solution $\overline{u}$. It is solution of $k+1$ linearly independent equations of the form either $u_i = 1$, or $u_i = -1$, or $M_i.u = w$, or $\sum_i u_i = 0$. Let us note the system

$$\begin{aligned}
\forall i \in P,\ u_i &= 1,\\
\forall i \in N,\ u_i &= -1,\\
\forall i \in H,\ M_i.u &= w,\\
\sum_{1\leq i\leq k} u_i &= 0,
\end{aligned}$$

where P, N, H are the subsets of $\{1, \ldots, k\}$ where the corresponding constraints are active.

We can remove w by setting $M_j.u = M_i.u$ for some fixed $i \in H$ and all $j \in H \setminus \{i\}$. Then, $\overline{u}$ is solution of a system of k equations in dimension k, with coefficients in $\{-1, 0, 1\}$.

We use the same trick as in the proof of Lemma 3; $\overline{u}$ is solution of a system of k linear equations with all coefficients in $\{-1, 0, 1\}$. Therefore all coordinates of $\overline{u}$ are rational numbers with a common denominator $D \leq k^{k/2}$. Then $M_i.\widetilde{u} = M_i.\overline{u}$ has a denominator $D \leq k^{k/2}$ and is positive; therefore $M_i.\widetilde{u} \geq 1/k^{k/2}$. This proves the expected result. □

Lemma 5. *Consider M a $K \times K$ matrix with coefficients in $\{0, 1\}$ and assume that the Nash equilibrium (x^*, y^*) is unique. Let J be the support of y^* and $k = \#J$. Then*

$$\forall j \notin J,\ (x^*)^T M e_j \geq v + \frac{1}{k^{\frac{k}{2}}}.$$

Proof. According to Lemma 3, v and the coefficients of x^*, y^* are multiple of some constant $c \geq \frac{1}{k^{\frac{k}{2}}}$. Thus, for every j, $(x^*)^T M e_j$ is also a multiple of c. Fix $j \notin J$. By Corollary 5, $(x^*)^T M e_j > v$, which implies that

$$(x^*)^T M e_j \geq v + c.$$

□

Combining Lemmas 4 and 5 yields the following

Theorem 3. *Consider a matrix M of size $K \times K$ and with coefficients in $\{0, 1\}$. Assume that there is a unique Nash equilibrium (x^*, y^*). Let k be the size of the supports of x^*, y^*. Then*

$$\forall y \in \Delta, V(y) - V(y^*) \geq \frac{1}{2k^k} \|y - y^*\|_\infty. \tag{9.7}$$

Proof. Define $c = \frac{1}{k^{\frac{k}{2}}}$. Let J be the support of y^*. For every $y \in \Delta$, one can write $y = ay' + by''$, with $a = \sum_{j \in J} y_i \in [0, 1]$, $a + b = 1$ and $y', y'' \in \Delta$ satisfying: $\forall j \notin J$, $y'_j = 0$ and $\forall j \in J$, $y''_j = 0$. For every index i, one has

$$y_i - y_i^* = \begin{cases} ay'_i - y_i^* = a(y'_i - y_i^*) - by_i^* & \text{if } i \in J \\ by''_i & \text{if } i \notin J \end{cases}$$

Thus

$$\|y - y^*\|_\infty = \max\{\|a(y' - y^*) - by^*\|_\infty, b\|y''\|_\infty\}. \tag{9.8}$$

Then, define $\delta = \|y - y^*\|_\infty$, $\delta_1 = \|y' - y^*\|_\infty$ and $\delta_2 = \|y''\|_\infty$. One has

$$V(y) \geq (x^*)^T My = a(x^*)^T My' + b\sum_{j \notin J}(x^*)^T M(y''_j e_j).$$

By Lemma 3(b), $(x^*)^T My' = v$; and by Lemma 5, $(x^*)^T Me_j \geq v + c$ for all $j \notin J$. Thus

$$V(y) \geq av + b(v + c)\sum_{j \notin J} y''_j,$$

that is,

$$V(y) - v \geq cb. \tag{9.9}$$

By Equation (9.8), either $\delta = b\delta_2 \leq b$, or $\delta = \|a(y' - y^*) - by^*\|_\infty$. If $\delta \leq b$, the result is given by Equation (9.9) because $c < 1$ and hence $c^2 \leq c$. From now on, assume that $\delta = \|a(y' - y^*) - by^*\|_\infty$. Then, $\delta \leq a\delta_1 + b\|y^*\|_\infty \leq a\delta_1 + b$. Equivalently,

$$a\delta_1 \geq \delta - b. \tag{9.10}$$

We split the end of the proof into two cases.

Case 1: $b \geq c\delta/2$. Then

$$\begin{aligned} V(y) - v &\geq cb \quad \text{by Equation (9.9)} \\ &\geq c^2\delta/2 \quad \text{by assumption on } b \end{aligned}$$

which gives the expected result in case 1.

Case 2: $b < c\delta/2$. Since y' has the same support as y^*, there exists $x' \in \Delta$ with the same support as x^* such that $x'My' - v \geq c\delta_1$ by Lemma 4. Moreover,

$$V(y) - v \geq x'My - v = a(x'My' - v) + b(x'My'' - v).$$

Hence,

$$\begin{aligned} V(y) - v &\geq ac\delta_1 - vb \quad (\text{because } x'My'' \geq 0) \\ &\geq c\delta(a\delta_1/\delta) - vb \\ &\geq c\delta\left(1 - \frac{b}{\delta}\right) - vb \quad (\text{using Equation (9.10)}) \\ &\geq c\delta\left(1 - \frac{c}{2} - \frac{v}{2}\right) \quad (\text{using } b < c\delta/2) \\ &\geq c\delta(1 - c)/2 \quad (\text{using } v \leq 1) \end{aligned}$$

Since $1 - c \geq c$, we get the expected result in case 2. □

9.2.5 Application to sparse bandit algorithms

Consider a matrix M as in Theorem 3. By Lemma 3, (x^*, y^*) can be written with a common denominator at most $k^{k/2}$. Define $C = lcm(1, 2, 3, \ldots, \lfloor k^{k/2} \rfloor)$. By the prime number theorem, it is known [1] that $C = O(\exp(k^{k/2}(1 + o(1))))$.

We discuss in parallel the truncated bandit algorithm (Algorithm 7) and the rounded bandit algorithm (Algorithm 6), as follows.

By construction of the algorithms, with probability $1-\delta$, the bandit algorithm finds a u-Nash equilibrium (x, y), for

- $u < 1/(4k^{k/2}k^k)$ for the TBANDIT algorithm.
- $u < 1/(4Ck^k)$ for the RBANDIT algorithm.

By Theorem 3, this implies that

- $\|x - x^*\|_\infty \leq 2uk^k < 1/(2k^{k/2})$ (idem for $\|y - y^*\|_\infty$) for TBANDIT;
- $\|x - x^*\|_\infty \leq 1/(2C)$ (idem for $\|y - y^*\|_\infty$) for RBANDIT.

Then:

- Truncated algorithm: all non-zero coordinates of x^* are at least $1/k^{k/2}$ and $|x_i^* - x_i| < 1/(2k^{k/2})$ with probability $\geq 1 - \delta$ (and the same for y^*, y); so with probability $1-\delta$ the Nash equilibrium (x', y') of the reduced game is the solution (x^*, y^*) (after filling missing coordinates with 0).
- Rounded algorithm: the denominator of the coordinates of x^*, y^* is a divisor of C, so with probability $1 - \delta$,

$$\|Cx - Cx^*\|_\infty < 1/2, \ \|Cy - Cy^*\|_\infty < 1/2,$$

and Cx^* and Cy^* are integers. So $x^* = \lfloor Cx + \frac{1}{2} \rfloor / C$; RBANDIT finds the exact solution with probability $\geq 1 - \delta$.

For example, if using the Grigoriadis & Khachiyan algorithm [Grigoriadis and Khachiyan, 1995], or variants of EXP3 [Bubeck et al., 2009], one can ensure precision u with fixed probability in time

[1] See details in `http://mathworld.wolfram.com/LeastCommonMultiple.html`.

- $O\left(K \log K \frac{1}{u}\right) = O\left(K(\log K)(k^{3k})\right)$ for the truncated version;
- $O\left(K \log K \frac{1}{u}\right) = O(K(\log K)k^{2k} \exp(2k^{k/2}(1 + o(1))))$ for the rounded version.

Then, we get, after rounding (rounded version) or after truncating and polynomial-time solving (truncated version), the exact solution y^* with fixed probability and time

- $O\left(K \log K \cdot k^{3k} + poly(k)\right)$ for the truncated algorithm (Algorithm 7);
- $O\left(K \log K \cdot k^{2k} \exp(2k^{k/2}(1 + o(1)))\right)$ for the rounded algorithm (Algorithm 6).

The truncated algorithm (Algorithm 7) is therefore better.

9.3 Experiments

We work on the Pokemon card game[2]. More precisely, we work on the *metagaming* part, i.e. the choice of the deck; the *ingaming* is then handled by a simulator with exact solving. The source code is freely available at `http://www.lri.fr/~teytaud/games.html`.

At first a normal EXP3 is executed using our empirically tuned formula

$$\begin{aligned} p(i) &= \left(1 + \frac{c-1}{\sqrt{t}}\right)^{-1} \\ &\times \left(1/(c \times \sqrt{t}) + (1 - 1/\sqrt{t}) \times \exp(S_i/\sqrt{t}) / \sum_j \exp(S_j/\sqrt{t})\right) \end{aligned} \tag{9.11}$$

to compute the probability of arm i and we normalize the probabilities if needs be. After T iterations, the TEXP3 takes the decision whether an arm is part of the NE is based upon a threshold ζ as explained in Algorithm 8. Algorithm 8 is based on Algorithm 7, with constants adapted empirically and without the exact solving at the end.

Figures 9.1(a), 9.1(c) and 9.1(e) show the performance of TEXP3 playing against EXP3 for different values of c and Figures 9.1(b), 9.1(d)

[2]See details in `https://en.wikipedia.org/wiki/Pokemon_Trading_Card_Game`.

Algorithm 8 TEXP3, an algorithm proposed in [Flory and Teytaud, 2011] used in these experiments.

Input: A $K \times K$ matrix M, defined by a mapping $(i, j) \mapsto M_{i,j}$. A number T of rounds.

1: Run EXP3 with Equation (9.11), which provides an approximation (x, y) of the Nash equilibrium.

2: Define:

$$\begin{aligned} \zeta &= \max_{a \in K} \frac{(Tx_a)^{0.7}}{T} \\ x'_i &= x_i \text{ if } x_i \geq \zeta \text{ and } x'_i = 0 \text{ otherwise.} \\ x''_i &= x'_i / \sum_{j \in \{1,\dots,K\}} x'_j. \end{aligned}$$

3: Define:

$$\begin{aligned} \zeta' &= \max_{a \in K} \frac{(Ty_a)^{0.7}}{T} \\ y'_i &= y_i \text{ if } y_i \geq \zeta' \text{ and } y'_i = 0 \text{ otherwise.} \\ y''_i &= y'_i / \sum_{j \in \{1,\dots,K\}} y'_j. \end{aligned}$$

Output: x'' and y'' as approximate Nash equilibrium of the complete game.

and 9.1(f) present the performance of TEXP3 and EXP3 when playing against the random uniform baseline; the probability distribution obtained by EXP3 and TEXP3 after T iterations of EXP3 and (for TEXP3) after truncation and re-normalization are used against random. To ensure that no player gains from being the first player, we make them play as both the row player and the column player and we display the result. Each point in the Figure consists in the means from 100 independent runs.

In all Figures, TEXP3 provides a consistent improvement over EXP3. Even in Figure 9.1(b), where EXP3 seems relatively weak against the random baseline, TEXP3 manages to maintain a performance similar to the ones in Figure 9.1(d) and 9.1(f).

9.4 Conclusion: sparsity makes the computation of Nash equilibria faster

[Grigoriadis and Khachiyan, 1995, Auer et al., 1995, Bubeck et al., 2009] are great steps forward in zero-sum matrix games, and beyond. They provide algorithms solving $K \times K$ matrix games, with precision ϵ and for a fixed confidence, in time $O(K \log K/\epsilon^2)$. As noticed in [Grigoriadis and Khachiyan, 1995], this has the surprising property that the complexity is sublinear in the size of the matrix, with a fixed risk.

We here show that, with coefficients in $\{0, 1\}$, if there is a unique sparse Nash equilibrium with support of size k for each player, then this bound can be reduced to $K \log K \cdot k^{3k}$, with no precision parameter (we provide an exact solution), with a fixed confidence $1 - \delta$:

- the dependency in K is the same as in [Grigoriadis and Khachiyan, 1995];
- there is no dependency in ϵ.

Practical relevance of this work

We want here to discuss the practical relevance of our results; two aspects are (i) the existence of very sparse problems, and (ii) the possible implementation of real world algorithms inspired by our results.

The first point is the existence of very sparse problems. We have seen that the sparsity level that we need, for our algorithm to outperform the state of the art for exact solutions, is $k << \log(K)/\log(\log(K))$. Obviously, one can design arbitrarily sparse zero-sum matrix games. Also, in real games, the quantity of moves worth being analyzed is usually much smaller than the number of legal moves; it is difficult to quantify this numerically for existing games as finding the meaningful strategies requires a lot of expertise and is difficult to quantify. For theoretical games, the classical centipede [Rosenthal, 1981] game does not have a unique Nash equilibrium; but if we remove strategies which make no sense (i.e. we do not consider the variants of a strategy after a move which finishes the game) the centipede game is highly sparse - only one strategy is in the Nash equilibrium, the one which immediately defects. The centipede game is not a zero-sum game, but zero-sum variants exist with the same sparsity property [Fey et al., 1996].

We also provide experimental results which show the relevance of this work in a real-world case. In particular, the TEXP3 modification, with its default parametrization from [Flory and Teytaud, 2011], performs better than the EXP3 counterpart for all tested values of the parameters. TEXP3 is not new, but the formal proof given here is new.

Extensions

The algorithm that we propose is based on the rounding of the solution given by EXP3 [Auer et al., 1995], or Grigoriadis' algorithm [Grigoriadis and Khachiyan, 1995], or INF [Bubeck et al., 2009]. Our algorithm works thanks to Theorem 3; any matrix game such that Theorem 3 is valid and such that Grigoriadis' algorithm, or EXP3 or INF works properly can be tackled similarly. In particular:

- Our bound in Theorem 3 (based on the constant in Lemma 3) can be adapted to the case of rational coefficients in M (instead of just 0 and 1);
- EXP3 and INF have no problem for stochastic cases as discussed below.

This leads to the following extensions:

- A first natural extension is the case of rational coefficients with a given denominator. The bound can be adapted to this case.
- Another extension is the case of stochastic versions of matrix games: i.e. if player 1 plays i and player 2 plays j, then the reward is a random variable $\widetilde{M}_{i,j}$, with expectation $M_{i,j}$, independently sampled at each trial. This does not change the result provided that $M_{i,j}$ verifies a condition as above, i.e. a common denominator bounded by some known integer.

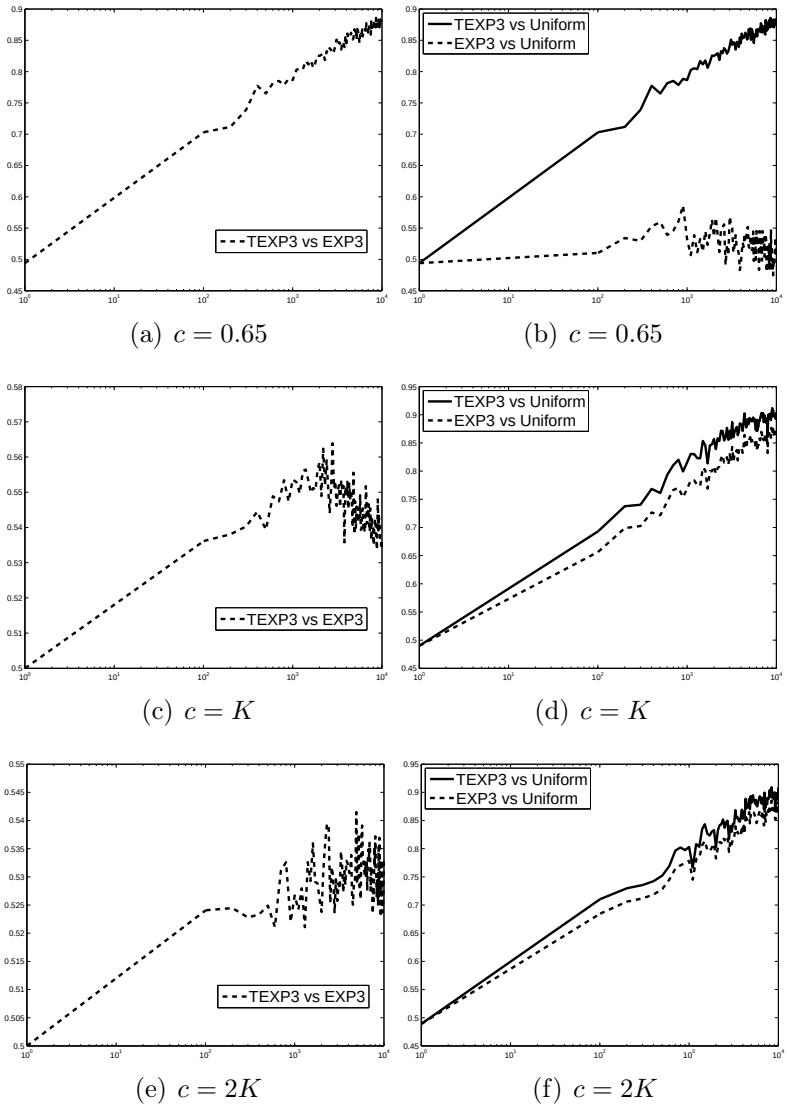
Second, we point out that in the real world, using sparsity provides substantial benefit. For example, [Flory and Teytaud, 2011, Chou et al., 2013] get benefits on a real-world internet card game using sparsity; their algorithms are similar to ours (for finding approximate results), but without the final step for finding an exact solution. We reproduce here their experiments in yet another game (Section 9.3) and get improvements far better than the theoretical bound. This suggests that, beyond the guaranteed improvement, there is a large improvement in many practical cases.

Further work

K is usually huge; an algorithm linear in K might be not practical. Algorithms with complexity $O(\sqrt{K})$, $O(\log(K))$ might be useful; we know that such algorithms can not do any approximation of a Nash equilibrium without additional assumptions, and maybe some regularity assumptions on the strategies of the row player and on the strategies of the column player are required for this. Such a framework (K huge but regularity assumptions on row strategies and regularity assumptions on column strategies) looks like a relevant framework for applications.

A distinct further work is the case in which we have no upper bound on the sparsity parameter k.

Finally, two assumptions might be partially removed: uniqueness of the Nash-equilibrium, and 0-sum nature of the game. In particular, extending to the case of a unique subgame perfect equilibrium looks like a promising direction.



(a) $c = 0.65$ (b) $c = 0.65$

(c) $c = K$ (d) $c = K$

(e) $c = 2K$ (f) $c = 2K$

Figure 9.1
Performance (%) in terms of budget T for the game of Pokemon using 2 cards. The left column shows TEXP3 playing against EXP3 for different values of c. The right column shows EXP3 and TEXP3 playing against the random uniform baseline. We tested a wide range of values for c and TEXP3 performs better than EXP3 regardless of c.

10 Nash-planning for scenario-based decision making

The optimization of power systems involves complex uncertainties, such as technological progress, political context, geopolitical constraints. Negotiations at COP21 are complicated by the huge number of scenarios that various people want to consider; these scenarios correspond to many uncertainties. These uncertainties are difficult to modelize as probabilities, due to the lack of data for future technologies and due to partially adversarial geopolitical decision makers. Tools for such difficult decision making problems include Wald and Savage criteria, possibilistic reasoning and Nash equilibria. In this chapter, we investigate the rationale behind the use of a two-player Nash equilibrium approach in such a difficult context; we show that the computational cost is indeed smaller than for simpler criteria. Moreover, it naturally provides a selection of decisions and scenarios, and it has a natural interpretation in the sense that Nature does not make decisions taking into account our own decisions.

As a summary, we get a fast criterion, faster than Wald or Savage criteria, with a natural interpretation. The algorithm naturally provides a matrix of results, namely the matrix of outcomes in the most interesting decisions and for the most critical scenarios. These decisions and scenarios are also equipped with a ranking.

The state-of-the art of decision making in uncertain environments is discussed in Section 2.4. Section 10.1 describes our proposed approach. In particular, Section 10.1.3 summarizes our method. Experiments are provided in Section 10.2. Section 10.3 concludes.

10.1 Our proposal: NashUncertaintyDecision

Our proposed tool is as follows:

- We use Nash equilibria, for their principled nature.
- We compute the equilibria thanks to adversarial bandit algorithms, as detailed in the next section.
- We use sparsity, for (i) reducing the computational cost (ii) reducing the number of pure strategies in our recommendation.

The resulting algorithm has the following advantage:

- It is fast; this is not intuitive, but Nash equilibria, in spite of the complex theories behind this concept, can be approximated quickly, without computing the entire matrix of $R(k, s)$. A pioneering work in this direction was [Grigoriadis and Khachiyan, 1995]; within logarithmic terms and dependency in the precision, the cost is roughly the square root of the size of the matrix.
- It naturally provides a submatrix of $R(k, s)$, for the best k and the most critical s.

We believe that such outcomes are natural tools for including in platforms for simulating large scale power systems involving huge uncertainties.

10.1.1 The algorithmic technology under the hood: computing Nash equilibria with adversarial bandit algorithms

For the computational cost issue for computing Nash equilibria, there exist algorithms reaching approximate solutions much faster than the exact linear programming approach [von Stengel, 2002]. Some of these fast algorithms are based on the bandit formalism. The Multi-Armed Bandit (MAB) problem [Lai and Robbins, 1985,Katehakis and Veinott Jr, 1987,Auer et al., 1995] is a model of exploration/exploitation trade-offs, aimed at optimizing the expected payoff. Let us define an adversarial multi-armed bandit with $K \in \mathbb{N}^+$ $(K > 1)$ arms and let $\mathcal{K}$ denote the set of arms. Let $\mathcal{T} = \{1, \ldots, T\}$ denote the set of time steps, with $T \in \mathbb{N}^+$ a finite time horizon. At each time step $t \in \mathcal{T}$, the algorithm chooses $i_t \in \mathcal{K}$ and obtains a reward $R_{i_t,t}$. The reward $R_{i_t,t}$ is a mapping $(\mathcal{K}, \mathcal{T}) \mapsto \mathbb{R}$.

Algorithm 9 Generic adversarial multi-armed bandit. The problem is described through the arm sets, a method and the *compute reward*, i.e. the mapping $(\mathcal{K}, \mathcal{T}) \mapsto \mathbb{R}$.

Input: a time horizon (computational budget) $T \in \mathbb{N}^+$
Input: a set of arms $\mathcal{K}$
Input: a probability distribution π on $\mathcal{K}$
1: **for** $t \leftarrow 1$ to T **do**
2: Select arm $i_t \in \mathcal{K}$ based upon π
3: *Get reward* $R_{i_t,t}$
4: Update the probability distribution π using $R_{i_t,t}$.
5: **end for**

The generic adversarial bandit is detailed in Algorithm 9. In the case of adversarial problems, when we search for a Nash equilibrium for a reward function $(k, s) \mapsto R(k, s)$, two bandit algorithms typically play against each other. One of them is Nature, and the other plays our role. At the end, our bandit algorithm recommends a (possibly mixed) strategy over the K arms. This recommended distribution is the empirical distribution of play during the games against the Nature bandit.

Such a fast approximate solution can be provided by $Exp3$ (Exponential weights for Exploration and Exploitation) [Auer et al., 2002a] and its $Exp3.P$ variant [Auer et al., 2002b], presented in Algorithm 10. $Exp3$ has the same efficiency as the Grigoriadis and Khachiyan method [Grigoriadis and Khachiyan, 1995] for finding approximate Nash equilibria, and can be implemented with two bandits playing one against each other, e.g. one for us and one for Nature. $Exp3.P$ is not anytime, it requires the time horizon in order to initialize some input meta-parameters. [Busa-Fekete et al., 2010] optimized Adaptive Boosting (AdaBoost), a popular machine-learning meta-algorithm, by the adversarial bandit algorithm $Exp3.P$, and proposed two parametrizations of the algorithm, as detailed in Table 10.1. [Bubeck and Cesa-Bianchi, 2012] proved a high probability bound on the weak reward of $Exp3.P$.

Algorithm 10 *Exp*3.*P*: variant of *Exp*3, proved to have a high probability bound on the weak reward. η and γ are two parameters.

Input: $\eta \in \mathbb{R}$
Input: $\gamma \in (0, 1]$
Input: a time horizon (computational budget) $T \in \mathbb{N}^+$
Input: $K \in \mathbb{N}^+$ is the number of arms

$y \leftarrow 0$
for $i \leftarrow 1$ to K **do** ▷ initialization
    $w_i \leftarrow \exp(\frac{\eta\gamma}{3}\sqrt{\frac{T}{K}})$
end for
for $t \leftarrow 1$ to T **do**
    for $i \leftarrow 1$ to K **do**
        $p_i \leftarrow (1-\gamma)\frac{w_i}{\sum_{j=1}^{K} w_j} + \frac{\gamma}{K}$
    end for
    Generate i_t according to $(p_1, p_2, \ldots, p_K)$
    Compute reward $R_{i_t,t}$
    for $i \leftarrow 1$ to K **do**
        if $i == i_t$ **then**
            $\hat{R}_i \leftarrow \frac{R_{i_t,t}}{p_i}$
        else
            $\hat{R}_i \leftarrow 0$
        end if
        $w_i \leftarrow w_i \exp\left(\frac{\gamma}{3K}(\hat{R}_i + \frac{\eta}{p_i\sqrt{TK}})\right)$
    end for
end for
return probability distribution $(p_1, p_2, \ldots, p_K)$

Table 10.1: Notations and parameters of algorithms using in the experiments. T is the horizon, i.e. simulation number. "$+t$" (resp. "$+p$") refers to the variant of $Exp3.P$ or $tExp3.P$ with theoretical (resp. practical) parametrization. ϵ is the precision. We use $\epsilon = 1e-6$ in our experiments.

Notation	Parameters of $Exp3.P$	
	η	γ
$Exp3.P+p$	0.3	0.15
$tExp3.P+p$		
$Exp3.P+t$	$2\sqrt{\log \frac{KT}{\epsilon}}$	$\min(0.6, 2\sqrt{\frac{3K\log(K)}{5T}})$
$tExp3.P+t$		

10.1.2 Another ingredient under the hood: sparsity

[Teytaud and Flory, 2011] proposed a truncation technique on sparse problem. Considering the Nash equilibria for two-player finite-sum matrix games, if the Nash equilibrium of the problem is sparse, the small components of the solution can be removed and the remaining submatrix is solved exactly. This technique can be applied to some adversarial bandit algorithm such as *Grigoriadis'* algorithm [Grigoriadis and Khachiyan, 1995], $Exp3$ [Auer et al., 2002a] or Inf [Bubeck et al., 2009]. The properties of this sparsity technique are as follows. Asymptotically in the computational budget, the convergence to the Nash equilibria is preserved [Teytaud and Flory, 2011]. The computation time is lower if there exists a sparse solution [Teytaud et al., 2014]. The support of the obtained approximation has at most the same number of pure strategies and often far less [Teytaud and Flory, 2011]. Essentially, we get rid of the random exploration part of the empirical distribution of play.

10.1.3 Overview of our method

We first give a high level view of our method, in Alg. 11. All the algorithmic challenge is hidden in the $tExp3.P$ algorithm, defined later. We now present the computation engine $tExp3.P$. We apply the truncation technique [Teytaud and Flory, 2011] to $Exp3.P$. We present in Algorithm 12 the resulting algorithm, denoted as $tExp3.P$.

Algorithm 11 The SNash (Sparse-Nash) algorithm for solving decision under uncertainty problems.

Input: A family $\{1, \ldots, K\}$ of possible decisions (investment policies).
Input: A family $\{1, \ldots, S\}$ of scenarios.
Input: A mapping $(k, s) \mapsto R_{k,s}$, providing
Run $tExp3.P$ on the mapping R, get a probability distribution on $\mathcal{K}$ and a probability distribution on $\mathcal{S}$.
Output $k_1, \ldots, k_m$ the policies with positive probability and $s_1, \ldots, s_p$ the scenarios with positive probability. Emphasize the policy with highest probability.
Output the matrix of $R(k_i, s_j)$.

10.2 Experiments

We propose a simple model of investments in power systems. Our model is not supposed to be realistic; it is aimed at being easy to reproduce.

10.2.1 Power investment problem

We consider each investment *policy*, sometimes called action or decision, a vector $\mathbf{k} = (C, F, X, S, W, P, T, U, N, A) \in \{0, \frac{1}{2}, 1\}^{10}$. A *scenario* is a vector $\mathbf{s} = (Z, WB, PB, TB, XB, UB, SB, CC, NT) \in \{0, \frac{1}{2}, 1\}^{9}$. Detailed descriptions of parameters are provided in Table 10.2. Let $\mathcal{S}$ be the set of possible scenarios and $\mathcal{K}$ be the set of possible policies. The utility function R is a mapping $(\mathcal{K}, \mathcal{S}) \mapsto \mathbb{R}$. Given decision $\mathbf{k} \in \mathcal{K}$ and scenario $\mathbf{s} \in \mathcal{S}$, a reward can be computed by

$$
\begin{aligned}
R(\mathbf{k}, \mathbf{s}) = \frac{2}{3}(1 + rand) \cdot (N(1 - Z)/5 \\
- cost \cdot (N + U + T + P + W + S + X + F + C) \\
+ 7XB \cdot X + W(1 + WB)(SB + \sqrt{S})/2 \\
+ 3P(PB + SB) - 4C \cdot CC - F \cdot NT \\
+ S(1 - Z) + P \cdot Z + U \cdot UB \\
+ T \cdot S \cdot (1 + TB - SB/2) \\
- F \cdot NT + A \cdot (1 + W + P - 2SB)).
\end{aligned}
$$

where *cost* is a meta-parameter.

Table 10.2: Parameters and descriptions of policy variables (vector $\mathbf{k}$) and scenario (vector $\mathbf{s}$) in power investment problem.

$\mathbf{k} \in \{0, \frac{1}{2}, 1\}$	Corresponding investment
C	Coal
F	Nuclear fission
X	Nuclear fusion
S	Supergrids
W	Wind power
P	PV units
T	Solar thermal
U	Unconventional renewable
N	Nanogrids
A	massive storage in Scandinavia
$\mathbf{s} \in \{0, \frac{1}{2}, 1\}$	Nature's action
Z	Massive geopolitical issues
WB	Wind power technological breakthrough
PB	PV Units breakthrough
TB	Solar thermal breakthrough
XB	Fusion breakthrough
UB	Unconventional renewable breakthrough
SB	Local storage breakthrough
CC	Climate change disaster
NT	Nuclear terrorism

This provides a reward function $R(\mathbf{k}, \mathbf{s})$, with which we can build a matrix R of rewards. However, with a ternary discretization for each variable we get a huge matrix, that we will not construct explicitly - more precisely, it would be impossible to construct it explicitly with a real problem involving hours of computation for each $R(\mathbf{k}, \mathbf{s})$. Fortunately, approximate algorithms can solve Nash equilibria with precision ϵ with $O(K \log(K)/\epsilon^2)$ requests to the reward function, i.e. far less than the quadratic computation time K^2 needed for reading all entries in the matrix. We do experiments on this investment problem and apply the algorithms described in Table 10.1. We consider policies and scenarios in discrete domains: $\mathcal{K} = \{0, \frac{1}{2}, 1\}^{10}$, $\mathcal{S} = \{0, \frac{1}{2}, 1\}^9$. The reward matrix $R_{3^{10} \times 3^9}$ can be defined by $\forall i \in \{1, \ldots, 3^{10}\}$, $\forall j \in \{1, \ldots, 3^9\}$, $R_{i,j} = R(\mathbf{k}_i, \mathbf{s}_j)$, but the reward is noisy as previously mentioned, where $\mathbf{k}_i$

denotes the i^{th} policy in $\mathcal{K}$ and $\mathbf{s}_i$ denotes the j^{th} scenario in $\mathcal{S}$. Thus, each line of the matrix is a possible policy and each column is a scenario, $R_{i,j}$ is the reward obtained by apply the policy $\mathbf{k}_i$ to the scenario $\mathbf{s}_j$. Experiments are performed for different numbers of time steps in the bandit algorithms, i.e. we consider T simulations for each

$$T \in \{1, 2, 8, 10, 32, 128, 512, 2048\} \times \lceil 3^{10}/10 \rceil.$$

Thus when playing with the "theoretical" parametrization, for each T, the input meta-parameters η and γ are different, as they depend on the budget T. In the entire paper, when we show an expected reward $R(\mathbf{k}, \mathbf{s})$ for some $\mathbf{s}$ and for $\mathbf{k}$ learned by one of our methods, we refer to 10000 trials; $R(\mathbf{k}, \mathbf{s})$ are played for 10000 randomly drawn pairs $(\mathbf{k}_{i_n}, \mathbf{s}_{j_n})$ i.i.d. according to the random variables i_n and j_n proposed by the considered policies. The performance is the average reward of these 10000 trials $R(\mathbf{k}_{i_1}, \mathbf{s}_{j_1}), \ldots, R(\mathbf{k}_{i_{10000}}, \mathbf{s}_{j_{10000}})$. There is an additional averaging, over learning. Namely, each learning (i.e. the sequence of $Exp3$ iteration for approximating a Nash equilibrium) is repeated 100 times. The meta-parameter *cost* is set to 1 in our experiments.

We use the parametrizations of variants of $Exp3.P$ presented in Table 10.1. [Teytaud and Flory, 2011] proposed $\alpha = 0.7$ as truncation parameter in truncated $Exp3.P$ and [Teytaud et al., 2014] used the same value. The sparsity level, as well as the performance, are given in Table 10.3.

We observe that when the number of simulations is bigger than the cardinality of the search domain, i.e. the number of possible pure policies, then $\alpha \simeq 0.9$ leads to better empirical mean reward against the uniform policy. Values between 0.5 and 1 are the best ones. When learning with few simulations ($5905 = \lceil K/10 \rceil$), the non-truncated solutions and non-sparse solutions are as weak as a random strategy. Along with the increment of simulation times, the non-truncated solutions and non-sparse solutions become stronger, but still weaker than the truncated solutions. When we use the truncation, we get significant mean reward even with a small horizon, i.e. the $tExp3.P + t$ succeeds in finding better and "purer' policies than $Exp.3$.

10.2.1.1 The parameters of $Exp3.P + t$

When learning with few simulations ($5905 = \lceil K/10 \rceil$), the non-truncated solutions and non-sparse solutions are as weak as a random strategy.

Table 10.3: In these tables, the result is the average value of 100 learnings. The standard deviation is shown after ±. "simul." refers to "simulation number", i.e. horizon. **Top:** Empirical mean reward obtained using different truncation parameter α. **Middle:** Average sparsity level (over $3^{10} = 59049$ arms), i.e. number of pure policies in the support of the obtained approximation, of solutions provided by $Exp3.P + t$ in power investment problem. "non-truncated" means that all elements of the solution provided are below the threshold ζ (line 2 in Algorithm 12); "non-sparse" means that all elements of the solution provided are above the threshold ζ. In both cases, we play with the original solution provided by $Exp3.P + t$. **Bottom:** Exploitability using different truncation parameter α for solutions provided by $Exp3.P + t$ in power investment problem. In this table, we exclude the solutions which can not be truncated, i.e. all elements are below or all elements are above the truncation threshold ζ. If at least one solution is excluded, we show between parenthesis the number of solutions excluded from the 100 learning. When $\alpha = 1.0$, $\zeta = \max_{i \in \{1,\dots,K\}} p_i$. Thus, the remaining policy (policies), after truncation, is the element with the highest probability.

α	**Empirical mean reward**							
	$\lceil K/10 \rceil$ simul.	$2\lceil K/10 \rceil$ simul.	$8\lceil K/10 \rceil$ simul.	K simul.	$32\lceil K/10 \rceil$ simul.	$128\lceil K/10 \rceil$ simul.	$512\lceil K/10 \rceil$ simul.	$2048\lceil K/10 \rceil$ simul.
0.1	-.629 ± .067	4.510 ± .119	2.595 ± .006	2.174 ± .006	-.029 ± .004	1.050 ± .004	2.184 ± .005	4.105 ± .006
0.3	-.622 ± .067	4.572 ± .125	3.299 ± .010	3.090 ± .009	2.195 ± .017	3.892 ± .018	6.555 ± .008	6.822 ± .004
0.5	-.627 ± .067	4.615 ± .130	3.896 ± .016	3.779 ± .015	3.592 ± .016	5.275 ± .020	6.741 ± .007	6.853 ± .004
0.7	-.629 ± .067	4.651 ± .135	4.501 ± .030	4.454 ± .027	4.674 ± .022	6.101 ± .016	6.777 ± .007	6.858 ± .005
0.9	-.631 ± .067	4.711 ± .138	5.021 ± .058	5.149 ± .062	5.703 ± .040	6.536 ± .017	6.813 ± .007	6.873 ± .004
1	4.288 ± .188	4.249 ± .212	4.853 ± .158	5.027 ± .143	5.709 ± .101	6.137 ± .163	6.413 ± .136	6.844 ± .028
1.4	4.513 ± .115	-.472 ± .107	-.478 ± .004	-.434 ± .004	-.022 ± .004	1.052 ± .005	2.182 ± .006	4.121 ± .005
2.8	2.939 ± .142	-.477 ± .103	-.480 ± .004	-.442 ± .004	-.022 ± .004	1.053 ± .004	2.185 ± .005	4.106 ± .006
5.6	.014 ± .153	-.502 ± .090	-.486 ± .004	-.434 ± .003	-.025 ± .004	1.051 ± .005	2.188 ± .006	4.119 ± .005

α	**Average sparsity level over $3^{10} = 59049$ arms**							
	$\lceil K/10 \rceil$ simul.	$2\lceil K/10 \rceil$ simul.	$8\lceil K/10 \rceil$ simul.	K simul.	$32\lceil K/10 \rceil$ simul.	$128\lceil K/10 \rceil$ simul.	$512\lceil K/10 \rceil$ simul.	$2048\lceil K/10 \rceil$ simul.
0.1	0.01 ± 0.01	2.62 ± 0.14	1873.70 ± 10.87	4621.87 ± 28.34	non-truncated	non-truncated	non-truncated	non-truncated
0.3	0.01 ± 0.01	2.09 ± 0.11	491.53 ± 7.74	955.32 ± 13.78	12140.13 ± 234.46	7577.95 ± 154.37	710.45 ± 11.28	320.43 ± 2.91
0.5	0.01 ± 0.01	1.57 ± 0.08	126.18 ± 3.71	216.63 ± 5.53	1502.24 ± 33.85	687.42 ± 19.37	33.01 ± 1.14	10.16 ± 0.27
0.7	0.01 ± 0.01	1.20 ± 0.06	24.80 ± 1.23	36.69 ± 1.69	168.02 ± 6.94	63.04 ± 2.49	6.57 ± 0.27	2.59 ± 0.11
0.9	0.01 ± 0.01	1.00 ± 0.02	3.54 ± 0.23	3.73 ± 0.26	7.35 ± 0.49	5.12 ± 0.29	1.93 ± 0.09	1.17 ± 0.04
1.0	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
1.4	4.31 ± 0.48	0.03 ± 0.02	non-sparse	non-sparse	non-sparse	non-sparse	non-sparse	non-sparse
2.8	3131.10 ± 1248.08	0.08 ± 0.05	non-sparse	non-sparse	non-sparse	non-sparse	non-sparse	non-sparse
5.6	46543.91 ± 2384.93	0.33 ± 0.25	non-sparse	non-sparse	non-sparse	non-sparse	non-sparse	non-sparse

α	**Exploitability indicator: worst score against pure strategies**							
	$\lceil K/10 \rceil$ simul.	$2\lceil K/10 \rceil$ simul.	$8\lceil K/10 \rceil$ simul.	K simul.	$32\lceil K/10 \rceil$ simul.	$128\lceil K/10 \rceil$ simul.	$512\lceil K/10 \rceil$ simul.	$2048\lceil K/10 \rceil$ simul.
0.5	$-0.598(99)$	$-1.733 \pm 0.108(3)$	-5.560 ± 0.070	-5.693 ± 0.058	-5.725 ± 0.060	-3.479 ± 0.061	-0.576 ± 0.041	0.056 ± 0.024
0.7	$-0.598(99)$	$-1.498 \pm 0.101(3)$	-4.028 ± 0.094	-4.132 ± 0.094	-4.038 ± 0.074	-1.243 ± 0.032	0.010 ± 0.018	0.268 ± 0.011
0.9	$-0.598(99)$	$-1.363 \pm 0.102(3)$	-2.012 ± 0.107	-1.859 ± 0.115	-1.369 ± 0.081	-0.195 ± 0.028	0.272 ± 0.011	0.330 ± 0.003
1.0	-1.560 ± 0.123	-1.327 ± 0.101	-0.938 ± 0.078	-0.971 ± 0.092	-0.455 ± 0.060	0.182 ± 0.021	0.323 ± 0.005	$\mathbf{0.333 \pm 0.000}$

Along with the increment of simulation times, the non-truncated solutions and non-sparse solutions become stronger, but still weaker than the truncated solutions. Sparsity level "0.01" means that one and only one solution of the 100 learnings has one element above the threshold ζ, the other 99 solutions of the 99 learnings have no element above the threshold ζ. This situation is not far from the non-truncated or non-sparse case. If the solution is sparse, we get a better empirical mean reward even with a small horizon, i.e. the $tExp3.P + t$ succeeds in finding better pure policies.

We see that truncated algorithms outperform their non-truncated counterparts, in particular, truncation clearly shows its strength when the number of simulations is small in front of the size of search domain.

10.2.2 A modified power investment problem

Now we modify the reward function by adding the term in bold, a reward can be computed by

$$R'(\mathbf{k}, \mathbf{s}) = R(\mathbf{k}, \mathbf{s}) + \mathbf{c} \cdot ((\mathbf{X} == \mathbf{XB}) + (\mathbf{C} == \mathbf{CC}) + (\mathbf{NT} == \mathbf{F}) + (\mathbf{P} == \mathbf{PB})).$$

where *cost* and c are meta-parameters.

As presented in the previous section, we can build a matrix R' with the reward function $R'(\mathbf{k}, \mathbf{s})$. We do experiments on this modified investment problem and apply the algorithms described in Table 10.1. We consider policies and scenarios in discrete domains as used in the previous section. The meta-parameter *cost* is set to 1 in our experiments. The meta-parameter c is set to $\{1, 2, \ldots, 10\}$ in our experiments.

We present the results with $c = 1$ and $c = 10$ in Tables 10.4 and 10.5:

- in both testcases, $\alpha = 0.7$ (recommended by previous work) is always better than the baseline, to which the truncation technique is not applied;
- for the testcase with $c = 1$, $\alpha = 0.9$ outperforms the other values of α at most of time;
- in both testcases, $\alpha = 0.9$ does not provide good results when $T = K$;
- when the budget is big, $\alpha = 0.99$ provides better results.

Table 10.4: Results for reward matrix R' computed with $c = 1$. In these tables, the result is the average value of 100 learnings. The standard deviation is shown after ±. "NT" means that the truncation technique is not applied; "non-sparse" means that all elements of the solution provided are above the threshold ζ. **Top:** Average sparsity level (over $3^{10} = 59049$ arms), i.e. number of pure policies in the support of the obtained approximation, of solutions provided by $Exp3.P+t$ in power investment problem. **Middle:** Proxy exploitability (to be maximized) using different truncation parameter α for solutions provided by $Exp3.P+t$ in power investment problem. The proxy exploitability is the difference between the best robust score in the table, minus the robust score. **Bottom:** Robust score (to be minimized) using different truncation parameter α for solutions provided by $Exp3.P + t$ in power investment problem. The robust score is the worst of the scores against pure policies.

α	**Average sparsity level over $3^{10} = 59049$ arms**					
	$T = K$	$T = 10K$	$T = 50K$	$T = 100K$	$T = 500K$	$T = 1000K$
0.1	13804.380 ± 52.015	non-sparse	non-sparse	non-sparse	non-sparse	non-sparse
0.3	2810.120 ± 59.083	non-sparse	non-sparse	non-sparse	non-sparse	non-sparse
0.5	395.920 ± 15.835	non-sparse	non-sparse	59048.960 ± 196.946	49819.430 ± 195.016	non-sparse
0.7	43.230 ± 2.624	58925.340 ± 26.821	55383.140 ± 150.057	46000.020 ± 277.653	9065.180 ± 159.610	non-sparse
0.9	3.600 ± 0.260	992.940 ± 64.474	796.500 ± 41.724	503.600 ± 24.927	97.670 ± 5.445	52632.820 ± 522.505
0.99	1.110 ± 0.031	2.250 ± 0.171	2.500 ± 0.180	2.310 ± 0.156	1.790 ± 0.121	6.700 ± 0.612

α	**Proxy exploitability**					
	$T = K$	$T = 10K$	$T = 50K$	$T = 100K$	$T = 500K$	$T = 1000K$
NT	1.369e-02	2.092e-02	1.946e-02	1.492e-02	2.669e-02	4.525e-02
0.1	1.109e-02	2.092e-02	1.946e-02	1.492e-02	2.669e-02	4.525e-02
0.3	5.485e-03	2.092e-02	1.946e-02	1.492e-02	2.669e-02	4.525e-02
0.5	**0.000e+00**	2.092e-02	1.946e-02	1.492e-02	2.454e-02	4.525e-02
0.7	4.328e-04	2.091e-02	1.854e-02	1.083e-02	1.705e-02	4.525e-02
0.9	7.778e-02	**0.000e+00**	**0.000e+00**	**0.000e+00**	**0.000e+00**	4.304e-02
0.99	1.425e-01	7.806e-02	5.385e-02	4.564e-02	2.456e-02	**0.000e+00**
Pure	1.554e-01	1.191e-01	8.638e-02	6.503e-02	3.443e-02	5.537e-02

α	**Robust score**					
	$T = K$	$T = 10K$	$T = 50K$	$T = 100K$	$T = 500K$	$T = 1000K$
NT	4.922e-01 ± 5.649e-07	4.928e-01 ± 1.787e-06	4.956e-01 ± 4.016e-06	4.991e-01 ± 5.892e-06	5.221e-01 ± 1.404e-05	4.938e-01 ± 1.687e-06
0.1	4.948e-01 ± 5.739e-05	4.928e-01 ± 1.787e-06	4.956e-01 ± 4.016e-06	4.991e-01 ± 5.892e-06	5.221e-01 ± 1.404e-05	4.938e-01 ± 1.687e-06
0.3	5.004e-01 ± 1.397e-04	4.928e-01 ± 1.787e-06	4.956e-01 ± 4.016e-06	4.991e-01 ± 5.892e-06	5.221e-01 ± 1.404e-05	4.938e-01 ± 1.687e-06
0.5	**5.059e-01 ± 2.272e-04**	4.928e-01 ± 1.787e-06	4.956e-01 ± 4.016e-06	4.991e-01 ± 5.891e-06	5.242e-01 ± 5.491e-05	4.938e-01 ± 1.687e-06
0.7	5.054e-01 ± 1.327e-03	4.928e-01 ± 3.835e-06	4.965e-01 ± 3.896e-05	5.031e-01 ± 1.016e-04	5.317e-01 ± 9.573e-05	4.938e-01 ± 1.687e-06
0.9	4.281e-01 ± 6.926e-03	**5.137e-01 ± 4.199e-04**	**5.151e-01 ± 5.007e-04**	**5.140e-01 ± 4.965e-04**	**5.487e-01 ± 9.413e-04**	4.960e-01 ± 1.828e-04
0.99	3.634e-01 ± 8.191e-03	4.357e-01 ± 6.873e-03	4.612e-01 ± 5.380e-03	4.683e-01 ± 4.834e-03	5.242e-01 ± 3.302e-03	**5.390e-01 ± 3.167e-03**
Pure	3.505e-01 ± 7.842e-03	3.946e-01 ± 7.181e-03	4.287e-01 ± 6.203e-03	4.489e-01 ± 5.410e-03	5.143e-01 ± 3.597e-03	4.837e-01 ± 5.558e-03

Table 10.5: Results for reward matrix R' computed with $c = 10$. In these tables, the result is the average value of 100 learnings. The standard deviation is shown after ±. "NT" means that the truncation technique is not applied; "non-sparse" means that all elements of the solution provided are above the threshold ζ. **Top:** Average sparsity level (over $3^{10} = 59049$ arms), i.e. number of pure policies in the support of the obtained approximation, of solutions provided by $Exp3.P+t$ in power investment problem. **Middle:** Proxy exploitability (to be minimized) using different truncation parameter α for solutions provided by $Exp3.P+t$ in power investment problem. The proxy exploitability is the difference between the best robust score in the table, minus the robust score. **Bottom:** Robust score (to be maximized) using different truncation parameter α for solutions provided by $Exp3.P + t$ in power investment problem. The robust score is the worst of the scores against pure policies.

α	**Average sparsity level over $3^{10} = 59049$ arms**					
	$T = K$	$T = 10K$	$T = 50K$	$T = 100K$	$T = 500K$	$T = 1000K$
0.1	6394.625 ± 84.308	non-sparse	non-sparse	non-sparse	non-sparse	non-sparse
0.3	1337.896 ± 40.491	non-sparse	non-sparse	non-sparse	non-sparse	non-sparse
0.5	206.146 ± 12.647	non-sparse	non-sparse	non-sparse	non-sparse	non-sparse
0.7	25.563 ± 2.045	non-sparse	non-sparse	non-sparse	59048.750 ± 0.250	non-sparse
0.9	3.729 ± 0.353	42616.313 ± 1476.644	47581 ± 1015.506	38361.182 ± 1091.373	4510.125 ± 726.595	58323.125 ± 157.971
0.99	1.208 ± 0.072	4.479 ± 0.575	5.333 ± 0.565	6.000 ± 0.969	2.875 ± 1.076	8.500 ± 2.204

α	**Proxy exploitability**					
	$T = K$	$T = 10K$	$T = 50K$	$T = 100K$	$T = 500K$	$T = 1000K$
NT	1.494e-03	4.594e-04	3.592e-03	4.772e-03	9.903e-03	3.388e-03
0.1	7.727e-04	4.594e-04	3.592e-03	4.772e-03	9.903e-03	3.388e-03
0.3	5.838e-04	4.594e-04	3.592e-03	4.772e-03	9.903e-03	3.388e-03
0.5	**0.000e+00**	4.594e-04	3.592e-03	4.772e-03	9.903e-03	3.388e-03
0.7	9.391e-05	4.594e-04	3.592e-03	4.772e-03	9.903e-03	3.388e-03
0.9	9.758e-03	**0.000e+00**	2.860e-03	2.992e-03	**0.000e+00**	3.371e-03
0.99	3.236e-02	3.647e-03	**0.000e+00**	**0.000e+00**	1.211e-02	**0.000e+00**
Pure	3.848e-02	3.204e-02	2.559e-02	2.362e-02	1.463e-02	3.103e-02

α	**Robust score**					
	$T = K$	$T = 10K$	$T = 50K$	$T = 100K$	$T = 500K$	$T = 1000K$
NT	1.151e-01 ± 6.772e-08	1.151e-01 ± 2.175e-07	1.153e-01 ± 3.707e-07	1.154e-01 ± 5.797e-07	1.167e-01 ± 2.046e-06	1.152e-01 ± 1.297e-06
0.1	1.158e-01 ± 1.843e-05	1.151e-01 ± 2.175e-07	1.153e-01 ± 3.707e-07	1.154e-01 ± 6.019e-07	1.167e-01 ± 2.046e-06	1.152e-01 ± 1.297e-06
0.3	1.160e-01 ± 3.441e-05	1.151e-01 ± 2.175e-07	1.153e-01 ± 3.707e-07	1.154e-01 ± 6.019e-07	1.167e-01 ± 2.046e-06	1.152e-01 ± 1.297e-06
0.5	**1.166e-01 ± 9.751e-05**	1.151e-01 ± 2.175e-07	1.153e-01 ± 3.707e-07	1.154e-01 ± 5.797e-07	1.167e-01 ± 2.046e-06	1.152e-01 ± 1.297e-06
0.7	1.165e-01 ± 6.176e-04	1.151e-01 ± 2.175e-07	1.153e-01 ± 3.707e-07	1.154e-01 ± 5.797e-07	1.167e-01 ± 2.051e-06	1.152e-01 ± 1.297e-06
0.9	1.068e-01 ± 3.176e-03	**1.156e-01 ± 4.586e-05**	1.160e-01 ± 6.348e-05	1.172e-01 ± 9.722e-05	**1.266e-01 ± 3.829e-04**	1.152e-01 ± 4.288e-06
0.99	8.423e-02 ± 3.118e-03	1.119e-01 ± 2.316e-03	**1.189e-01 ± 1.888e-03**	**1.202e-01 ± 2.101e-03**	1.145e-01 ± 4.519e-03	**1.186e-01 ± 7.684e-04**
Pure	7.810e-02 ± 2.570e-03	8.354e-02 ± 2.710e-03	9.327e-02 ± 2.202e-03	9.658e-02 ± 2.097e-03	1.120e-01 ± 3.625e-03	8.755e-02 ± 6.497e-03

10.3 Conclusion: Nash-methods have computational and modeling advantages for decision making under uncertainties

Uncertainties are a big issue in long term energy planning.

Technically speaking, we tuned a parameter-free adversarial bandit algorithm $tExp3.P + t$, for large scale problems, efficient in terms of performance itself, and also in terms of sparsity. $tExp3.P+t$ performed better than $tExp3.P$ without truncation. Moreover, $tExp3.P + t$ with truncation parameter $\alpha = 0.7$, which is theoretically optimal [Teytaud and Flory, 2011], got stable performance in the experiments.

From a user point of view, we propose a tool with the following advantages:

- Natural extraction of interesting policies and critical scenarios.
- Faster computational cost than the Wald or Savage classical methodologies.

We methodology only requires a mapping $R : (k, s) \mapsto R(k, s)$, which computes the outcome if we use the policy k and the outcome is the scenario s. Multiple objective functions can be handled: if we have two objectives (e.g. economy and greenhouse gas pollution), we can just duplicate the scenarios, one for which the criterion is economy, and one for which the criterion is greenhouse gas.

Given a problem, the algorithm will display a matrix of rewards for different policies and for several scenarios (including, by the trick above, several criteria such as particular matter, greenhouse, and cost).

Algorithm 12 $tExp3.P$, combining $Exp3.P$ and the truncation method. α is the truncation parameter.

Input: $R_{m\times n}$, matrix defined by mapping $(i,j) \mapsto R_{i,j}$
Input: a time horizon (computational budget) $T \in \mathbb{N}^+$
Input: α, truncation parameter

Run $Exp3.P$ during T iterations; get an approximation (p,q) of the Nash equilibrium
$\zeta = \max_{i\in\{1,\ldots,m\}} \frac{(Tp_i)^\alpha}{T}$ ▷ compute the threshold for p
for $i \leftarrow 1$ to m **do** ▷ Truncation
    if $p_i \geq \zeta$ **then**
        $p'_i = p_i$
    else
        $p'_i = 0$
    end if
end for
for $i \leftarrow 1$ to m **do**
    $p''_i = \frac{p'_i}{\sum_{j=1}^{m} p'_j}$
end for
$\zeta' = \max_{i\in\{1,\ldots,n\}} \frac{(Tq_i)^\alpha}{T}$ ▷ compute the threshold for q
for $i \leftarrow 1$ to n **do** ▷ Truncation
    if $q_i \geq \zeta'$ **then**
        $q'_i = q_i$
    else
        $q'_i = 0$
    end if
end for
for $i \leftarrow 1$ to n **do**
    $q''_i = \frac{q'_i}{\sum_{j=1}^{n} q'_j}$
end for
return p'' and q'' as an approximate Nash equilibrium of the game

11 Optimizing random seeds

11.1 Introduction: portfolios of random seeds

Artificial intelligence (AI) has been invaded by ensemble methods [Breiman, 1996, Shapire et al., 1997]. In games, some recent chapters propose to do so, and in particular to combine variants of a single program, thanks to tricks on random seeds.

11.1.1 The impact of random seeds

Given a stochastic AI, we can check its performance against a baseline program (possibly itself) as we vary the random seed. I.e. we can generate K different random seeds, and for each of these seeds play K_t games against the baseline. We can then plot the winning rates, sort, and compare the variations to the standard deviations. Results are presented in Figure 11.1 and show for several games that the seed has a significant impact.

The methodologies presented in this chapter are based on this phenomenon.

11.1.2 Related work

Several works were dedicated to combining several AIs in the past. [Nagarajan et al., 2015] combines several different AIs. [Gaudel et al., 2010] uses Nash methods for combining several opening books.

[Saint-Pierre and Teytaud, 2014] proposed to construct several AIs from a single stochastic one and to combine them by the BestSeed and Nash methods detailed below.

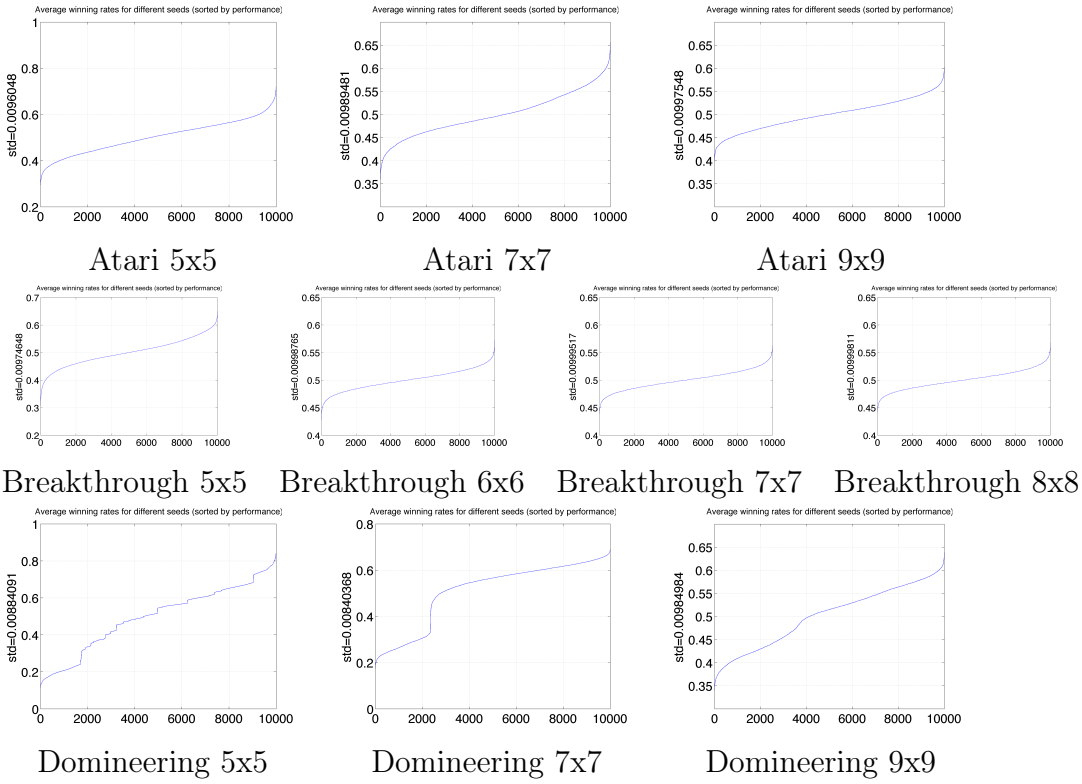


Figure 11.1: Impact of the seed on the success rate. For the n^{th} value, we consider the n^{th} worst seed for Black and the n^{th} seed for White, and display their average scores against all opponent seeds. The label on the y-axis shows the standard deviation of these averages; we see that there are good seeds (far above the 50 % success rate, by much more than the standard deviation).

11.1.3 Outline of the present chapter

The present introduction has shown the impact of random seeds. Section 11.2 presents some classical algorithms. Section 11.3.1 proposes a modified parametrization of these algorithms. Section 11.3.2 presents the obtained algorithms, after this modification. Section 11.4 presents the games used in our experiments. Section 11.5 shows our experimental results.

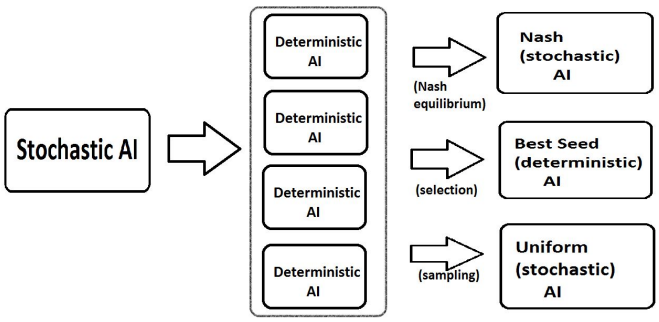


Figure 11.2: Summary of the approach. We suppose that a single stochastic AI is available (Section 11.2.1). We create many deterministic AIs from a single stochastic one (Section 11.2.2). Using it through a matrix construction (Section 11.2.3), we create boosted AIs (Section 11.2.4), which outperform the original one for various criteria.

11.2 Algorithms for boosting an AI using random seeds

This section presents an overview of two methods proposed in [Saint-Pierre and Teytaud, 2014] for building a boosted algorithm, from a set of seeds:

- the Nash-approach;
- the BestSeed-approach.

We also define, for comparison, the uniform portfolio, which is just a uniform sampling of the considered random seeds, as detailed later; it is not stronger than the original AI.

11.2.1 Context

We assume that an AI is given. This AI is supposed to be stochastic; even with the same flow of information, it will not always play the same sequence of moves. This is for example the case for Monte Carlo Tree Search [Coulom, 2006, Kocsis and Szepesvari, 2006a]. We propose an extension of the method used in [Saint-Pierre and Teytaud, 2014] and apply it to three new games, namely Atari-Go, Domineering and Breakthrough.

11.2.2 Creating a probability distribution on random seeds

Typically, a stochastic computer program uses a random seed. The random seed w is randomly drawn (using the clock, usually) and then a pseudo-random sequence is generated. Therefore, a stochastic program is in fact a random variable, distributed over deterministic program. Let us define:

- AI is our game playing artificial intelligence; it is stochastic.
- $AI(\omega)$ is a deterministic version; ω is a seed, which is randomly drawn in the original AI.

We can easily generate plenty of ω and therefore one stochastic AI becomes several deterministic AIs, termed $AI_1, \ldots, AI_K$.

11.2.3 Matrix construction

Let us assume then one of the players plays as Black and the other plays as White. We can do the same construction as above for the AI playing as Black and for the AI playing as White. We get $AI_1,\ldots,AI_K$ for Black, and $AI'_1, \ldots, AI'_{K_t}$ for White. From now on, we present the algorithm for Black - still, for this, we need the AI' as well. The algorithm for enhancing the AI as White is similar. Let us define $M_{i,j} = 1$ when AI_i (playing as Black) wins against AI'_j (playing as White). Otherwise, $M_{i,j} = 0$. Also, let us define $M'_{i,j} = 1$ when AI'_i (playing as White) wins against AI_j (playing as Black) - we have $M'_{i,j} = 1 - M_{j,i}$.

11.2.4 Boosted AIs

In [Saint-Pierre and Teytaud, 2014], they use $K = K_t$, hence they use the same matrix for Black and for White - up to a transformation $M \mapsto 1 -^t M$. The point in the present chapter is to show that we can save up time by using $K \neq K_t$. This means that we need two matrices:

- M (used for the learning for Black) is the matrix of $M_{i,j}$ for $1 \leq i \leq K$ and $1 \leq j \leq K_t$.
- M' (used for the learning for White) is the matrix of $M'_{i,j}$ for $1 \leq i \leq K$ and $1 \leq j \leq K_t$.

If $K_t \leq K$, M and M' have $K_t \times K_t$ entries in common (up to transformation $M'_{i,j} = 1 - M_{j,i}$); therefore building M and M' needs the result of $2K \times K_t - K_t^2$ game results.

11.2.4.1 The BestSeed approach

Given M, the BestSeed approach consists in selecting one seed. We just pick up i^* such that $\sum_{j=1}^{K_t} M_{i^*,j}$ is maximal. Our BestSeed approach for Black will use random seed i^*; it is a deterministic program.

11.2.4.2 The Nash approach

For the Nash approach, we select a probability distribution on seeds. We compute (p, q), a Nash equilibrium of M. The Nash approach for Black will use seed i with probability p_i. The Nash approach provides a stochastic policy, usually stronger than the original policy [Saint-Pierre and Teytaud, 2014].

11.3 Rectangular algorithms

11.3.1 Hoeffding's bound: why we use $K \neq K_t$

At first view, the approach in [Saint-Pierre and Teytaud, 2014] is simple and sound: they need one matrix for both Black and White. However, their approach needs the result of K^2 games. With our rectangular approach, if we use K different seeds and K_t opponent seeds, we need $2K \times K_t - K_t^2$ games.

Let us now check the precision of our approach. Our algorithms use averages of rows and averages of columns. Let us define μ_i the average value of the i^{th} row of M, if K_t was infinite - this is the average winning rate of AI_i playing as Black against AI playing as White. And let us define $\hat{\mu}_i$ the average value that we get, with our finite value K_t. Hoeffding's bound [Hoeffding, 1963] tells us that with probability $1 - \delta$, $|\mu_i - \hat{\mu}_i| \leq \sqrt{-\log(\delta/2)/(2K_t)}$. By Bonferroni correction (i.e. union bound), with probability $1 - \delta$, for all $i \leq K$, $|\mu_i - \hat{\mu}_i| \leq \sqrt{-\log(\delta/(2K))/(2K_t)}$. For a requested precision ϵ, we can do as follows:

- Choose a value of K large enough, so that at least one seed i is optimal within precision $\epsilon/2$.
- Choose K_t such that $\sqrt{-\log(\delta/(2K))/(2K_t)} \leq \epsilon/2$.

This means K_t logarithmic as a function of K, multiplied by $O(\epsilon^{-2})$. As a conclusion, for a fixed precision ϵ and a fixed confidence δ, finding

the best seed among K seeds can be done with K_t logarithmic as a function K.

11.3.2 Rectangular algorithms: $K \neq K_t$

We now summarize the two approaches, in Algorithm 13.

Algorithm 13 Approach for boosting a game stochastic game AI.

Input: a stochastic AI playing as Black, a stochastic AI' playing as White.

Output: a boosted AI termed BAI playing as Black, a boosted AI BAI' playing as White.

1: Build $M_{i,j} = 1$ if AI_i (Black) wins against AI'_j (White) for $i \in \{1, \ldots, K\}$ and $j \in \{1, \ldots, K_t\}$.
2: Build $M'_{i,j} = 1$ if AI'_i (White) wins against AI_j (Black) for $i \in \{1, \ldots, K\}$ and $j \in \{1, \ldots, K_t\}$.
3: **if** BestSeed // deterministic boosted AI **then**
4: BAI is AI_i where i maximises $\sum_{j=1}^{K_t} M_{i,j}$.
5: BAI' is AI'_i where i maximises $\sum_{j=1}^{K_t} M'_{i,j}$.
6: **end if**
7: **if** Nash // stochastic boosted AI **then**
8: Compute (p, q) a Nash equilibrium of M.
9: BAI is AI_i with probability p_i
10: Compute (p', q') a Nash equilibrium of M'.
11: BAI' is AI'_j with probability p'_i
12: **end if**
13: **if** Uniform // stochastic AI **then**
14: BAI is AI_i with probability $1/K$.
15: BAI' is AI'_j with probability $1/K$.
16: **end if**

11.4 Testbeds

We provide experiments on three board games, namely Domineering, Atari-Go and Breakthrough. All these games can be played on arbitrary squared board sizes. In all our experiments, we use a MCTS implementation.

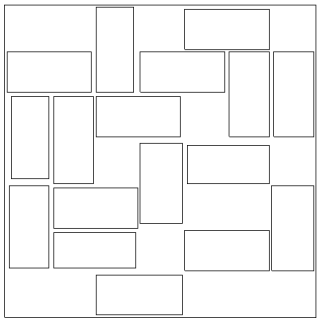


Figure 11.3: Example of Domineering game. The vertical player can not play anymore and loses the game.

11.4.1 The Domineering Game

Domineering is a two-player game with very simple rules: each player in turn puts a tile on empty locations in the board. The game starts with an empty board. The first player who can not play loses the game (Figure 11.3). Usually, one of the player has vertical 2x1 tiles, and the other has horizontal 1x2 tiles. Domineering can be played on boards of various shapes, most classical cases are rectangles or squares. For squared boards, Domineering is solved until board size 10x10 [Breuker et al., 2000, Bullock, 2002]. Domineering was invented by Göran Andersson [Gardner and Ball, 1974]. Jos Uiterwijk recently proposed a knowledge based method that can solve large rectangular boards without any search [Uiterwijk, 2013].

11.4.2 Breakthrough

The breakthrough game, invented by Dan Troyka in 2000, has very simple rules: all pieces can move straight ahead or in diagonal (i.e. three possible target locations). Captures are possible in diagonal only. Players play in turn, and the first player who reaches the opposite first row or captures all opponents pieces has won. There is no draw in Breakthrough - there is always at least one legal move, and pieces can only go forward (straight or diagonal) so that loops can not occur. The original position is as in Figure 11.4. This game won the 2001 8x8 Game Design Competition.

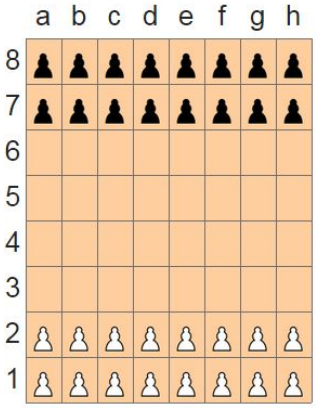


Figure 11.4: The initial position at breakthrough. The first player who reaches the opposite side has won. Source: Wikipedia.

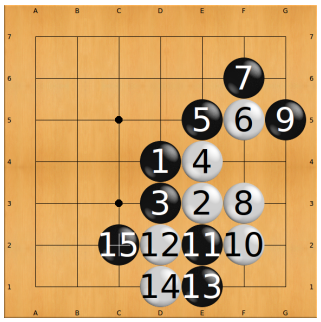


Figure 11.5: Example of Atari-Go game. White plays F1 and captures E1 and E2 - White wins the game.

11.4.3 Atari-Go

Yasuda Yasutoshi popularized this variant of the game of Go; the key difference is that the first player who makes a capture wins the game. Atari-Go is also known as Ponnuki-Go, One-capture-Go, or Capture-Go.

11.5 Experiments

11.5.1 Criteria

We consider the following criteria for our boosted AI playing both as Black and White:

- Generate K' seeds, randomly, for Black and K' seeds, randomly, for White.
- Consider the worst success rate SR of our boosted AI playing as White against these K' strategies for Black. Consider the worst success rate SR of our boosted AI playing as Black against these K' strategies for White.
- Our success rate is the average of these two success rates.

This is a strong challenge for K' large; since we consider separately White and Black, we have indeed K'^2 opponent strategies (each of the K' seeds for Black and each of the K' seeds for White) and consider the worst success rate. We will define this opponent as a K'-exploiter: it is an approximator of the exploitability property of Nash equilibria. It represents what can be done if our opponent could play the game K' times and select the best outcome.

For $K' = 1$, this opponent is playing exactly as the original AI: this is the success rate against a randomly drawn seed. A score $\geq 50\%$ against $K' = 1$ means that we have outperformed the original AI, i.e. boosting has succeeded; but it is satisfactory to have also a better success rate, against $K' > 1$, than the original AI.

11.5.2 Experimental setup

In order to validate the method, we take care that our algorithm is tested with a proper cross-validation: the opponent uses seeds which have never been used during the learning of the portfolio. This is done for all our experiments, BestSeed, Uniform, or Nash. For this reason, there is no bias in our results.

11.5.3 Results

All results are averaged over 100 runs. Results for Domineering are

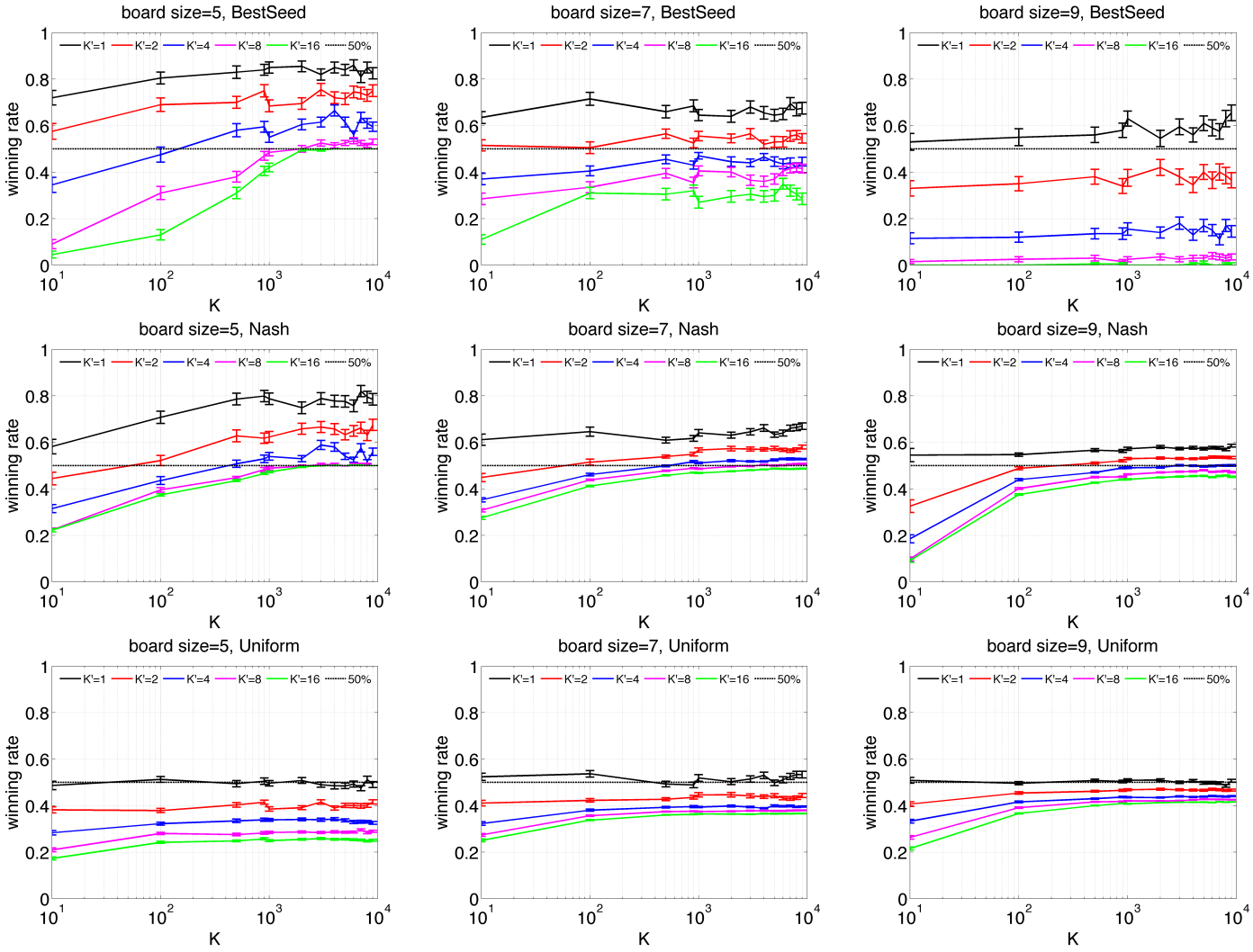


Figure 11.6: Results for domineering, with the BestSeed and the Nash approach, against the baseline ($K' = 1$) and the exploiter ($K' > 1$). $K_t = 900$ in all experiments. The performance of the uniform version (original algorithm) is also presented for comparison.

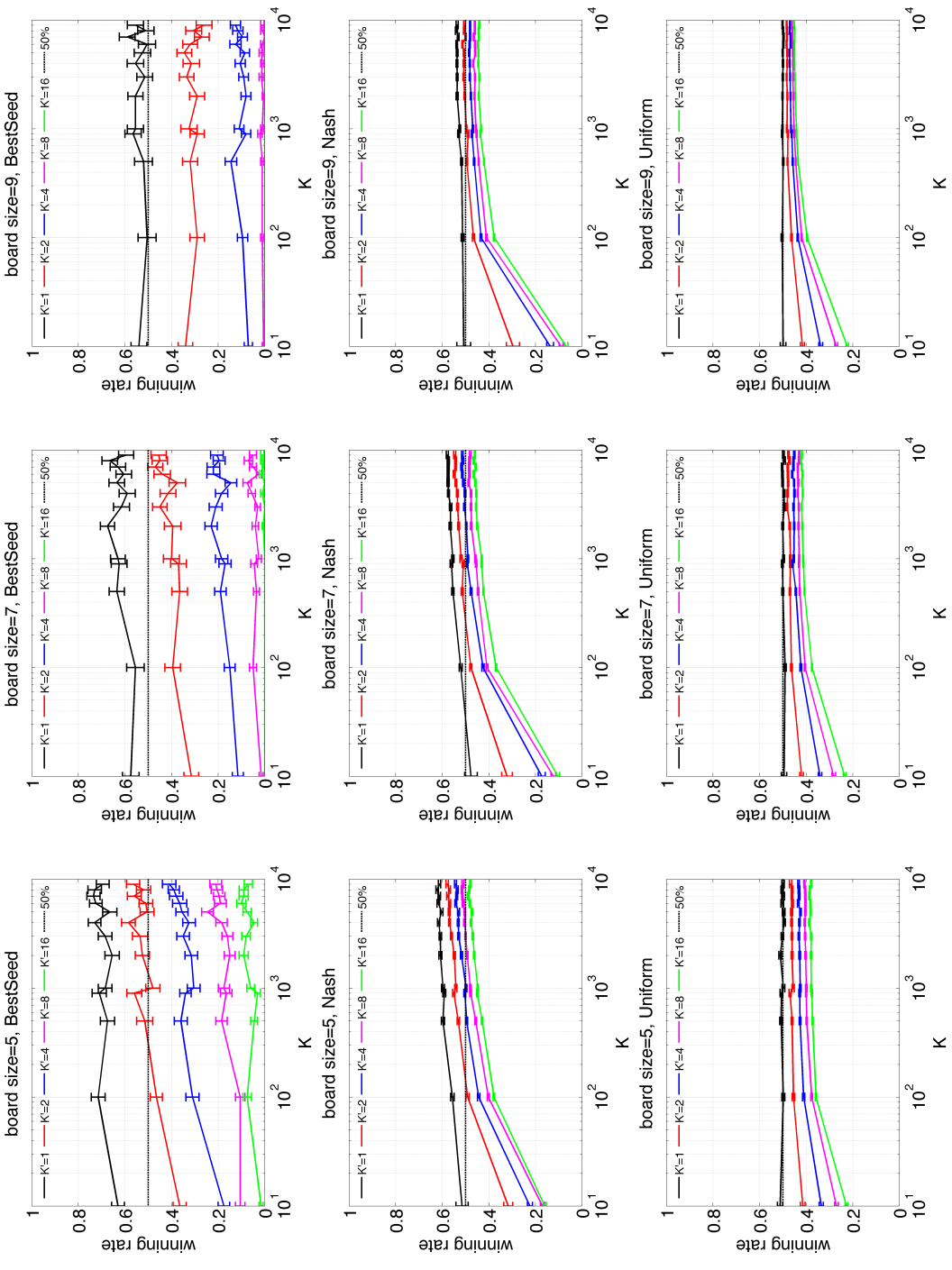


Figure 11.7: Results for Atari-Go, with the BestSeed and the Nash approach, against the baseline ($K' = 1$) and the exploiter ($K' > 1$). $K_t = 900$ in all experiments. The performance of the uniform version (original algorithm) is also presented for comparison.

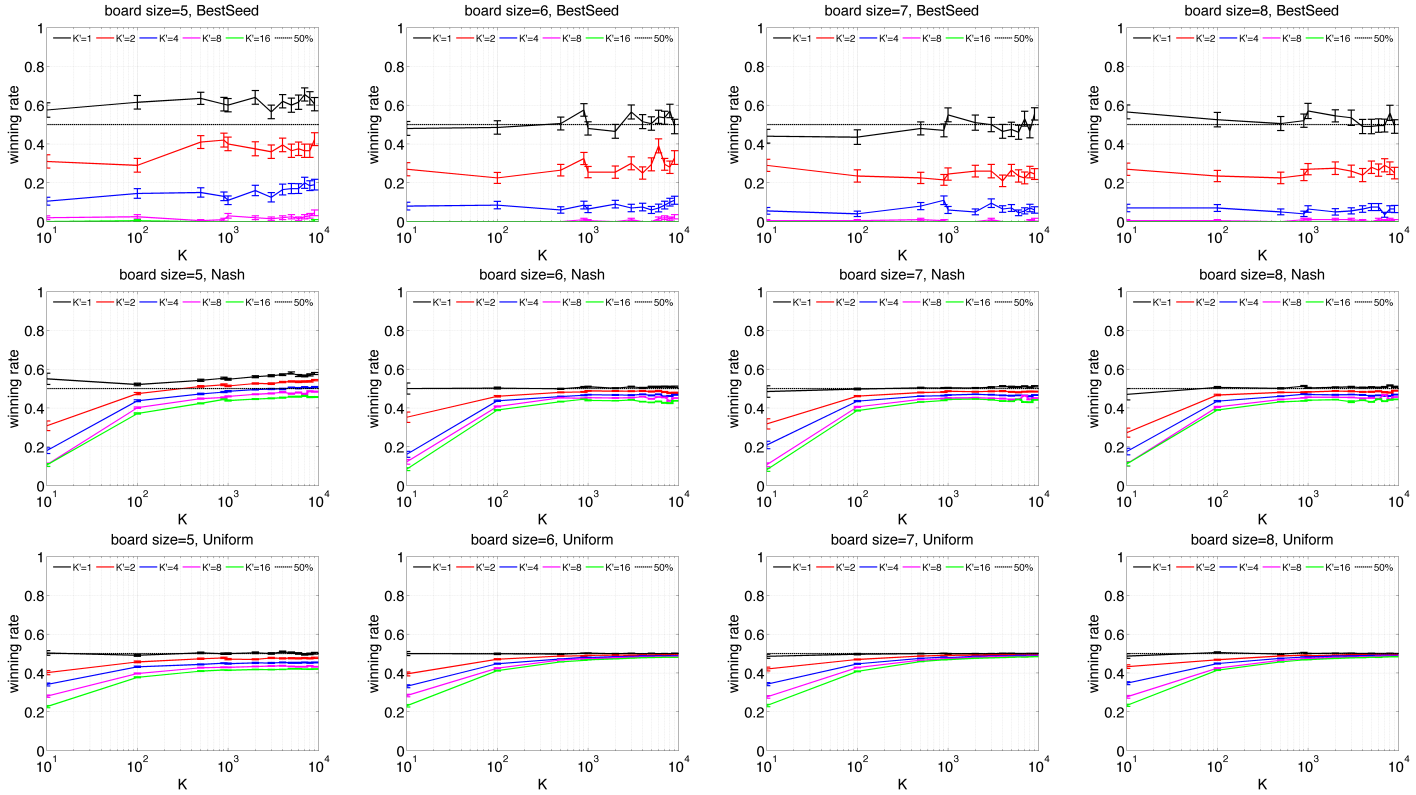


Figure 11.8: Results for Breakthrough, with the BestSeed and the Nash approach, against the baseline ($K' = 1$) and the exploiter ($K' > 1$). $K_t = 900$ in all experiments. The performance of the uniform version (original algorithm) is also presented for comparison.

presented in Figure 11.6. Results for Atari are presented in Figure 11.7. Results for Breakthrough are presented in Figure 11.8. In short, BestSeed performs well against the original algorithm (corresponding to $K' = 1$), but its performance against the exploiter ($K' > 1$) is very weak. On the other hand, the Nash version outperforms the original algorithm both in terms of success rate against $K' = 1$ (in all cases) and against $K' > 1$ in most cases (i.e. curves on the middle column in Figs. 11.6, 11.7, 11.8 are better than those on the right column) - however, for breakthrough in large size the results were (very) slightly detrimental for $K' > 1$, i.e. the "exploiter" could learn strategies against it.

11.6 Experiments with transfer

Results above were performed in a classical machine learning setting, i.e. with cross-validation; we now check the transfer, i.e. the fact that we improve an AI, we get a better performance when we test its performance against another AI.

This means, that whereas previous sections have obtained results such as

"When our algorithm takes A as baseline AI, the boosted counterpart A' outperforms A by XXX % winning rate."

we get results such as:

"When our algorithm takes A as baseline AI, the boosted counterpart A' outperforms A in the sense that the winning rate of A' against B is greater than the winning rate of A against B, for each B in a family { B1, B2, ..., Bk } of programs different from A."

11.6.1 Transfer to GnuGo

We applied BestSeed to GnuGo, a well known AI for the game of Go, with Monte Carlo tree search and a budget of 400 simulations. The BestSeed approach was applied with a 100x100 learning matrix, corresponding to seeds $\{1, \ldots, 100\}$ for Black and seeds $\{1, \ldots, 100\}$ for White.

Then, we tested the performance against GnuGo "classical", i.e. the non-MCTS version of GnuGo; this is a really different AI with different playing style. We got positive results as shown in Table 11.1. Results are presented for Black; for White the BestSeed had a negligible impact.

Table 11.1: Performance of BestSeed-Gnugo-MCTS against various GnuGo-default programs, compared to the performance of the default Gnugo-MCTS. The results are for GnuGoMCTS playing as Black vs GnuGo-classical playing as White, and the games are completely independent of the learning games (which use only Gnugo-MCTS). Results are averaged over 1000 games. All results in 5x5, komi 6.5, with a learning over 100x100 random seeds.

Opponent	Performance of BestSeed	Performance of the original algorithm with randomized random seed
GnuGo-classical level 1	1. (± 0)	.995 (± 0)
GnuGo-classical level 2	1. (± 0)	.995 (± 0)
GnuGo-classical level 3	1. (± 0)	.99 (± 0)
GnuGo-classical level 4	1. (± 0)	1. (± 0)
GnuGo-classical level 5	1. (± 0)	1. (± 0)
GnuGo-classical level 6	1. (± 0)	1. (± 0)
GnuGo-classical level 7	.73 (± .013)	.061 (± .004)
GnuGo-classical level 8	.73 (± .013)	.106 (± .006)
GnuGo-classical level 9	.73 (± .013)	.095 (± .006)
GnuGo-classical level 10	.73 (± .013)	.07 (± .004)

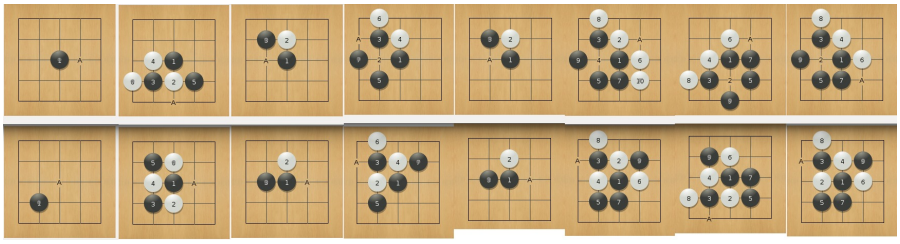


Figure 11.9: Comparison between moves played by BestSeed-MCTS (top) and the original MCTS algorithm (bottom) in the same situations.

11.6.2 Transfer: validation by a MCTS with long thinking time

Figure 11.9 provides a summary of differences between moves chosen (at least with some probability) by the original algorithm, and the ones chosen in the same situation by the algorithm with optimized seed. These situations are the 8 first differences between games played by the original GnuGo and by the GnuGo with our best seed.

We use GnugoStrong, i.e. Gnugo with a larger number of simulations, for checking if Seed 59 leads to better moves.

GnugoStrong is precisely defined as « gnugo –monte-carlo –mc-

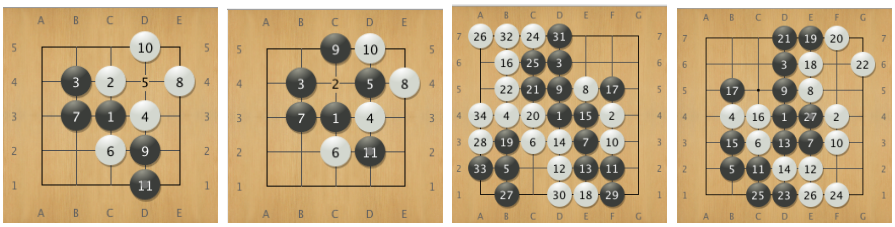
games-per-level 100000 –level 1». We provide below some situations in which Seed 59 (top) proposes a move different from the original Gnugo with the same number of simulations. Gnugo is not deterministic; therefore this is simple the 8 first differences found in our sample of games (we played games until we find 8 differences).

The conclusions from this GnugoStrong experiment are as follows:

- GnugoStrong prefers Top; Bottom is considered as a loss (i.e. playing the entire game leads to a loss (experiment reproduced 5 times)).
- Here black moves are the same up to symmetries, and seed 59 is the case in which the white opponent lost the most frequently. Both are considered as wins for Black.
- Both are considered as wins for Black.
- Both are considered as wins for Black.
- Both are considered as wins for Black.
- GnugoStrong prefers Top; Bottom is considered as a loss (i.e. playing the entire game leads to a loss (experiment reproduced 5 times)).
- GnugoStrong prefers Top; Bottom is considered as a loss (i.e. playing the entire game leads to a loss (experiment reproduced 5 times)).
- GnugoStrong prefers Top; Bottom is considered as a loss (i.e. playing the entire game leads to a loss (experiment reproduced 5 times)).

11.6.3 Transfer: human analysis

Figure 11.10 provides some AI v.s. human games on 5x5 and 7x7 board. Each human player played with AIs using BestSeed-MCTS and the original MCTS algorithm in the same situations, without knowing which their identity, then judged which AI is stronger. BestSeed-MCTS has been judged to be more powerful opponent by both human players.



(a) Original MCTS. (b) BestSeed-MCTS. (c) Original MCTS. (d) BestSeed-MCTS.

Figure 11.10: Comparison between moves played by BestSeed-MCTS (black) and the original MCTS algorithm (black) in the same situations. The white player of 5x5 games is a Taiwanese amateur 6 dan and the player of 7x7 games is a Taiwanese amateur 1 dan. Both players played against the AI without knowing opponents' algorithms. BestSeed-MCTS has been judged to be more powerful opponent by both human players.

11.7 Conclusions: optimizing the seed works well for many games on small boards

Our results (winning rate of the boosted algorithm against the non-boosted baseline) are roughly for BestSeed:

- 73.5%, 67.5%, 59% for Atari-Go in 5x5, 7x7 and 9x9 respectively.
- 65.5%, 57.5%, 55.5%, 57% for Breakthrough in 5x5, 6x6, 7x7 and 8x8 respectively.
- 86%, 71.5%, 65.5% for Domineering in 5x5, 7x7 and 9x9 respectively.

From Figure 11.1, we can guess that larger values of K would provide better results. We might see these results as a very simple and effective tool for building an opening book with no development effort, no human expertise, no storage of database.

We clearly see in Section 11.5.3 that $K >> K_t$ provides significant improvement; K is much more the key point than K_t.

The online computational overhead of the methods used in this chapter is negligible, as both for BestSeed and Nash it is just determining the random seed at the beginning of the algorithm. The boosted

AIs significantly outperform the baselines. This does not require any source code development. We will see in Section 12 a different use of seeds - whereas here we learn seeds for the initialization of the AI, there we learn seeds specifically for positions, leading to an improvement in the scalability of MCTS (Figure 12.8).

Novelty

The BestSeed and the Nash algorithms (Section 11.2) are not new. The analysis of $K_t < K$ (Section 11.3.1) is new. The algorithms with $K_t < K$ (Section 11.3.2) are new. The analysis of the impact of seeds is new (Figure 11.1). The application to three games (Domineering, Atari-Go, Breakthrough) is new.

12 Optimizing position-specific random seeds: Tsumegos

12.1 Problem Statement

In this section we formalize the problem at hand. Section 12.1.1 describes the application under study. Section 12.1.2 explains the computation of a game value and introduces the concept of matrix games.

12.1.1 Tsumego Problems

Tsumego problems are played on a Go board, with size usually 9, 13 or 19. In most cases, the difficulty lies in the capture of other stones or creating eyes for your own group of stones. Stones are the pieces used in Go. A capture is done by surrounding an opposing stone or group of stones by occupying all orthogonally-adjacent points. A liberty is an open intersection on the board next to a stone. An enclosed liberty is called an "eye". A group of stones with at least two separate eyes is said to be unconditionally "alive". The lynchpin of the game is to capture enemy stones and/or maintain your own groups of stones alive by creating eyes.

Yoji Ojima, the author of Zen, the current best Computer Go program, proposed in the computer-Go mailing list a set of Go problems (Tsumego) [1]. These problems are the current standard for MCTS research and the testbed at the origin of Zen's domination on Computer Go. This set of problems can be separated into 5 different families. The rest of this section presents these families.

Oiotoshi. An oiotoshi is a situation in which a player threatens to capture a group of stones. The other player can connect his group

[1] `https://github.com/ujh/HouseBot/tree/master/test/mctest`

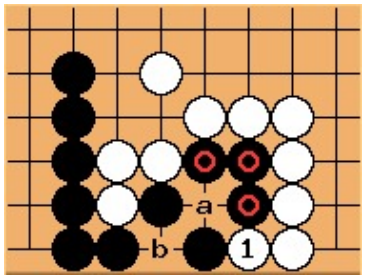


Figure 12.1: Oiotoshi. After White-(1), Black-a leads to an Oiotoshi killing 6 black stones; whereas Black-b leads to only 3 stones captured.

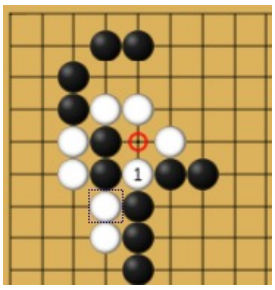


Figure 12.2: Snapback. If Black captures by the red circle, White captures back with (1).

to another one, but the resulting (extended) group is still under threat and ultimately is killed. Basically, it is a situation where a player must sacrifice some stones to let others survive. Figure 12.1 presents an example of Oiotoshi from Senseis.org.

Oiotoshi with ko. In Go, it is forbidden for a player to execute an action that results in the exact same state before its last action. Such a situation is called a ko. In simpler terms, given a state s_t the state s_{t+2} cannot be the same as s_t. The notion of ko is relatively easy to grasp, yet the main complexity of Go stems from this rule: [Robson, 1983] has shown the Exptime complexity of a family of Go positions.

Snapback. A snapback is a situation in which capturing a stone leads to a bigger capture by the opponent. For instance, Figure 12.2 presents a classical example [2].

[2] `http://en.wikipedia.org/wiki/Snapback\_(Go)\#Capturing_tactics`

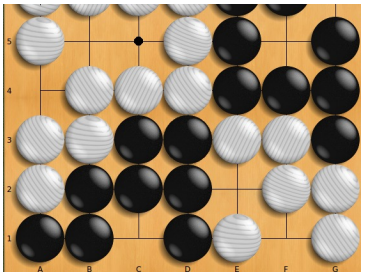


Figure 12.3: Seki. If Black E2, White captures all D2. If White E2, Black captures all F2. Thus the situation should stay at "equilibrium".

Seki. There is seki when a group of Black stones and a group of White stones are such that if Black attacks, it is killed by White. The opposite is also true, if White attacks, it is killed by Black. It is a standstill where the first to take an action loses. The two groups are alive but not stable enough for attacking. A situation in which a seki is involved is difficult to evaluate, because it is very unstable and many simple Monte Carlo will conclude that there is 50% probability that Black kills and 50% probability that White kills. Figure 12.3 presents such a situation.

Life & Death. In Life problems, the player must ensure that a group (one of his own groups) is alive. In Kill problems, the player must ensure that a given opponent group of stones die. Life & Death problems are critical in Go. Moreover, one should never play more actions than necessary for making a group alive, otherwise stones are wasted. Life & Death problems are difficult to correctly handle in Monte Carlo simulations compared to humans because the number of correct actions is usually very small and involves rules that humans learn by abstract representation. Since most Go problems are related to Life & Death, this category is for problems which do not fall into other categories. Figure 12.4 displays a simple Life & Death problem.

Semeai (liberty race). In a semeai, a Black and a White group are embroiled in a fight to death. Each group can only survive by killing one another. In the simplest case (Semeais can have complex liberties), White plays in Black liberties until Black is dead, Black plays in White liberties until White is dead; the first who fills all the opponent's liberties has won the semeai. As for the other families, semeais in real games have complex effects. It is known [Cazenave and Saffidine, 2011] that

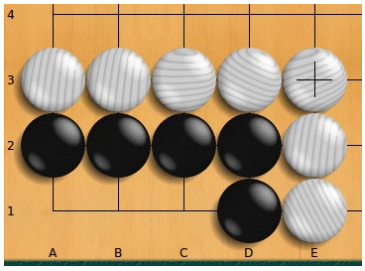


Figure 12.4: Life & Death. B1 is the only position where Black can make two eyes. Thus, White-B1 kills B2, whereas Black-B1 makes B2 alive.

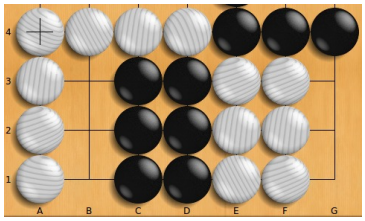


Figure 12.5: Semeai where both player have 3 liberties.

semeais are difficult for Monte Carlo because, unless there are domain specific knowledge introduced in the simulation policy to bias the simulation, even a simple semeai (i.e. a semeai in which one of the players wins easily) leads to 50% win for Black and 50% win for White. Figure 12.5 displays a simple semeai where both player have 3 liberties.

12.1.2 Game Value Evaluation

We use the following definitions. The **horizon** of a game is the maximum number of moves (i.e. actions) in this game. A **strategy** is a (possibly stochastic) mapping from states to legal moves. Let S be the state space of Go. Given the current state $s \in S$, A_s is the set of legal moves in s. S_f denotes the set of final states. A **perfect policy** $\pi^*(\cdot)$ is a policy such that for all s, $\pi^*(s)$ is an optimal action.

Values are either 1 for a win, or 0 for a loss. In a two-player finite sum deterministic game such as the game of Go, a win for a player means a defeat for the other. We here assume a game with no draw. We use the value function as in Definition 3.

Definition 3 (Value function)**.** *Considering a two-player 1-sum deterministic game with no draw, the game value function is defined by:*

$$v(s) = \begin{cases} 1 & \textit{a win for player 1, in case of optimal play starting in } s \\ 0 & \textit{a win for player 2, at least in case of perfect play} \end{cases}$$

for any state $s \in S$, where S is the state space.

If a perfect policy π^* is available, evaluating a state s is easy. One needs only to execute one simulation with π^*. When it reaches a final state $s_f \in S_f$, the value $v(s)$ of the state s is defined by $v(s) = v(s_f)$.

As mentioned previously, the ability to play optimally at any given state is seldom possible. Rather, we try to evaluate the value $v(s)$ by doing imperfect simulations from s. Each simulation i returns a value $\tilde{v}_i(s)$ that represents its evaluation of the current state s. Traditionally the value $v(s)$ of a game is evaluated through the average outcome $\hat{v}(s)$ of simulations $i \in \{1, \ldots, n\}$, where n is the total number of simulations. Equation (12.1) formalizes this approach.

$$\hat{v}(s) = \frac{1}{n} \sum_{i=1}^{n} \tilde{v}_i(s) \tag{12.1}$$

This is the classical Monte Carlo (MC) estimate, with n simulations, of the value $v(s)$ of a state s. It converges, for $n \to \infty$, to $\mathbb{E}[\tilde{v}_1(s)]$ if simulations are independent.

12.1.3 Weighted Monte Carlo

Usually $\mathbb{E}[\tilde{v}_1(s)] \neq v(s)$. Therefore, there are many cases in which $\hat{v}(s) \neq v(s)$, even when $n \to \infty$. This can lead to very poor approximations, e.g. for Tsumego problems such as seki and semeai [Cazenave and Saffidine, 2011]. This is the motivation for our approach which consists in weighting the average outcomes. We formalize the idea in Definition 4.

Definition 4 (Weighted Monte Carlo Estimate)**.** *Let $\tilde{v}_i(\cdot)$ denote the evaluation of function $v : S \mapsto \mathbb{R}$ at the i^{th} independent simulation. For any $s \in S$, $n \in \mathbb{N}^+$,*

$$\hat{v}(s) = \sum_{i=1}^{n} w_i \tilde{v}_i(s) \tag{12.2}$$

with $\sum_{i=1}^{n} w_i = 1$ and $\forall i \in \{1, \ldots, n\}$, $1 \geq w_i \geq 0$.

The question that remains is to find a good vector of weight w described in Definition 4.

12.2 Proposed algorithm and mathematical proof

Section 12.2.1 describes an auxiliary matrix game used in the rest of the chapter. Section 12.2.2 describes the notion of Nash Equilibria (NE), states our hypothesis and describes the algorithmic computation of a NE. Section 12.2.3 provides a mathematical analysis.

As this chapter focuses on the game of Go, we use B and W to refer to the Black and White player, respectively. However, our approach is consistent on other two-player finite sum deterministic games.

12.2.1 Construction of an associated matrix game

In a two-player deterministic game, the simulation policy π combines a randomized policy π_B for situations with Black to play and a randomized policy π_W for situations with White to play. π_B and π_W depend on random seeds. Therefore, a Monte Carlo simulation consists in choosing a random seed for each player, i.e. $b(i)$ for the Black player and $w(i)$ for the White player at the i^{th} simulation, where b and w are 2 sequences of random seeds, and following the simulation policies, executing an action until it reaches a final state $s_f \in S_f$ and return $\tilde{v}_i(s)$. The choice of the random seeds is therefore equivalent to a matrix game, where each player chooses a random seed.

Given $K_1, K_2 \in \mathbb{N}^+$, let us build a matrix $M_{K_1 \times K_2}$ by: $\forall i \in \{1, \ldots, K_1\}$, $\forall j \in \{1, \ldots, K_2\}$, $M_{i,j}$ is the result of the game (0 or 1) when using random seed $b(i)$ for Black (row player) and $w(j)$ for White (column player), see also Figure 12.6(a).

For consistency with Equation (12.2), let us renumber simulations with

$$\tilde{v}_{n(i,j)}(s) = M_{i,j} \quad \text{with} \quad n(i,j) = (i-1)K_2 + j. \tag{12.3}$$

We then define the associated matrix game as follows:

- $\forall i \in \{1, \ldots, K_1\}$, $\forall j \in \{1, \ldots, K_2\}$, simultaneously:
  - the row player chooses $b(i)$ as random seed;
  - the column player chooses $w(j)$ as random seed.

- The row player receives reward $M_{i,j}$ and the column player receives reward $1 - M_{i,j}$.

12.2.2 Nash Equilibria for matrix games

Nash Equilibria (NE) for two-player finite sum matrix games are recalled in Definition 1. Our hypothesis is that Equation (12.2), i.e. $\hat{v}(s) = \sum_{i=1}^{n} w_i \tilde{v}_i(s)$, applied with

- Either
$$w_{n(i,j)} = x_i y_j \tag{12.4}$$
where (x, y) is a Nash equilibrium of the matrix game M defined in Section 12.2.1. This case is termed "Nash reweighting".

- Or $w_{n(i,j)} = 1$ for i maximizing $\sum_{j=1}^{K_2} M_{i,j}$ and j minimizing $\sum_{i=1}^{K_1} M_{i,j}$, and $w_{n(i,j)} = 0$ for other (i, j). This case is termed "Best Seed (BS) reweighting".

is a better estimate of $v(s)$ than the simple Unpaired Monte Carlo estimate from equation (12.1) with the same number of simulations, or the Paired Monte Carlo estimate given by

$$\frac{1}{K_1 \times K_2} \sum_{i=1}^{K_1} \sum_{j=1}^{K_2} M_{i,j} \tag{12.5}$$

with $K_1 \times K_2 = n$.

Using Equations (12.4) and (12.3), Equation (12.2) becomes

$$\hat{v}(s) = \sum_{i=1}^{K_1 K_2} w_i \tilde{v}_i(s) = \sum_{i=1}^{K_1} \sum_{j=1}^{K_2} x_i y_j \tilde{v}_{n(i,j)}(s) \tag{12.6}$$

in the Nash case, and $\hat{v}_0(s) = \tilde{v}_{n(i,j)}$ in the BS case, with some i (randomly break tie) maximizing $\sum_{j=1}^{K_2} M_{i,j}$ and j minimizing $\sum_{i=1}^{K_1} M_{i,j}$.

This is summarized in Algorithm 14.

We will now (Section 12.2.3) show that this approach is asymptotically consistent in a general setting, which is not the case for the classical MC estimate as explained in Section 12.1.2.

Algorithm 14 The construction of matrix game and its solving.

Input: current state s
Input: a policy π_B for Black, depending on a seed in $\mathbb{N}^+$
Input: a policy π_W for White, depending on a seed in $\mathbb{N}^+$
for $i \in \{1, \dots, K_1\}$ **do**
    for $j \in \{1, \dots, K_2\}$ **do**
        $M_{i,j} \leftarrow$ outcome of the game starting in s with π_B playing as Black with seed $b(i)$ and π_W playing as White with seed $w(j)$ ▷ *construction of the matrix M*
    end for
end for
Compute strategies x for Black player and y for White player ▷ for matrix M, with either BS or NE
Output: xMy ▷ *approximate value of the game M*

12.2.3 Mathematical analysis

We use the definitions introduced in Section 12.1.2.

Property 5. *If the value of a Tsumego s is $v(s) = 1$ (resp. $v(s) = 0$), then there is a strategy π_B^* for Black (resp. π_W^* for White) which always wins. Moreover, if the game has a finite horizon, this strategy can be chosen deterministically.*

Proof. Proof immediate by definition for the existence of a strategy; it can be chosen deterministically, by induction. □

We will now consider artificial players (BAI, Black Artificial Intelligence, and WAI, White Artificial intelligence), with pseudo-randomization.

Definition 5 (Nash solving and BS solving)**.** *Consider a game position s (e.g. a Go problem, termed Tsumego). Consider BAI_i and WAI_j, strategies for Black and White respectively, depending on a random seed. BAI_i uses the random seed $b(i)$. WAI_j uses the random seed $w(j)$.*

Let us define $M_{i,j}$:

- *$M_{i,j} = 1$ if BAI_i wins against WAI_j when starting in s.*
- *$M_{i,j} = 0$ otherwise.*

Let us define $M_{K \times K}$ the matrix of the $M_{i,j}$, $\forall (i,j) \in \{1, \dots, K\}^2$. Let us define

- *the Nash estimate $v^{(K)}$, which is the Nash value of $M_{K\times K}$,*
- *and the BS-estimate $v_0^{(K)} = M_{i_0,j_0}$ with i_0 such that $\sum_j M_{i_0,j}$ is maximum and j_0 such that $\sum_i M_{i,j_0}$ is minimum (randomly break ties).*

Theorem 4. *Let us assume that*

- *the $b(i)$'s (resp. the $w(j)$'s) are randomly independently drawn, according to some probability distribution;*
- *the game has finite horizon;*
- *each deterministic strategy can be generated (i.e. there is a random seed $i > 0$ which generates it).*

Then for any state s, $v^{(K)}(s) \to v(s)$ almost surely as $K \to \infty$.

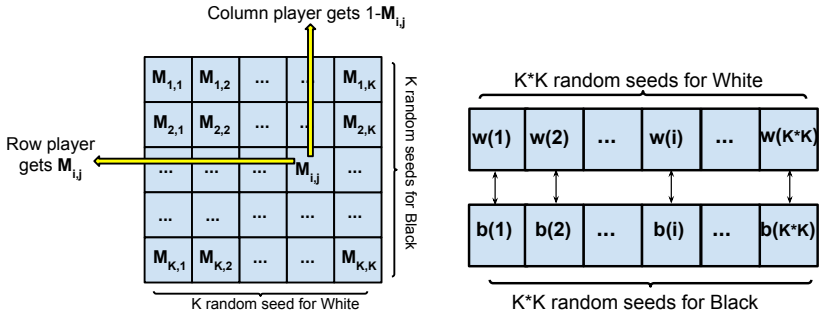
Proof. Consider a state s. Without loss of generality, let us assume that $v(s) = 1$. By Property 5, there is a deterministic strategy for Black which always wins.

For each i, with non-zero probability, the seed $b(i)$ is this perfect strategy for Black. Then, with probability 1, there exists i_0 such that $\forall j$, $M_{i_0,j} = 1$. This implies $\forall K \geq i_0, v^{(K)} = 1$. The result is proved. □

The same proof holds for $v_0^{(K)}$ instead of $v^{(K)}$.

There are various algorithms for computing $v^{(K)}$, including polynomial time [von Stengel, 2002] or sublinear time approximations [Grigoriadis and Khachiyan, 1995, Auer et al., 1995]. At the end, $v^{(K)}$ is a weighted estimate as in Equation (12.2), instead of a uniform average. The classical MC estimate is the Unpaired variant below, but we also include the Paired case as follows:

- **Paired** case: The average of the matrix, if the MC sampling is performed by randomly sampling K random seeds for Black and K random seeds for White and averaging the $K \times K$ corresponding games. Weights are $\forall i \in \{1, \ldots, n\}, w_i = \frac{1}{n}$ with $n = K^2$. See Figure 12.6(a).
- **Unpaired** (standard) case: The average of $K \times K$ games between $K \times K$ randomly drawn random seeds for Black and $K \times K$ randomly drawn seeds for White. Weights are $\forall i \in \{1, \ldots, n\}, w_i = \frac{1}{n}$ with $n = K^2$. See Figure 12.6(b).



(a) Paired case: one random seed for Black per row, one random seed for White per column. The estimate is the average of all values in the matrix.

(b) Unpaired case: $K \times K$ times, a randomly drawn seed for Black plays against a randomly drawn seed for White.

Figure 12.6: Weighted estimate in paired case and unpaired case.

Usually, MC estimates are biased. Therefore the simple mathematics above show that at least for K sufficiently large, $v^{(K)}$ and $v_0^{(K)}$ will outperform the simple MC estimates by reaching optimality. Our experiments will show that far before reaching an exact value as in the theorem above, $v^{(K)}$ is more accurate than *Paired* or *Unpaired* (i.e. usual) MC estimates.

12.3 Experiments on Tsumego

In this section we evaluate the performance of computing a NE (denoted as *Nash* in the figures) compared to 2 baselines (*Paired* MC and *Unpaired* MC - the latter being the classical MC estimate) for different number of simulations (also expressed in matrix size). This comparison is executed for 5 different families of Tsumego problems. Sections 12.3.1-12.3.5 present the results for each family of problems independently. Section 12.3.6 aggregates and discusses the findings.

The first baseline, *Unpaired* (Equation (12.1)), is the classic board evaluation through independent simulations. The second baseline, *Paired*, is an evaluation of the board with simulations where we reuse the same set of seeds for each player. To execute a simulation, we use a MC Tree Search algorithm. From Section 12.3.1 to Section 12.3.6 we focus on 2 settings. The first one (referred as **setting A**) is GnuGo-MCTS [Bayer et al., 2008] with 1 000 simulations per move and the second one (re-

ferred as **setting B**) is GnuGo-MCTS with 80 000 simulations per move. We limited the number of seeds to 31^2, leading to 961 MC simulations.

In Figures 12.7 and 12.8, each figure shows the number of simulations along the x-axis. On the y-axis is displayed the average frequency at which the predictor (*Unpaired*, *Paired*, *Nash*) finds the correct value. The average score is computed as follows. 1 is given for a victory from the Black player and 0 for a victory from the White player. To evaluate a specific Tsumego problem, we execute K^2 simulations. For the *Unpaired* and *Paired* baselines, we compute the average of K^2 simulations whereas *Nash* computes the Nash of the matrix of size $K \times K$. If the value is below 0.5, it means it predicts a victory for the White player and thus we input 0. If the value is over 0.5, it predicts a victory for the Black player and we input 1. For the special case where the value is 0.5, we leave the value as it is. Each selected problem is a victory for the Black player. Thus, the higher the results the better the predictor. Each experiment is repeated 1 000 times.

12.3.1 Oiotoshi - oiotoshi with ko

We identify 10 oiotoshi problems and 4 oiotoshi with ko. For each GnuGo setting, if a problem outputs the same score regardless of the seed, we remove this problem from analysis. For instance, using setting A, we keep 6 oiotoshi problems and 3 oiotoshi with ko. For setting B, we keep 5 oiotoshi problems and all 4 oiotoshi with ko. Typically a problem is removed if it is either too easy to solve (only outputs 1) or too hard (only outputs 0) given a number of simulations. Figure 12.7(a) presents the results. From setting A in Figure 12.7(a), it is important to observe that the *Nash* predictor needs only a submatrix of size $K = 3$ to clearly outperform the others. For instance, with only 9 simulations, it still outperforms the baseline *Unpaired* with 961 simulations by a margin of more than 20 %. The *Paired* predictor performs the worst in this setting. For setting B, the overall results of each predictor are already much higher. Even so, for the same number of simulations the *Nash* predictor outperforms the *Unpaired* by more than 10 % and again, even with a very small number of simulations it predicts well above the other baselines. We recall that the score is directly the frequency at which the Tsumego under consideration is solved.

12.3.2 Seki & Snapback

We identify 3 Seki and 1 Snapback. All of them are considered for both settings A and B. This family is rather small yet it reveals an interesting behavior over the predictors. Figure 12.7(b) presents the results. The setting A from Figure 12.7(b) shows a clear improvement using the $Nash$ predictor over the 2 other baselines. With only 9 simulations it reaches 61 % and peaks at 75 %. Both $Paired$ and $Unpaired$ oscillate between 40 % and 50 %. However, in setting B, the situation is reversed. The $Nash$ predictor decreases in quality as the number of simulation grows whereas the baselines $Paired$ and $Unpaired$ stay relatively stable at 75 %. This behavior of the $Nash$ predictor can be explained on an illustrative example. Let us assume that the situation is a theoretical Win for Black. Let us assume that the probability to generate a correct seed (winning with probability 1) is extremely small for Black, whereas White will play with probability $\frac{1}{3}$ a very good policy which wins in all cases except if Black plays perfectly. Otherwise, with probability $\frac{2}{3}$, White generates a policy which loses almost certainly. Then, unless a good seed has been found for Black (which is unlikely by assumption), the $Nash$ will assume that White wins, as soon as K is large enough for finding a very good policy (probability $\simeq 1 - (\frac{2}{3})^K$). MC, on the other hand, sees Black winning with probability $\frac{2}{3}$, and therefore by law of large numbers will conclude that Black wins. This is basically what happens in this disappointing case.

12.3.3 Kill

There are 10 Kill Tsumego problems selected for this experiment. In setting A, all of them are non trivial whereas in setting B, there are 8 problems left to study. Figure 12.7(c) shows the results. Figure 12.7(c) shows that for the setting A, the $Nash$ predictor outperforms the other 2 baselines by a margin of 20 %. This is an excellent example of situations where traditional predictors, such as the 2 baselines, are very bad at evaluating the value of a game. However, in the setting B, the $Unpaired$ predictor is equally as good as the $Nash$ predictor. The $Paired$ predictor seems to be less effective at predicting the value of a game for the case of kill problem.

12.3.4 Life

There are 10 Life problems selected for this experiment. In setting A, 9 of them are non trivial whereas in setting B, there are 8 problems left to study. Figure 12.7(d) shows the results. It seems that in Figure 12.7(d) for the setting A, the 3 predictors are relatively good with a small advantage for the $Nash$ predictor. Based on the standard deviations ($< 10^{-2}$), when there are more than 289 simulations and less then 841 there is a statistically significant advantage for the $Nash$ predictor. In setting B, the $Nash$ predictor is again significantly better than the 2 other baselines. Results in "kill" and results in "live" differ. For a small number of simulations, such as in setting A, the $Unpaired$ predictor is good at predicting live problems (almost as good as the $Nash$ predictor) but does not perform well for kill problems. The situation is reversed as the number of simulations grows, such as in setting B. This time, the $Unpaired$ predictor becomes better at predicting the kill problems but not so in survival mode (live problems).

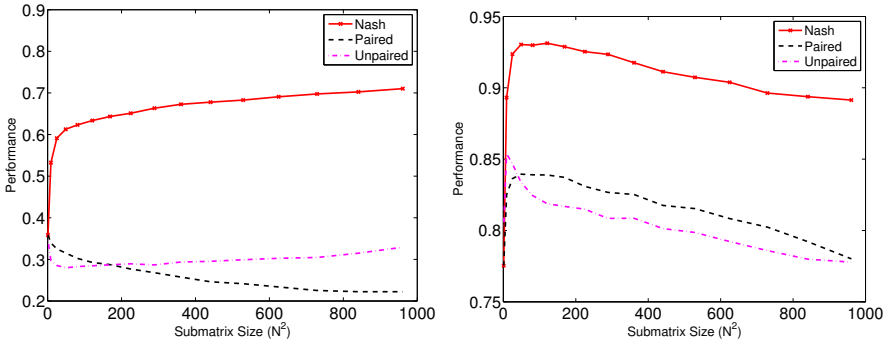
12.3.5 Semeai

There are 10 Semeai Tsumego problems selected for this experiment. In both settings A and B, all of them are non trivial. Figure 12.7(e) shows the results. From setting A in Figure 12.7(e), it appears that all 3 predictors are equally good in their predictions. In setting B however the predictor $Nash$ only needs 9 simulations to significantly outperform the 2 baselines. Perhaps even more importantly, the $Nash$ predictor keeps improving as the number of seeds grows rather than stagnating as the 2 baselines clearly do.

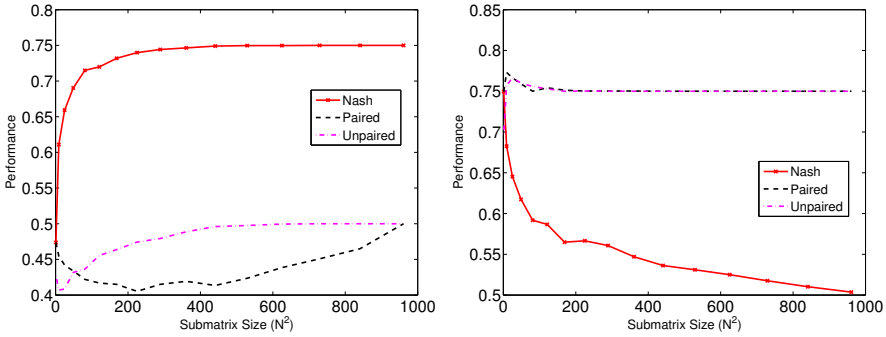
12.3.6 Discussion

We first discuss over the aggregation of the information gathered from the 50 problems to evaluate whether the $Nash$ predictor is an overall improvement over the classic ones. Second we add another baseline, a single MCTS, to further enhance the comparison. Third and last, we provide general conclusions and insights on the results. In both settings we used the full 50 matrix. Figure 12.8 shows the aggregated results.

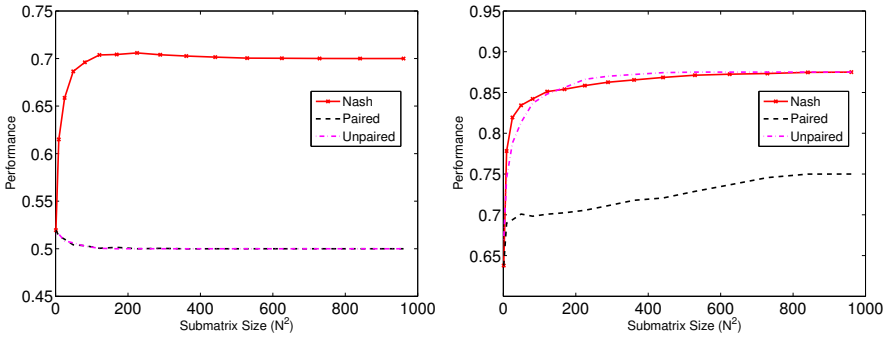
As expected from previous results, for both settings the $Nash$ predictor provides a statistically significant improvement over the baselines $Paired$ and $Unpaired$. Moreover, we can also conclude that the pre-



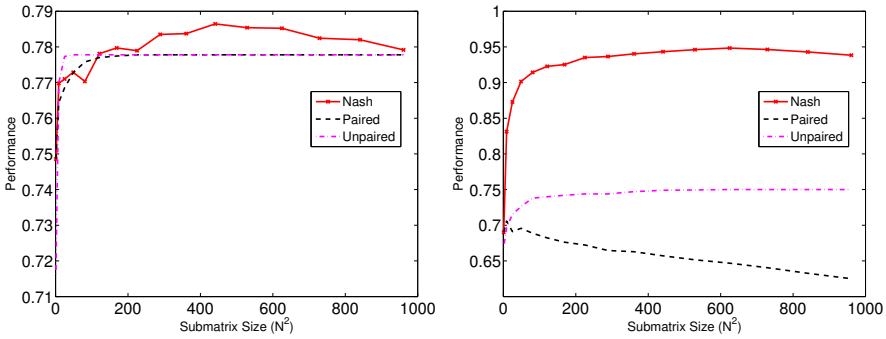
(a) Oiotoshi and Oiotoshi with ko.



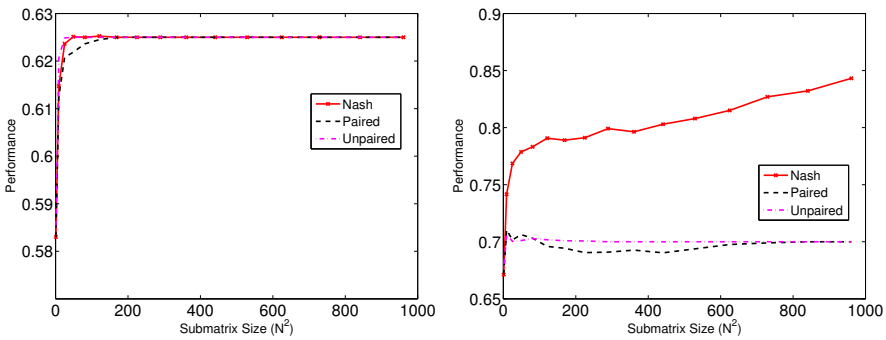
(b) Seki and Snapbacki.



(c) Kill.



(d) Live.



(e) Semeai.

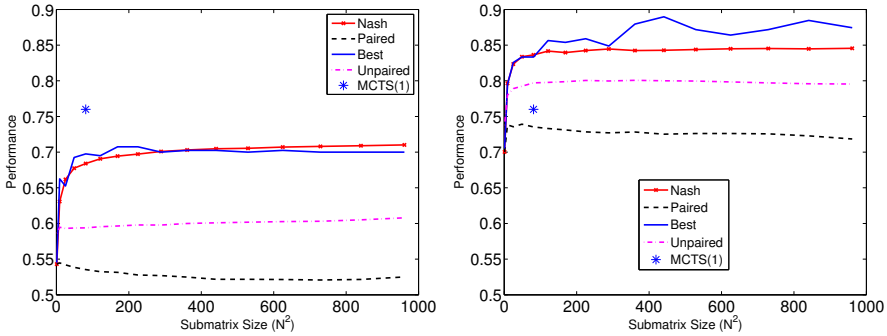


Figure 12.8: Average over 50 problems. These are the results for setting A (Left) and setting B (Right). Standard deviation $< 10^{-2}$, with Nash-portfolio and BestSeed. The MCTS(1) point corresponds to the case of one single MCTS run using all the budget in terms of number of simulations. We see that MCTS(1) outperforms our weighted average of 81 MCTS runs in the setting A; however, it is less parallel. On the setting B, corresponding to larger numbers of simulations, we see that our approach outperforms MCTS by far - this is consistent with the limited scalability of MCTS for huge numbers of simulations. The result for setting B (right) therefore shows that our approach outperforms MCTS for large numbers of simulations. For each value on the x-axis, methods are compared for the same number of simulations - this is actually a bit unfair *against* our proposal, as a single run of (i.e. method MCTS(1), to which our methods are compared) is slower than a combination of several smaller MCTS because (i) the speed of simulations decreases significantly with large numbers of simulations and (ii) our method is far more parallel.

dictor *Unpaired* is better than the *Paired* one. The example of the kill and live problems leads to the possibility of adjusting the number of simulations based upon the problem at hand. The fact that the *Nash* predictor usually needs only a small fraction of simulations to, generally speaking, significantly outperform classic predictors such as *Paired* and *Unpaired* is a potential important improvement in board evaluations.

Also, we compare a single MCTS (labeled $MCTS(1)$) with our approach. Since the building of a matrix M effectively splits up the total number of simulations available, one can wonder whether a sim-

ple MCTS that consumes the entire budget is more efficient than our weighted average of several MCTS. Our method has other advantages, as our simulations are more parallel. In setting A, each independent MCTS run is a MCTS with 1 000 simulations. For a fair comparison, the single MCTS is given $81 \times 1\ 000$ simulations for performing its evaluation. In setting B, the single MCTS is given $81 \times 80\ 000$ simulations to output its evaluation. The results are plotted in Figure 12.8, with x-axis corresponding to the given budget. In setting A it seems that a single MCTS outperforms the approach proposed in this chapter whereas in setting B there is a clear advantage for the Nash approach. This can be explained by the fact that after a certain number of simulations, bluntly increasing the number of simulations does not improve the performance of a single MCTS. Figure 12.8 shows that a Nash approach yields better results in such case.

12.4 Conclusion: weighted MCTS has a better scalability than MCTS

In this chapter we study a novel way of evaluating game values using the principle of Nash Equilibrium. First we provide a theoretical validation of the approach. Second we apply this methodology to 50 different Tsumego problems and show that the $Nash$ predictor significantly outperforms the classical baselines such as $Paired$ and $Unpaired$ in most cases. The only case where the $Nash$ predictor is inferior in terms of performance is explained and discussed. Overall, compared to classical Monte Carlo, the $Nash$ predictor, in spite of its only asymptotic mathematical guarantee, requires a number of simulations several orders of magnitude smaller to output a predictor that is still significantly better than the baseline predictors.

Importantly, we did not only outperform a simple averaging of smaller MCTS runs. We also outperformed (see Figure 12.8, right) a single MCTS run using, alone, the same number of simulations as all the single MCTS runs of our averaging. This shows that for large numbers of simulations, our approach outperforms MCTS for evaluating positions. In short, for N large enough, our weighted averaging of 81 single MCTS runs with N simulations is better than a MCTS run with $81N$ simulations.

As a conclusion, our methodology, based on the reweighting of sim-

ulations, outperformed a standard MCTS implementation in Go, for large numbers of simulations, without computational overhead.

Part IV

Portfolios and noisy optimization

13 Portfolios and noisy optimization: outline

Portfolios usually benefit from the huge difference between the cost of comparing N solutions and determining N solutions: comparing solutions is by far cheaper than computing solutions, hence portfolio methods are efficient as soon as the efficiency of different methods highly depend on the problem under work. The case of noisy optimization is quite different; here, comparing solutions is expensive. We will see that portfolio methods are nonetheless quite efficient in the noisy framework.

Chapter 14 considers the case of noisy optimization solvers in continuous optimization. Chapter 15 compares differential evolution to classical bandit algorithms on a black-box portfolio selection problem.

14 Portfolio of noisy optimization methods

14.1 Outline of this chapter

While applying portfolio methods is usual in combinatorial optimization, this chapter is devoted to the new application in noisy optimization. Section 14.2 describes the algorithms under consideration and provides theoretical results. Section 14.3 is dedicated to experimental results. Section 14.4 concludes.

14.2 Algorithms and analysis

Section 14.2.1 introduces some notations. Section 14.2.2 provides some background and criteria. Section 14.2.3 describes two portfolio algorithms, one with fair distribution of budget among solvers and one with unfair distribution of budget. Section 14.2.4 then provides theoretical guarantees.

14.2.1 Notations

In this section, $\mathbb{N}^* = \{1, 2, 3, \dots\}$, "a.s." stands for "almost surely", i.e., with probability 1, and "s.t." stands for "such that". A summary of notations can be found at the end of the manuscript. If X is a random variable, then $(X^{(1)}, X^{(2)}, \dots)$ denotes a sample of independent identically distributed random variables, copies of X. $o(.), O(.), \Theta(.)$ are the standard Landau notations. $\mathcal{N}$ denotes a standard Gaussian distribution, in dimension given by the context.

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 $(x, w) \mapsto f(x, w)$ where $x \in \mathcal{D}$ and w is a random variable independently sampled at each call to f. The user can only choose x. For short, we will use the notation $f(x)$. The reader should keep in mind that this function is stochastic. $\hat{\mathbb{E}}_s [f(x)]$ denotes the empirical evaluation of $\mathbb{E}[f(x)]$ over $s \in \mathbb{N}^*$ resamplings, i.e., $\hat{\mathbb{E}}_s [f(x)] = \frac{1}{s} \sum_{j=1}^{s} (f(x))^{(j)}$.

14.2.2 Definitions and Criteria

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 has the same goal, and can use $M \in \{2, 3, \dots\}$ different given solvers. A good AS tool should ensure that it is nearly as efficient as the best of the individual solvers[1], for any problem in some class of interest.

14.2.2.1 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 $x_1, \dots, x_n$ (i.e., after spending n evaluations from the budget).

Some algorithms make no difference between x_n and $\tilde{x}_n$, but in the general case of a noisy optimization setting the difference matters [Coulom, 2012, Fabian, 1967, Shamir, 2013].

We recall the Simple Regret (SR) and the slope of SR for noisy optimization (defined in Section 2.1.1). The SR is formalized as follows:

$$SR_n = \mathbb{E}\left(f(\tilde{x}_n) - f(x^*)\right). \tag{14.1}$$

SR_n is the simple regret after n evaluations; n is then the budget. The $\mathbb{E}$ operator refers to the w part, i.e., with complete notations,

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

[1] A solver is termed "individual solver" when it is not a portfolio. In this section, unless stated otherwise, a "solver" is an "individual solver".

In many cases, it is known that the simple regret has a linear convergence in a log-log scale [Fabian, 1967, Chen, 1988, Coulom, 2012]. Therefore we will consider this slope. The slope of the simple regret is then defined as

$$s(SR) = \lim_{n\to\infty} \frac{\log(SR_n)}{\log(n)}, i \tag{2.1}$$

where the limit holds almost surely, since SR_n is a random variable.

For example, the gradient method proposed in [Fabian, 1967] (approximating the gradient by finite differences) reaches a simple regret slope arbitrarily close to -1 on sufficiently smooth problems, for an additive centered noise, without assuming variance decreasing to zero around the optimum.

14.2.2.2 Simple regret criterion for portfolio.

For a portfolio algorithm in the black-box setting, $\forall i \in \{1, \ldots, M\}$, $\tilde{x}_{i,n}$ denotes the point

- that the solver number i *recommends* as an approximation of the optimum;
- after this solver has spent n evaluations from the budget.

Similarly, the simple regret given by Equation 14.1 corresponding to solver number i after n evaluations (i.e., after solver number i has spent n evaluations), is denoted by $SR_{i,n}$. For $n \in \mathbb{N}^*$, i^*_n denotes the solver chosen by the selection algorithm when there are at most n evaluations per solver.[2]

Another important concept is the difference between the two kinds of terms in the regret of the portfolio. We distinguish these two kinds of terms in the next two definitions:

Definition 6 (Solvers' regret)**.** *The solvers' regret with index n, denoted by $SR_n^{Solvers}$, is the minimum simple regret among the solvers after at most n evaluations each, i.e., $SR_n^{Solvers} = \min_{i\in\{1,\ldots M\}} SR_{i,n}$.*

Definition 7 (Selection regret)**.** *The selection regret with index n, denoted by $SR_n^{Selection}$ includes the additional regret due to mistakes in*

[2] This is not uniquely defined, as there might be several time steps at which the maximum number of evaluations in a solver is n; however, the results in the rest of this section are independent of this subtlety.

choosing among these M solvers after at most n evaluations each, i.e., $SR_n^{Selection} = \mathbb{E}\left(f(\tilde{x}_{i^*_n,n}) - f(x^*)\right)$.

Similarly, $\Delta_{i,n}$ quantifies the regret for choosing solver i at iteration n.

Definition 8. *For any solver $i \in \{1,\dots,M\}$ and any number of evaluations $n \in \mathbb{N}^*$, we denote by $\Delta_{i,n}$ the quantity:* $\Delta_{i,n} = SR_{i,n} - \min_{j\in\{1,\dots,M\}} SR_{j,n}$.

Finally, we consider a function that will be helpful for defining our portfolio algorithms.

Definition 9 (LAG function)**.** *A lag function* LAG : $\mathbb{N}^* \to \mathbb{N}^*$ *is a non-decreasing function such that for all $n \in \mathbb{N}^*$,* LAG$(n) \leq n$.

14.2.3 Portfolio algorithms

In this section, we present two AS methods. A first version, in Section 14.2.3.1, shares the computational budget uniformly; a second version has an unfair sharing of computation budget, in Section 14.2.3.2.

14.2.3.1 Simple Case : Uniform Portfolio NOPA

We present in Algorithm 15 a simple noisy optimization portfolio algorithm (NOPA) which does not apply any sharing and distributes the computational budget equally over the noisy optimization solvers.

In this NOPA algorithm, we compare, at iteration r_n, recommendations chosen at iteration LAG(r_n), and this comparison is based on s_n resamplings, where n is the number of algorithm selection steps. We have designed the algorithm as follows:

- **A stable choice of solver:** The selection algorithm follows the recommendation of the same solver i^* at all iterations in $\{r_n,\dots,r_{n+1}-1\}$. This choice is based on comparisons between old recommendations (through the *lag* function LAG).
- **The chosen solver updates are taken into account.** For iteration indices $m < p$ in $\{r_n,\dots,r_{n+1}-1\}$, the portfolio chooses the same solver i^*, but does not necessarily recommends the same point because possibly the solver changes its recommendation, i.e., possibly $\tilde{x}_{i^*,m} \neq \tilde{x}_{i^*,p}$.

Algorithm 15 Noisy Optimization Portfolio Algorithm (NOPA).

Input: noisy optimization solvers $Solver_1, Solver_2, \ldots, Solver_M$
Input: a *lag* function $\text{LAG} : \mathbb{N}^* \mapsto \mathbb{N}^*$ ▷ As in Definition 9
Input: a non-decreasing integer sequence $r_1, r_2, \ldots$ ▷ Periodic comparisons
Input: a non-decreasing integer sequence $s_1, s_2, \ldots$ ▷ Number of resamplings

$n \leftarrow 1$ ▷ Number of selections
$m \leftarrow 1$ ▷ NOPA's iteration number
$i^* \leftarrow null$ ▷ Index of recommended solver
$x^* \leftarrow null$ ▷ Recommendation
while budget is not exhausted **do**
  if $m \geq r_n$ **then**
    $i^* = \underset{i \in \{1,\ldots,M\}}{\arg\min}\, \hat{\mathbb{E}}_{s_n}[f(\tilde{x}_{i,\text{LAG}(r_n)})]$ ▷ Algorithm selection
    $n \leftarrow n + 1$
  else
    for $i \in \{1, \ldots, M\}$ **do**
      Apply one evaluation for $Solver_i$
    end for
    $m \leftarrow m + 1$
  end if
  $x^* = \tilde{x}_{i^*,m}$ ▷ Update recommendation
end while

Effect of the lag. Due to the LAG(.) function, we compare the $\tilde{x}_{i,\text{LAG}(r_n)}$ (for $i \in \{1, \ldots, M\}$), and not the $\tilde{x}_{i,r_n}$. This is the key point of this algorithm. Comparing the $\tilde{x}_{i,\text{LAG}(r_n)}$ is much cheaper than comparing the $\tilde{x}_{i,r_n}$, because the fitness values are not yet that good at iteration $\text{LAG}(r_n)$, so they can be compared faster - i.e., with less evaluations - than recommendations at iteration r_n. We will make this more formal in Section 14.2.4, and see under which assumptions this lag has more pros than cons, namely when algorithms have somehow sustained rates. In addition, with lag, we can define INOPA, which saves up significant parts of the computation time.

The first step for formalizing this is to understand the two different kinds of evaluations in portfolio algorithms for noisy optimization. Contrarily to noise-free settings, comparing recommendations requires

a dedicated budget, which is far from negligible. It follows that there are two kinds of evaluations:

- **Portfolio budget (Algorithm 15, Lines 10-12):** this corresponds to the M evaluations per iteration, dedicated to running the M solvers (one evaluation per solver and per iteration).
- **Comparison budget (Algorithm 15, Line 7):** this corresponds to the s_n evaluations per solver at the n^{th} algorithm selection. 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 $M \cdot r_n$ evaluations in the portfolio budget for the r_n first iterations. We will see below (Section 14.2.4) conditions under which the other costs can be made negligible, whilst preserving the same regret as the best of the M solvers.

14.2.3.2 INOPA: Improved NOPA, with unequal budget

Algorithm 16 proposes a variant of NOPA, which distributes the budget in an unfair manner. The solvers with good performance receive a greater budget.

14.2.4 Theoretical analysis

We here show

- a bound on the performance of NOPA (Section 14.2.4.2);
- a bound on the performance of INOPA (Section 14.2.4.3);
- that the *lag* term is necessary (Section 14.2.4.4).

14.2.4.1 Preliminary

We define 2 extra properties which are central in the proof.

Definition 10 ($P_{as}^{(i)}((\epsilon_n)_{n\in\mathbb{N}^*})$)**.** *For any solver $i \in \{1, \ldots, M\}$, for some positive sequence $(\epsilon_n)_{n\in\mathbb{N}^*}$, we define $P_{as}^{(i)}((\epsilon_n)_{n\in\mathbb{N}^*})$:*

$$P_{as}^{(i)}((\epsilon_n)_{n\in\mathbb{N}^*}) : a.s. \quad \exists n_0,\ \forall n_1 \geq n_0,\ \Delta_{i,n_1} < 2\epsilon_{n_1} \implies \forall n_2 \geq n_1,\ \Delta_{i,n_2} < 2\epsilon_{n_2}.$$

Algorithm 16 Improved Noisy Optimization Portfolio Algorithm (INOPA).

Input: noisy optimization solvers $Solver_1, Solver_2, \ldots, Solver_M$
Input: a *lag* function $\text{LAG} : \mathbb{N}^* \mapsto \mathbb{N}^*$ ▷ Refer to Definition 9
Input: a non-decreasing positive integer sequence $r_1, r_2, \ldots$ ▷ Periodic comparisons
Input: a non-decreasing integer sequence $s_1, s_2, \ldots$ ▷ Number of resamplings

$n \leftarrow 1$ ▷ Number of selections
$m \leftarrow 1$ ▷ NOPA's iteration number
$i^* \leftarrow null$ ▷ Index of recommended solver
$x^* \leftarrow null$ ▷ Recommendation
while budget is not exhausted **do**
  if $m \geq \text{LAG}(r_n)$ or $i^* = null$ **then**
    $i^* = \underset{i \in \{1,\ldots,M\}}{\arg\min}\, \hat{\mathbb{E}}_{s_n}[f(\tilde{x}_{i,\text{LAG}(r_n)})]$ ▷ Algorithm selection
    $m' \leftarrow r_n$
    while $m' < r_{n+1}$ **do**
      Apply one evaluation to solver i^*
      $m' \leftarrow m' + 1$
      $x^* = \tilde{x}_{i^*,m'}$ ▷ Update recommendation
    end while
    $n \leftarrow n + 1$
  else
    for $i \in \{1, \ldots, M\} \backslash i^*$ **do**
      Apply $\text{LAG}(r_n) - \text{LAG}(r_{n-1})$ evaluations for $Solver_i$
    end for
    $m \leftarrow m + 1$
  end if
end while

Informally speaking, if $P_{as}^{(i)}((\epsilon_n)_{n\in\mathbb{N}^*})$ is true, then almost surely for a large enough number of evaluations, the difference between the simple regret of solver $i \in \{1,\dots,M\}$ and the optimal simple regret is either always at most $2\epsilon_n$ or always larger - there is no solver infinitely often alternatively at the top level and very weak.

Definition 11 ($P_{as}((\epsilon_n)_{n\in\mathbb{N}^*})$)**.** *For some positive sequence $(\epsilon_n)_{n\in\mathbb{N}^*}$, we define $P_{as}((\epsilon_n)_{n\in\mathbb{N}^*})$ as follows:*

$$\forall i \in \{1,\dots,M\}\,,\ P_{as}^{(i)}((\epsilon_n)_{n\in\mathbb{N}^*}) \text{ holds.}$$

Remark 5. *In Definitions 10 and 11, we might choose slightly less restrictive definitions, for which the inequalities only hold for integers n such that $\exists i$, $\text{LAG}(r_i) = n$ or $r_i = n$.*

Definitions above can be applied in a very general setting. The simple regret of some noisy optimization solvers, for instance Fabian's algorithm, is almost surely $SR_n \leq (1+o(1))\frac{C}{n^\alpha}$ after $n \in \mathbb{N}^*$ evaluations (C is a constant), for some constant $\alpha > 0$ arbitrarily close to 1. This result is proved in [Fabian, 1967], with optimality proved in [Chen, 1988].

We prove the following proposition for such a case; it will be convenient for illustrating "abstract" general results to standard noisy optimization frameworks.

Proposition 1. *Assume that each solver $i \in \{1,\dots,M\}$ has almost surely simple regret $(1+o(1))\frac{C_i}{n^{\alpha_i}}$ after $n \in \mathbb{N}^*$ evaluations.*

We define C, α^*, C^*:

$$C = \frac{1}{3}\min\left\{|C_i - C_j| \mid 1 \leq i,j \leq M; C_i - C_j \neq 0\right\}. \tag{14.2}$$

$$\alpha^* = \max_{i\in\{1,\dots,M\}} \alpha_i. \tag{14.3}$$

$$C^* = \min_{i\in\{1,\dots,M\} \text{ s.t. } \alpha_i=\alpha^*} C_i. \tag{14.4}$$

We also define the set of optimal solvers:

$$\begin{aligned} SetOptim &= \{i \in \{1,\dots,M\} | \alpha_i = \alpha^*\} \\ \text{and } SubSetOptim &= \{i^* \in SetOptim | C_{i^*} = C^*\} \qquad (14.5) \\ &= \{i \in \{1,\dots,M\} | \alpha_i = \alpha^* \text{ and } C_i = C^*\} \qquad (14.6) \end{aligned}$$

With these notations, if almost surely, $\forall i \in \{1, \ldots, M\}$*, the simple regret for solver* i *after* $n \in \mathbb{N}^*$ *evaluations is* $SR_{i,n} = (1 + o(1))\frac{C_i}{n^{\alpha_i}}$*, then* $P_{as}((\epsilon_n)_{n\in\mathbb{N}^*})$ *is true with* ϵ_n *defined as follows:*

$$\epsilon_n = \frac{C}{n^{\alpha^*}}. \tag{14.7}$$

Moreover, if $i_0 \in \{1, \ldots, M\}$ *satisfies:* $(\exists n_0 \in \mathbb{N}^*,\ \forall n \geq n_0,\ \Delta_{i_0,n} \leq 2\epsilon_n)$*, then* $i_0 \in SubSetOptim$*.*

Informally speaking, this means that if the solver i_0 is close, in terms of simple regret, to an optimal solver (i.e., a solver matching α^* and C^* in Equations 14.3 and 14.4), then it also has an optimal slope ($\alpha_{i_0} = \alpha^*$) and an optimal constant ($C_{i_0} = C^*$).

14.2.4.2 The $\log(M)$-shift for NOPA

We can now enunciate the first main theorem, stating that there is, with fair sharing of the budget as in NOPA, a $\log(M)$-shift, i.e., on a log-log scale (x-axis equal to the number of evaluations and y-axis equal to the log of the simple regret), the regret of the portfolio is just shifted by $\log(M)$ on the x-axis.

Theorem 5 (Regret of NOPA: the $\log(M)$ shift)**.** *Let* $(r_n)_{n\in\mathbb{N}^*}$ *and* $(s_n)_{n\in\mathbb{N}^*}$ *be two non-decreasing integer sequences. Assume that:*

- $\forall x \in \mathcal{D},\ Var\ f(x) \leq 1$*;*
- *for some positive sequence* $(\epsilon_n)_{n\in\mathbb{N}^*}$*,* $P_{as}((\epsilon_n)_{n\in\mathbb{N}^*})$ *(Definition 11) is true.*

  Then, there exists n_0 *such that:*

$$\forall n \geq n_0,\ SR_{r_n}^{Selection} < SR_{r_n}^{Solver} + 2\epsilon_{r_n} \tag{14.8}$$

$$\text{with probability at least } 1 - \frac{M}{s_n \epsilon^2_{\mathrm{LAG}(r_n)}}$$

$$\text{after } e_n = r_n \cdot M \cdot \left(1 + \sum_{i=1}^{n} \frac{s_i}{r_n}\right) \text{ evaluations.}$$

Moreover, if (s_n), $\text{LAG}(n)$, (r_n) *and* (ϵ_n) *satisfy* $\sum_{j=1}^{\infty} \frac{1}{s_j \epsilon^2_{\text{LAG}(r_j)}} < \infty$, *then, almost surely, there exists* n_0 *such that:*

$$\forall n \geq n_0,\ SR^{Selection}_{r_n} < SR^{Solver}_{r_n} + 2\epsilon_{r_n} \tag{14.9}$$

$$\textit{after}\ \ e_n = r_n \cdot M \cdot \left(1 + \sum_{i=1}^{n} \frac{s_i}{r_n}\right)\ \ \textit{evaluations.}$$

Remark 6. *Please notice that Equation 14.8 holds* with a given probability *whereas Equation 14.9 holds* almost surely. *The almost sure convergence in the assumption is proved for some noisy optimization algorithms [Fabian, 1967].*

We now use Proposition 1 to apply Theorem 5 on a classical case with almost sure convergence.

Application 1 ($\log(M)$ shift)**.** *Assume that for any solver* $i \in \{1, \ldots, M\}$, *the simple regret after* $n \in \mathbb{N}^*$ *evaluations is* $SR_{i,n} = (1 + o(1))\frac{C_i}{n^{\alpha_i}}$. *We define* $\epsilon_n = \frac{C}{n^{\alpha^*}}$ *(where* C *and* α^* *are defined as in Equations 14.2 and 14.3). Assume that* $\forall x \in \mathcal{D}$, $\mathcal{V}ar\ f(x) \leq 1$ *and that* (s_n), $(\text{LAG}(n))$ *and* (r_n) *satisfy:*

$$\begin{array}{rll} & \sum_{j=1}^{\infty} & \dfrac{1}{s_j \epsilon^2_{\text{LAG}(r_j)}} < \infty \\ \textit{and} & \sum_{i=1}^{n} & s_i = o(r_n). \end{array}$$

Then, almost surely,

i) for n *large enough,* $SR^{Selection}_{r_n} < SR^{Solver}_{r_n} + 2\epsilon_{r_n}$ *after* $e_n = r_n \cdot M \cdot \left(1 + \sum_{i=1}^{n} \frac{s_i}{r_n}\right)$ *function evaluations;*

ii) for n *large enough,* $SR^{Selection}_{r_n} \leq \max_{i \in SubSetOptim} SR_{i,r_n}$ *after* $e_n = r_n \cdot M \cdot \left(1 + \sum_{i=1}^{n} \frac{s_i}{r_n}\right)$ *function evaluations;*

iii) the slope of the selection regret verifies $\lim_{n\to\infty} \frac{\log(SR^{Selection}_{r_n})}{\log(e_n)} = -\alpha^*$.

$SR^{Selection}_{r_n}$ corresponds to the simple regret at iteration r_n of the portfolio, which corresponds to $e_n = r_n \cdot M \cdot \left(1 + \sum_{i=1}^{n} \frac{s_i}{r_n}\right)$ evaluations in the portfolio - hence the comment "after e_n function evaluations".

Example 1. *The following parametrization matches the conditions in Application 1.*

$$\begin{aligned} r_n &= \lceil n^{3+r+r'} \rceil; \\ \text{LAG}(n) &= \lceil \log(n) \rceil; \\ s_n &= \lceil n^{1+r'} \rceil, r > 0 \text{ and } r' \geq 1, n \in \mathbb{N}^*. \end{aligned}$$

14.2.4.3 The $\log(M')$-shift for INOPA

We now show that INOPA, which distributes the budget in an unfair manner, can have an improvement over NOPA. Instead of a factor M (number of solvers in the portfolio), we get a factor M', number of approximately optimal solvers. This is formalized in the following theorem:

Theorem 6 ($\log(M')$ shift)**.** *Let $(r_n)_{n\in\mathbb{N}^*}$ and $(s_n)_{n\in\mathbb{N}^*}$ two non-decreasing integer sequences. Assume that:*

- $\forall x \in \mathcal{D},\ Var\ f(x) \leq 1$*;*
- *for some positive sequence $(\epsilon_n)_{n\in\mathbb{N}^*}$, $P_{as}((\epsilon_n)_{n\in\mathbb{N}^*})$ (Definition 11) holds.*

We define $S = \{i | \exists n_0 \in \mathbb{N}^*, \forall n \geq n_0, \Delta_{i,n} < 2\epsilon_n\}$ and M' denotes the cardinality of the set S, i.e., $M' = |S|$. Then, there exists n_0 such that:

$$\forall n \geq n_0,\ SR_{r_n}^{Selection} < SR_{r_n}^{Solver} + 2\epsilon_{r_n} \tag{14.10}$$

$$\text{with probability at least } 1 - \frac{M}{s_n \epsilon^2_{\text{LAG}(r_n)}}$$

$$\text{after } e_n = r_n \cdot M' \cdot \left(1 + \frac{M}{M'} \sum_{i=1}^{n} \frac{s_i}{r_n}\right) + (M - M')\text{LAG}(r_n) \text{ evaluations.}$$

Then, if (s_n), $(\text{LAG}(n))$, (r_n) and (ϵ_n) satisfy $\sum_{j=1}^{\infty} \frac{1}{s_j \epsilon^2_{\text{LAG}(r_j)}} < \infty$, $\text{LAG}(n) = o(n)$ and $\sum_{j=1}^{n} s_j = o(r_n)$, then, almost surely, there exists n_0 such that:

$$\forall n \geq n_0,\ SR_{r_n}^{Selection} < SR_{r_n}^{Solver} + 2\epsilon_{r_n} \tag{14.11}$$

$$\text{after } e_n = r_n \cdot M' \cdot (1 + o(1)) \text{ evaluations.}$$

Using Proposition 1, we apply Theorem 6 above to the case of linearly convergent optimization solvers (linear in a log-log scale, with x-axis logarithmic of the number of evaluations and y-axis logarithmic of the simple regret).

Application 2 ($\log(M')$shift)**.** *Assume that* $\forall x \in \mathcal{D}$, $\mathcal{V}ar\ f(x) \leq 1$ *and for any solver* $i \in \{1, \dots, M\}$*, the simple regret after* $n \in \mathbb{N}^*$ *evaluations is* $SR_{i,n} = (1 + o(1))\frac{C_i}{n^{\alpha_i}}$*. We define* $\epsilon_n = \frac{C}{n^{\alpha^*}}$ *with* C *and* α^* *defined as in Equation 14.2 and 14.3. If* $(s_n)_{n\in\mathbb{N}^*}$, $\text{LAG}(n)_{n\in\mathbb{N}^*}$, $(r_n)_{n\in\mathbb{N}^*}$ *and* $(\epsilon_n)_{n\in\mathbb{N}^*}$ *are chosen such that* $\sum_{j=1}^{\infty} \frac{1}{s_j \epsilon^2_{\text{LAG}(r_j)}} < \infty$, $\text{LAG}(n) = o(n)$ *and* $\sum_{j=1}^{n} s_j = o(r_n)$*, then, almost surely, there exists* n_0 *such that:*

i) $\forall n \geq n_0$, $SR^{Selection}_{r_n} < SR^{Solver}_{r_n} + 2\epsilon_{r_n}$ *after* $e_n = M' \cdot r_n(1+o(1))$ *evaluations;*

ii) $\forall n \geq n_0$, $SR^{Selection}_{r_n} \leq \max_{i \in SubSetOptim} SR_{i,r_n}$ *after* $e_n = M' \cdot r_n(1 + o(1))$ *evaluations;*

iii) the slope of the selection regret verifies $\lim_{n\to\infty} \frac{\log(SR^{Selection}_{r_n})}{\log(e_n)} = -\alpha^*$.

As usual, $SR^{Selection}_{r_n}$ corresponds to the simple regret at iteration r_n of the portfolio, which corresponds to $e_n = r_n \cdot M' \cdot (1 + o(1))$ evaluations in the portfolio - hence the comment "after e_n function evaluations".

Example 2. *(log(M') shift) The parametrization of Example 1 also matches the assumptions of Application 2.*

14.2.4.4 The *lag* is necessary

In this section, we show that, if there is no *lag* (i.e., $\forall n$, $\text{LAG}(n) = n$) whenever there are only two solvers, and whenever these solvers have different slopes, the portfolio algorithm might not have a satisfactory behavior, in the sense that, in the example below, it will select infinitely often the worst solver - unless s_n is so large that the comparison budget is not small compared to the portfolio budget.

Example 3 (The *lag* is necessary)**.** *Let us consider the behavior of NOPA without* lag*. We assume the following:*

- *no lag:* $\forall n \in \mathbb{N}^*$, $\text{LAG}(r_n) = r_n$.
- *the noise is a standard normal distribution* $\mathcal{N}$*;*

- *there are $M = 2$ solvers and the two solvers of the portfolio are such that, almost surely, $SR_{i,m} = (1 + o(1))\frac{C_i}{m^{\alpha_i}}$ after $m \in \mathbb{N}^*$ evaluations, $i \in \{1, 2\}$, with $\alpha_1 = 1 - e$ and $\alpha_2 = 1 - 2e$, where $e \in [0, 0.5)$ is a constant.*
- *The comparison budget is moderate compared to the portfolio budget, in the sense that*
$$s_n = O(r_n^\beta) \tag{14.12}$$
with $\beta \leq 2 - 4e$.

Then, almost surely, the portfolio will select the wrong solver infinitely often.

14.3 Experimental results

This section is organized as follows. Section 14.3.1 introduces another version of algorithms, more adapted to some particular implementations of solvers. Section 14.3.2 describes the different solvers contained in the portfolios and some experimental results. Section 14.3.3 describes the similar solvers contained in the portfolios and some experimental results. In all tables, CT refers to the computation time and NL refers to "no *lag*". "s" as a unit refers to "seconds".

14.3.1 Real world constraints & introducing sharing

The real world introduces 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 Algorithm 15 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 detailed in Algorithm 17.

This is an adapted version of NOPA for coarse grain, i.e., in case the solvers can not be restricted to doing one fitness evaluation at a time. The adaptation of INOPA is straightforward. We now present experiments with this adapted algorithm.

With/without lag. In this section, unless specified otherwise, the portfolio *with lag* means $\forall n \in \mathbb{N}^*, \text{LAG}(n) < n$. For the portfolio *without*

Algorithm 17 Adapted version of NOPA in real world constraints.

Input: noisy optimization solvers $Solver_1, Solver_2, \ldots, Solver_M$
Input: a *lag* function $\text{LAG} : \mathbb{N}^* \mapsto \mathbb{N}^*$ ▷ Refer to Definition 9
Input: a non-decreasing integer sequence $r_1, r_2, \ldots$ ▷ Periodic comparisons
Input: a non-decreasing integer sequence $s_1, s_2, \ldots$ ▷ Number of resamplings
Input: a boolean *sharing*
$n \leftarrow 1$ ▷ Number of selections
$i^* \leftarrow null$ ▷ Index of recommended solver
$x^* \leftarrow null$ ▷ Recommendation
$R \leftarrow 0^M$ ▷ Vector of number of evaluations
while budget is not exhausted **do**
  if $\min_{i \in \{1,\ldots,M\}} R_i \geq r_n$ **then**
    $i^* = \arg\min_{i \in \{1,\ldots,M\}} \hat{\mathbb{E}}_{s_n}[f(\tilde{x}_{i,\text{LAG}(r_n)})]$ ▷ Algorithm selection
    $x^* = \tilde{x}_{i^*,R_{i^*}}$ ▷ Update recommendation
    if *sharing* **then**
      All solvers receive x^* as next iterate
    end if
    $n \leftarrow n + 1$
  else
    for $i \in \{1, \ldots, M\}$ **do**
      while $R_i < r_n$ **do**
        Apply one iteration for $Solver_i$, increase R_i by spent evaluation number
      end while
    end for
  end if
end while
$x^* = \tilde{x}_{i^*,R_{i^*}}$ ▷ Update recommendation

lag, at the n^{th} algorithm selection, we compare $\tilde{x}_{i,r_n}$ instead of $\tilde{x}_{i,\text{LAG}(r_n)}$. This means that we choose the identity as a *lag* function LAG, i.e., $\forall n \in \mathbb{N}^*, \text{LAG}(n) = n$.

14.3.2 Experiments with different solvers in the portfolio

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

- Solver 1: A self-adaptive (μ,λ) evolution strategy with resampling as explained in Algorithm 20, with parametrization $\lambda = 10d$, $\mu = 5d$ (in dimension d). The number of resamplings at iteration n is $N_{resample}(n) = 10n^2$. This solver will be termed $RSAES$ (resampling self-adaptive evolution strategy).
- Solver 2: Fabian's solver, as detailed in Algorithm 21, with parametrization $\gamma = 0.1$, $a = 1$, $c = 100$. This variant will be termed $Fabian1$.
- Solver 3: Another Fabian's solver with parametrization $\gamma = 0.49$, $a = 1$, $c = 2$. This variant will be termed $Fabian2$.
- Solver 4: 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 Algorithm 22, with parametrization $B = 1$, $\beta = 2$, $A = 100$, $\alpha = 4$. For short this solver will be termed $Newton$.
- NOPA NL: NOPA of solvers 1-4 *without lag*. Functions are $r_n = \lceil n^{4.2} \rceil$, $\text{LAG}(n) = n$, $s_n = \lceil n^{2.2} \rceil$ at n^{th} algorithm selection.
- NOPA: NOPA of solvers 1-4. Functions are $r_n = \lceil n^{4.2} \rceil$, $\text{LAG}(n) = \lceil n^{1/4.2} \rceil$, $s_n = \lceil n^{2.2} \rceil$ at n^{th} algorithm selection.
- NOPA+S.: NOPA of solvers 1-4, with information sharing enabled. Same functions.
- INOPA: INOPA of solvers 1-4. Same functions.
- INOPA+S.: INOPA of solvers 1-4, with information sharing enabled. Same functions.

Consistently with Equation 2.2, we evaluate 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.

14.3.2.1 Experiments in unimodal case

The experiments presented in this section have been performed on

$$f(x) = \|x\|^2 + \|x\|^z \mathcal{N} \tag{14.13}$$

with $\mathcal{N}$ a Gaussian standard noise. $z = 2$ is the so-called multiplicative noise case, $z = 0$ is the additive noise case, $z = 1$ is intermediate. The results in dimension 2 and dimension 15 are shown in Table 14.1 and 14.2.

We see on these experiments that:

- For $z = 2$ the noise-handling version of Newton's algorithm, $Newton$, performs best among the individual solvers.
- For $z = 1$ the noise-handling version of Newton's algorithm, $Newton$, performs best in dimension 2 and the second variant of Fabian's algorithm, $Fabian2$, performs best in dimension 15.
- For $z = 0$ the first variant of Fabian's algorithm, $Fabian1$, performs best (consistently with [Fabian, 1967]).
- The portfolio algorithm successfully reaches almost the same slope as the best of its solvers and sometimes outperforms all of them.
- Portfolio with *lag* performs better than without *lag*.
- In the case of small noise, NOPA with information sharing, termed NOPA+S., performs better than without information sharing, NOPA, in dimension 15.
- Results clearly show the superiority of INOPA over NOPA.

Incidentally, the poor behavior of RSAES on such a smooth case is not a surprise. Other experiments in Section 14.3.2.2 show that in multimodal cases, RSAES is by far the most efficient solver among solvers above.

14.3.2.2 Experiments in a multimodal setting

Experiments have been performed on a Cartpole control problem with neural network controller. The controller is a feed-forward neural network with one hidden layer of neurons. We use the same solvers as in the previous section. The results are shown in Table 14.3.

We see on these experiments that:

Table 14.1: Experiments on $f(x) = \|x\|^2 + \|x\|^z\mathcal{N}$ in dimension 2 with $z = 0, 1, 2$. Numbers in this table are slopes (Equation 2.2). We see that the portfolio successfully keeps the best of each world (INOPA has nearly the same slope as the best). Results are averaged over 50 runs. "s" as a unit refers to "seconds". *Optimal* means the optimum is reached for at least one run; the number of times the optimum was reached (over the 50 runs) is given between parentheses. The standard deviation is shown after ±. Table 14.2 presents the same results in dimension 15. "NL" refers to "no lag" cases, i.e., $\forall n \in \mathbb{N}^*, \text{LAG}(n) = n$.

Solver/Portfolio	Obtained slope for $d = 2$, $z = 0$				
	$CT = 10s$	$CT = 20s$	$CT = 40s$	$CT = 80s$	$CT = 160s$
RSAES	-.391 ± .009	-.391 ± .009	-.396 ± .010	-.381 ± .012	-.394 ± .012
*Fabian*1	**-1.188 ± .012**	**-1.188 ± .011**	**-1.217 ± .010**	**-1.241 ± .013**	**-1.265 ± .011**
*Fabian*2	−.172 ± .011	−.161 ± .009	−.178 ± .011	−.212 ± .015	−.226 ± .012
Newton	−.206 ± .009	−.206 ± .009	−.212 ± .010	−.237 ± .011	−.239 ± .011
NOPA NL	−.999 ± .047	−.870 ± .061	−.682 ± .064	−.748 ± .066	−.662 ± .067
NOPA+S. NL	−.210 ± .012	−.230 ± .011	−.243 ± .013	−.260 ± .013	−.255 ± .015
NOPA	−.897 ± .054	−.946 ± .049	−.835 ± .059	−.777 ± .064	−.932 ± .058
NOPA+S.	−.264 ± .013	−.298 ± .015	−.268 ± .011	−.304 ± .018	−.303 ± .016
INOPA	−.829 ± .069	−.950 ± .062	−.948 ± .055	−.913 ± .058	−.904 ± .055
INOPA+S.	−.703 ± .058	−.938 ± .056	−.844 ± .055	−.789 ± .055	−.776 ± .055

Solver/Portfolio	Obtained slope for $d = 2$, $z = 1$				
	$CT = 10s$	$CT = 20s$	$CT = 40s$	$CT = 80s$	$CT = 160s$
RSAES	−.526 ± .013	−.530 ± .016	−.507 ± .012	−.507 ± .017	−.522 ± .014
*Fabian*1	−1.247 ± .015	−1.225 ± .009	−1.252 ± .010	−1.276 ± .011	−1.314 ± .013
*Fabian*2	−1.785 ± .009	−1.755 ± .011	−1.782 ± .015	−1.777 ± .011	−1.738 ± .010
Newton	**-2.649 ± .010**	**-2.605 ± .008**	**-2.600 ± .011**	**-2.547 ± .011**	−2.517 ± .010
NOPA NL	−1.624 ± .011	−1.600 ± .011	−1.593 ± .016	−1.554 ± .013	−1.533 ± .014
NOPA+S. NL	−1.225 ± .013	−1.228 ± .013	−1.281 ± .014	−1.298 ± .015	−1.323 ± .017
NOPA	−1.925 ± .081	−1.954 ± .076	−1.661 ± .070	−1.805 ± .066	−1.694 ± .062
NOPA+S.	−1.491 ± .077	−1.624 ± .080	−1.693 ± .072	−1.632 ± .068	−1.537 ± .061
INOPA	−2.271 ± .062	−2.330 ± .061	−2.478 ± .033	−2.506 ± .047	**-2.599 ± .024**
INOPA+S.	−2.013 ± .070	−1.927 ± .074	−1.987 ± .074	−2.120 ± .081	−1.856 ± .078

Solver/Portfolio	Obtained slope for $d = 2$, $z = 2$				
	$CT = 10s$	$CT = 20s$	$CT = 40s$	$CT = 80s$	$CT = 160s$
RSAES	−.500 ± .013	−.491 ± .011	−.484 ± .011	−.526 ± .018	−.537 ± .015
*Fabian*1	−1.233 ± .010	−1.246 ± .013	−1.258 ± .011	−1.299 ± .014	−1.310 ± .013
*Fabian*2	−3.173 ± .010	−3.175 ± .009	−3.141 ± .008	−3.120 ± .013	−3.073 ± .011
Newton	−4.146 ± .004	−4.349 ± .008	−4.514 ± .004	−4.743 ± .012	−4.973 ± .011
NOPA NL	−2.911 ± .009	−2.871 ± .010	−2.796 ± .011	−2.770 ± .012	−2.717 ± .014
NOPA+S. NL	−2.919 ± .011	−2.818 ± .050	−2.785 ± .047	−2.684 ± .056	−2.762 ± .016
NOPA	−4.343 ± .006	−4.603 ± .013	−4.772 ± .013	**Optimal** (1)	−5.103 ± .011
NOPA+S.	−4.305 ± .041	−4.573 ± .011	−4.431 ± .091	−4.910 ± .048	−5.020 ± .059
INOPA	**Optimal** (1)	**Optimal** (2)	−4.698 ± .004	−4.435 ± .007	−4.408 ± 0
INOPA+S.	**Optimal** (1)	−3.302 ± .116	**Optimal** (2)	−4.409 ± .008	**Optimal** (35)

Table 14.2: Experiments on $f(x) = \|x\|^2 + \|x\|^z\mathcal{N}$ in dimension 15 with $z = 0, 1, 2$. Numbers in this table are slopes (Equation 2.2). We see that the portfolio successfully keeps the best of each world (INOPA has nearly the same slope as the best). Results are averaged over 50 runs. "s" as a unit refers to "seconds". *Optimal* means the optimum is reached for at least one run; the number of times the optimum was reached (over the 50 runs) is given between parentheses. The standard deviation is shown after ±. "NL" refers to "no lag" cases, i.e., $\forall n \in \mathbb{N}^*, \text{LAG}(n) = n$.

Solver/Portfolio	Obtained slope for $d = 15$, $z = 0$				
	$CT = 10s$	$CT = 20s$	$CT = 40s$	$CT = 80s$	$CT = 160s$
RSAES	.093 ± .002	.107 ± .002	.114 ± .002	.128 ± .002	.136 ± .003
*Fabian*1	**-.825 ± .003**	**-.826 ± .003**	**-.838 ± .003**	**-.834 ± .004**	**-.835 ± .003**
*Fabian*2	.096 ± .003	.108 ± .003	.108 ± .003	.114 ± .003	.125 ± .003
Newton	−.055 ± .002	−.062 ± .003	−.070 ± .003	−.069 ± .003	−.071 ± .003
NOPA NL	−.512 ± .046	−.393 ± .049	−.377 ± .048	−.425 ± .049	−.380 ± .046
NOPA+S. NL	.026 ± .008	−.026 ± .021	−.082 ± .025	−.237 ± .033	−.410 ± .028
NOPA	−.757 ± .003	−.750 ± .003	−.747 ± .003	−.734 ± .013	−.705 ± .018
NOPA+S.	.039 ± .007	.019 ± .013	.016 ± .019	.005 ± .024	−.079 ± .029
INOPA	−.762 ± .024	−.768 ± .024	−.822 ± .003	−.821 ± .003	−.826 ± .003
INOPA+S.	−.484 ± .033	−.508 ± .035	−.575 ± .038	−.603 ± .036	−.499 ± .037

Solver/Portfolio	Obtained slope for $d = 15$, $z = 1$				
	$CT = 10s$	$CT = 20s$	$CT = 40s$	$CT = 80s$	$CT = 160s$
RSAES	.094 ± .002	.102 ± .002	.118 ± .003	.128 ± .002	.137 ± .003
*Fabian*1	−.991 ± .003	−1.004 ± .003	−1.011 ± .003	−1.020 ± .003	−1.032 ± .003
*Fabian*2	**-1.399 ± .003**	**-1.376 ± .004**	−1.339 ± .003	−1.313 ± .003	−1.274 ± .004
Newton	−.793 ± .099	−.787 ± .095	−.959 ± .092	−.837 ± .086	−.875 ± .078
NOPA NL	−1.226 ± .003	−1.167 ± .012	−.978 ± .013	−.949 ± .008	−.943 ± .005
NOPA+S. NL	−.771 ± .058	−.869 ± .065	−.839 ± .068	−.860 ± .060	−.756 ± .052
NOPA	−.980 ± .018	−.962 ± .013	−.937 ± .005	−.941 ± .005	−.943 ± .004
NOPA+S.	−1.012 ± .020	−1.029 ± .025	−1.019 ± .021	−1.002 ± .014	−.951 ± .010
INOPA	−1.114 ± .016	−1.268 ± .026	−1.359 ± .027	−1.393 ± .018	**-1.482 ± .026**
INOPA+S.	−1.194 ± .030	−1.250 ± .038	**-1.556 ± .030**	**-1.441 ± .041**	−1.399 ± .051

Solver/Portfolio	Obtained slope for $d = 15$, $z = 2$				
	$CT = 10s$	$CT = 20s$	$CT = 40s$	$CT = 80s$	$CT = 160s$
RSAES	.094 ± .003	.102 ± .002	.113 ± .003	.125 ± .003	.146 ± .002
*Fabian*1	−.991 ± .003	−1.000 ± .003	−1.016 ± .003	−1.019 ± .003	−1.037 ± .004
*Fabian*2	−2.595 ± .003	−2.546 ± .003	−2.481 ± .003	−2.413 ± .003	−2.337 ± .004
Newton	−2.911 ± .279	−2.763 ± .291	−2.503 ± .285	−2.420 ± .265	−2.614 ± .240
NOPA NL	−2.257 ± .002	−2.184 ± .003	−2.106 ± .003	−2.000 ± .003	−1.891 ± .003
NOPA+S. NL	−1.220 ± .117	−1.690 ± .134	−2.181 ± .175	−2.131 ± .185	−2.307 ± .157
NOPA	−2.956 ± .121	−2.664 ± .107	−2.515 ± .095	−2.466 ± .090	−2.025 ± .050
NOPA+S.	**-3.996 ± .029**	**-3.796 ± .003**	**-3.567 ± .004**	−3.294 ± .003	−2.947 ± .026
INOPA	−3.005 ± .106	−3.157 ± .123	−3.319 ± .135	**-3.528 ± .144**	**-3.751 ± .136**
INOPA+S.	−3.090 ± .003	−2.942 ± .003	−2.791 ± .004	−2.673 ± .002	−2.574 ± .003

Table 14.3: Slope of simple regret for control of the "Cartpole" problem using a Neural Network policy with different numbers of neurons. This test case is multimodal. These results are averaged over 50 runs. "s" as a unit refers to "seconds". The standard deviation is shown after ± and shows the statistical significance of the results. Values close to 0 correspond to cases with no convergence to the optimum, i.e., a slope zero means that the log-log curve is horizontal. The test case is the one from [Weinstein and Littman, 2012, Couetoux, 2013].

Solver/Portfolio	Obtained slope with 2 neurons				
	$CT = 10s$	$CT = 20s$	$CT = 40s$	$CT = 80s$	$CT = 160s$
RSAES	$-.503 \pm .008$	$-.503 \pm .008$	$-.483 \pm .007$	**-.469 ± .006**	**-.465 ± .003**
*Fabian*1	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$
*Fabian*2	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$
Newton	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$
NOPA NL	$-.442 \pm .014$	$-.469 \pm .010$	$-.465 \pm .006$	$-.452 \pm .005$	$-.433 \pm .006$
NOPA+S. NL	$-.399 \pm .020$	$-.465 \pm .009$	$-.434 \pm .015$	$-.461 \pm .007$	$-.462 \pm .005$
NOPA	$-.480 \pm .014$	$-.465 \pm .009$	$-.466 \pm .008$	$-.430 \pm .013$	$-.431 \pm .009$
NOPA+S.	$-.461 \pm .017$	$-.436 \pm .020$	$-.475 \pm .011$	$-.431 \pm .015$	$-.415 \pm .015$
INOPA	$-.501 \pm .009$	$-.468 \pm .009$	$-.458 \pm .007$	$-.445 \pm .006$	$-.424 \pm .006$
INOPA+S.	**-.524 ± .011**	**-.522 ± .007**	**-.490 ± .006**	**-.469 ± .006**	$-.459 \pm .006$

Solver/Portfolio	Obtained slope with 4 neurons				
	$CT = 10s$	$CT = 20s$	$CT = 40s$	$CT = 80s$	$CT = 160s$
RSAES	**-.517 ± .009**	$-.503 \pm .006$	$-.481 \pm .006$	$-.458 \pm .007$	$-.452 \pm .004$
*Fabian*1	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$
*Fabian*2	$.002 \pm 0$	$-.007 \pm .010$	$-.016 \pm .013$	$-.006 \pm .008$	$-.005 \pm .007$
Newton	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$
NOPA NL	$-.485 \pm .010$	$-.465 \pm .011$	$-.461 \pm .007$	$-.460 \pm .004$	$-.440 \pm .004$
NOPA+S. NL	$-.474 \pm .010$	$-.487 \pm .008$	$-.490 \pm .007$	**-.474 ± .006**	**-.463 ± .004**
NOPA	$-.491 \pm .009$	$-.477 \pm .006$	$-.459 \pm .007$	$-.434 \pm .006$	$-.423 \pm .006$
NOPA+S.	$-.505 \pm .010$	$-.504 \pm .009$	**-.491 ± .007**	$-.470 \pm .006$	$-.452 \pm .006$
INOPA	$-.481 \pm .012$	$-.480 \pm .007$	$-.429 \pm .010$	$-.423 \pm .008$	$-.408 \pm .005$
INOPA+S.	$-.506 \pm .009$	**-.506 ± .008**	$-.479 \pm .007$	$-.466 \pm .006$	$-.439 \pm .005$

Solver/Portfolio	Obtained slope with 6 neurons				
	$CT = 10s$	$CT = 20s$	$CT = 40s$	$CT = 80s$	$CT = 160s$
RSAES	$-.496 \pm .008$	**-.508 ± .008**	$-.479 \pm .007$	$-.462 \pm .004$	$-.439 \pm .005$
*Fabian*1	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$
*Fabian*2	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$
Newton	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$
NOPA NL	$-.483 \pm .008$	$-.477 \pm .009$	$-.464 \pm .005$	$-.447 \pm .005$	$-.432 \pm .005$
NOPA+S. NL	$-.489 \pm .011$	$-.496 \pm .007$	$-.488 \pm .005$	**-.469 ± .005**	$-.464 \pm .005$
NOPA	$-.492 \pm .021$	$-.498 \pm .007$	$-.477 \pm .012$	$-.464 \pm .011$	$-.456 \pm .004$
NOPA+S.	$-.430 \pm .026$	$-.446 \pm .020$	$-.452 \pm .014$	$-.442 \pm .016$	$-.417 \pm .014$
INOPA	$-.501 \pm .010$	$-.485 \pm .010$	$-.487 \pm .006$	$-.463 \pm .006$	$-.447 \pm .003$
INOPA+S.	**-.517 ± .009**	$-.501 \pm .010$	**-.503 ± .006**	$-.468 \pm .005$	**-.467 ± .003**

Solver/Portfolio	Obtained slope with 8 neurons				
	$CT = 10s$	$CT = 20s$	$CT = 40s$	$CT = 80s$	$CT = 160s$
RSAES	$-.493 \pm .009$	$-.475 \pm .006$	$-.449 \pm .006$	$-.436 \pm .007$	$-.420 \pm .005$
*Fabian*1	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$
*Fabian*2	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$-.005 \pm .007$	$.002 \pm 0$
Newton	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$	$.002 \pm 0$
NOPA NL	$-.464 \pm .010$	$-.468 \pm .007$	$-.431 \pm .008$	$-.429 \pm .006$	$-.419 \pm .006$
NOPA+S. NL	$-.463 \pm .008$	$-.480 \pm .009$	$-.485 \pm .008$	$-.466 \pm .006$	$-.453 \pm .005$
NOPA	$-.483 \pm .011$	$-.485 \pm .009$	$-.475 \pm .006$	$-.465 \pm .005$	$-.436 \pm .005$
NOPA+S.	$-.510 \pm .009$	**-.498 ± .008**	**-.508 ± .006**	**-.482 ± .005**	$-.454 \pm .006$
INOPA	$-.488 \pm .010$	$-.463 \pm .009$	$-.455 \pm .007$	$-.422 \pm .007$	$-.426 \pm .006$
INOPA+S.	**-.523 ± .008**	$-.492 \pm .009$	$-.476 \pm .007$	$-.459 \pm .007$	**-.460 ± .004**

Table 14.4: Stochastic Unit Commitment problems, conformant planning. St: number of stocks.

Problem size	Considered NOA or NOPA					
St, T, d	$P.22$	$P.22 + S.$	$P.222$	$P.222 + S.$	Best NOA	Worst NOA
3, 21, 63	0.61 ± 0.07	0.63 ± 0.03	0.63 ± 0.05	0.63 ± 0.07	0.49 ± 0.08	0.81 ± 0.05
4, 21, 84	0.75 ± 0.02	0.75 ± 0.03	0.79 ± 0.05	0.76 ± 0.03	0.69 ± 0.06	1.27 ± 0.06
5, 21, 105	0.53 ± 0.04	0.58 ± 0.08	0.58 ± 0.03	0.52 ± 0.05	0.58 ± 0.04	1.44 ± 0.16
6, 15, 90	0.40 ± 0.05	0.39 ± 0.06	0.37 ± 0.06	0.39 ± 0.06	0.38 ± 0.06	0.96 ± 0.13
6, 21, 126	0.53 ± 0.08	0.54 ± 0.08	0.55 ± 0.07	0.54 ± 0.07	0.54 ± 0.07	1.78 ± 0.37
8, 15, 120	0.53 ± 0.03	0.50 ± 0.05	0.53 ± 0.02	0.51 ± 0.05	0.51 ± 0.04	1.70 ± 0.10
8, 21, 168	0.69 ± 0.04	0.77 ± 0.09	0.73 ± 0.06	0.71 ± 0.04	0.71 ± 0.06	2.68 ± 0.02
7, 21, 147	0.70 ± 0.07	0.70 ± 0.05	0.70 ± 0.07	0.70 ± 0.07	0.69 ± 0;06	2.28 ± 0.08

- *RSAES* is the most efficient individual solver.
- The portfolio algorithm successfully reaches almost the same slope as the best of its solvers.
- Sometimes, the portfolio outperforms the best of its solvers.
- Results clearly show the superiority of INOPA over NOPA.

14.3.2.3 Experiments on stochastic unit commitment problems

Experiments have been performed on stochastic unit commitment problems described in Section 7.4.2. The controller is a feed-forward neural network with one hidden layer of neurons. We use the same solvers as in previous section. The results are shown in Table 14.4.

We see on these experiments that:

- Given a same budget, a NOPA of identical solvers can outperform its NOAs.
- *RSAES* is usually the best NOA for small dimensions and variants of Fabian for large dimension.

14.3.3 The *lag*: experiments with different variants of Fabian's algorithm

In this section, we check if the version with *lag* disabled (i.e., $\forall n \in \mathbb{N}^*, \text{LAG}(n) = n$) can compete with the version with *lag* enabled (i.e., $\forall n \in \mathbb{N}^*, \text{LAG}(n) < n$). In previous experiments this was the case, we

here focus on a case in which solvers are close to each others and check if in such a case the *lag* is beneficial.

Fabian's algorithm [Fabian, 1967] is a gradient descent algorithm using finite differences for approximating gradients. $\frac{a}{n}$ is the step in updates, i.e., the current estimate is updated by adding $-\frac{a}{n}\nabla f$ where ∇f is the approximate gradient. ∇f is approximated by averaging multiple redundant estimates, each of them by finite differences of size $\Theta(c/n^\gamma)$. Therefore, Fabian's algorithm has 3 parameters, termed a, c and γ. In the case of approximately quadratic functions with additive noise, Fabian's algorithm can obtain a good $s(SR)$ with small $\gamma > 0$. However, a and c have an important non-asymptotic effect and the tuning of these a and c parameters is challenging. A portfolio of variants of Fabian's algorithm can help to overcome the tedious parameter tuning.

For these experiments, we consider the same noisy function as in Section 14.3.2.1. We use 5 noisy optimization solvers which are variants of Fabian's algorithm, as detailed in Algorithm 21, and portfolio of these solvers with and without *lag*:

- Solver 1: $Fabian1$ as used in Section 14.3.2.
- Solver 2: Fabian's solver with parametrization $\gamma = 0.1$, $a = 5$, $c = 100$.
- Solver 3: Fabian's solver with parametrization $\gamma = 0.1$, $a = 1$, $c = 200$.
- Solver 4: Fabian's solver with parametrization $\gamma = 0.1$, $a = 1$, $c = 1$.
- Solver 5: Fabian's solver with parametrization $\gamma = 0.1$, $a = 1$, $c = 10$.
- NOPA NL: Portfolio of solvers 1-5 *without lag*. Functions are $r_n = \lceil n^{4.2} \rceil$, $\text{LAG}(n) = n$, $s_n = \lceil n^{2.2} \rceil$ at n^{th} algorithm selection.
- NOPA: NOPA of solvers 1-5. Functions are $r_n = \lceil n^{4.2} \rceil$, $\text{LAG}(n) = \lceil n^{1/4.2} \rceil$, $s_n = \lceil n^{2.2} \rceil$ at n^{th} algorithm selection.
- NOPA+S.: NOPA of solvers 1-5, with information sharing enabled. Same functions.
- INOPA: INOPA of solvers 1-5. Same functions.

- INOPA+S.: INOPA of solvers 1-5, with information sharing enabled. Same functions.

Experiments have been performed in dimension 2 and dimension 15. These 5 variants of Fabian's algorithm have asymptotically similar performance. Table 14.5 compares the portfolio above without *lag*, NOPA and INOPA.

We see on these experiments that:

- The *lag* is usually beneficial, though this is not always the case.
- Again, INOPA clearly outperforms NOPA.

14.3.4 Discussion of experimental results

In short, experiments

- validate the use of portfolio (almost as good as the best solver, and sometimes better thanks to its inherent mitigation of "bad luck runs"); we incidentally provide, with INOPA applied to several independent copies of a same solver, a principled tool for restarts for noisy optimization;
- validate the improvement provided by unfair budget, as shown by the consistent improvement of INOPA vs NOPA;
- are less conclusive in terms of comparison "with *lag* / without *lag*", though on average *lag* is seemingly beneficial.

14.4 Conclusion: INOPA is a mathematically proved and empirically good portfolio method for black-box noisy optimization, and our simple sharing tests did not provide clear improvements

We have seen that noisy optimization provides a very natural framework for portfolio methods. Different noisy optimization algorithms have extremely different convergence rates (different slopes) on different test cases, depending on the noise level, on the multimodalities, on the dimension (see e.g. Tables 14.1 and 14.5, where depending on z

14.4. Conclusion: INOPA is a mathematically proved and empirically good portfolio method for black-box noisy optimization, and our simple sharing tests did not provide clear improvements

Table 14.5: Experiments on $f(x) = \|x\|^2 + \|x\|^z\mathcal{N}$ in dimension 2 and dimension 15 with $z = 0, 1, 2$. Results are mean of 1000 runs. Solvers are various parametrizations of Fabian's algorithm (see text). "s" as a unit refers to "seconds". The standard deviation is shown after $\pm$ and shows the statistical significance of the results. We use smaller time settings - this is because here all algorithms have the same asymptotic rate and the objective function is a simple super fast objective function; therefore we use small computation times so that we see what happens before reaching the asymptotic regime.

Portfolio	obtained slope for $d = 2$				
	$CT = 0.05s$	$CT = 0.1s$	$CT = 0.2s$	$CT = 0.4s$	$CT = 0.8s$
			$z = 0$		
NOPA NL	$-1.157 \pm .009$	$-1.223 \pm .010$	$-1.146 \pm .009$	$-1.109 \pm .009$	$-1.043 \pm .009$
NOPA+S. NL	$-1.030 \pm .009$	$-1.002 \pm .011$	$-.807 \pm .010$	$-.839 \pm .009$	$-.774 \pm .007$
NOPA	$-1.255 \pm .009$	$-1.203 \pm .009$	$-1.156 \pm .008$	$-1.145 \pm .007$	$-1.152 \pm .007$
NOPA+S.	$-1.030 \pm .009$	$-1.044 \pm .008$	$-.995 \pm .009$	$-.963 \pm .008$	$-.926 \pm .007$
INOPA	**-1.289 ± .010**	$-1.247 \pm .008$	**-1.201 ± .008**	**-1.188 ± 0.008**	**-1.153 ± 0.007**
INOPA+S.	$-1.246 \pm .010$	**-1.266 ± .008**	$-1.182 \pm .010$	-1.135 ± 0.010	-1.113 ± 0.009
			$z = 1$		
NOPA NL	$-1.529 \pm .005$	$-1.471 \pm .005$	$-1.455 \pm .004$	$-1.414 \pm .004$	$-1.381 \pm .004$
NOPA+S. NL	$-1.436 \pm .006$	$-1.401 \pm .006$	$-1.287 \pm .006$	$-1.225 \pm .005$	$-1.129 \pm .005$
NOPA	$-1.543 \pm .005$	$-1.490 \pm .004$	$-1.456 \pm .004$	$-1.416 \pm .003$	$-1.377 \pm .003$
NOPA+S.	$-1.469 \pm .005$	$-1.462 \pm .005$	$-1.377 \pm .005$	$-1.339 \pm .005$	$-1.301 \pm .004$
INOPA	**-1.656 ± .004**	**-1.578 ± .005**	**-1.531 ± .004**	**-1.497 ± .005**	**-1.443 ± .004**
INOPA+S.	$-1.638 \pm .005$	$-1.568 \pm .005$	$-1.503 \pm .005$	$-1.474 \pm .005$	$-1.425 \pm .005$
			$z = 2$		
NOPA NL	$-1.528 \pm .004$	$-1.505 \pm .005$	$-1.440 \pm .004$	$-1.394 \pm .004$	$-1.375 \pm .003$
NOPA+S. NL	$-1.456 \pm .005$	$-1.393 \pm .006$	$-1.303 \pm .005$	$-1.250 \pm .005$	$-1.168 \pm .005$
NOPA	$-1.540 \pm .005$	$-1.529 \pm .004$	$-1.439 \pm .004$	$-1.422 \pm .004$	$-1.384 \pm .003$
NOPA+S.	$-1.473 \pm .006$	$-1.450 \pm .005$	$-1.371 \pm .004$	$-1.339 \pm .004$	$-1.303 \pm .004$
INOPA	**-1.681 ± .005**	**-1.607 ± .004**	**-1.530 ± .005**	**-1.497 ± .004**	**-1.439 ± .005**
INOPA+S.	$-1.578 \pm .006$	$-1.570 \pm .006$	$-1.517 \pm .005$	$-1.465 \pm .005$	-1.434 ± 0.005

Portfolio	obtained slope for $d = 15$				
	$CT = 0.05s$	$CT = 0.1s$	$CT = 0.2s$	$CT = 0.4s$	$CT = 0.8s$
			$z = 0$		
NOPA NL	$-.673 \pm .001$	$-.688 \pm .001$	$-.699 \pm .001$	$-.761 \pm .002$	$-.779 \pm .002$
NOPA+S. NL	$-.664 \pm .001$	$-.684 \pm .001$	$-.703 \pm .001$	$-.716 \pm .001$	$-.750 \pm .001$
NOPA	$-.700 \pm .006$	$-.609 \pm .006$	$-.667 \pm .004$	$-.681 \pm .005$	$-.694 \pm .004$
NOPA+S.	$-.591 \pm .006$	$-.515 \pm .006$	$-.514 \pm .005$	$-.519 \pm .004$	$-.527 \pm .004$
INOPA	**-.839 ± .001**	**-.839 ± .001**	**-.841 ± .001**	**-.840 ± .001**	$-.839 \pm .001$
INOPA+S.	**-.839 ± .001**	**-.839 ± .001**	**-.841 ± .001**	$-.839 \pm .001$	**-.841 ± .001**
			$z = 1$		
NOPA NL	$-1.004 \pm .001$	$-.991 \pm .001$	$-.980 \pm .001$	$-.978 \pm .001$	$-1.062 \pm .001$
NOPA+S. NL	$-1.000 \pm .001$	$-.985 \pm .001$	$-.980 \pm .001$	$-.990 \pm .001$	$-1.066 \pm .001$
NOPA	$-1.154 \pm .001$	$-1.140 \pm .001$	$-1.117 \pm .001$	$-1.100 \pm .001$	$-1.086 \pm .001$
NOPA+S.	$-1.160 \pm .001$	$-1.133 \pm .001$	$-1.109 \pm .001$	$-1.084 \pm .001$	$-1.065 \pm .001$
INOPA	$-1.231 \pm .001$	**-1.249 ± .001**	**-1.238 ± .001**	**-1.218 ± .001**	**-1.200 ± .001**
INOPA+S.	**-1.242 ± .001**	$-1.198 \pm .003$	$-1.169 \pm .003$	$-1.151 \pm .002$	$-1.131 \pm .003$
			$z = 2$		
NOPA NL	$-.999 \pm .001$	$-.995 \pm .001$	$-.981 \pm .001$	$-.980 \pm .001$	$-1.065 \pm .001$
NOPA+S. NL	$-.999 \pm .001$	$-.979 \pm .001$	$-.973 \pm .001$	$-.987 \pm .001$	$-1.064 \pm .001$
NOPA	$-1.174 \pm .001$	$-1.135 \pm .001$	$-1.119 \pm .001$	$-1.101 \pm .001$	$-1.083 \pm .001$
NOPA+S.	$-1.152 \pm .002$	$-1.130 \pm .001$	$-1.103 \pm .001$	$-1.080 \pm .001$	$-1.061 \pm .001$
INOPA	**-1.234 ± .001**	**-1.251 ± .001**	**-1.237 ± .001**	**-1.219 ± .001**	**-1.197 ± .001**
INOPA+S.	$-1.085 \pm .001$	$-1.085 \pm .001$	$-1.083 \pm .001$	$-1.083 \pm .001$	$-1.085 \pm .001$

the best solver is a variant of Fabian or Newton's algorithm; and Table 14.3, where RSAES is the best). We proposed two versions of such portfolios, NOPA and INOPA, the latter using an unfair distribution of the budget. Both have theoretically the same slope as the best of their solvers, with better constants for INOPA (in particular, no shift, if *SubSetOptim* (see Equation 14.5) has cardinal 1).

We show mathematically a $\log(M)$ shift when using M solvers, when working on a classical log-log scale (classical in noisy optimization); see Section 14.2.4.2. 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 (Section 14.2.4.4), for getting the $\log(M)$ shift, that:

- the AS algorithm compares *old* recommendations (and selects a solver from this point of view),
- the portfolio recommends the *current* recommendation of this selected solver.

Additionally, we improve the bound to a $\log(M')$ shift, where M' is the number of optimal solvers, using an unfair distribution of the computational budget (Section 14.2.4.3). In particular, the shift is negligible when the optimal solver is unique.

A careful choice of portfolio parameters (function $\textsc{lag}(\cdot)$, specifying the *lag*; r_n, specifying the intervals $r_{n+1} - r_n$ between two comparisons of solvers; s_n, specifying the number of resamplings of recommendations for selecting the best) leads to such properties; we provide principled tools for choosing these parameters. Sufficient conditions are given in Theorem 5, with examples thereafter.

Experiments show (i) the efficiency of portfolios for noisy optimization, as solvers have very different performances for different test cases and NOPA has performance close to the best or even better when the random initialization has a big impact (ii) the clear and stable improvement provided by INOPA, thanks to an unfair budget distribution (iii) that the *lag* is usually beneficial, though this is not always the case. Importantly, without lag, INOPA could not be defined.

In noisy frameworks, we point out that portfolios might make sense even when optimizers are not orthogonal. Even with identical solvers, or closely related optimizers, the portfolio can mitigate the effect of unlucky random contributions. This is somehow related to restarts. See Table 14.5 for cases with very close solvers, and [Liu and Teytaud, 2014] with identical solvers.

14.4. Conclusion: INOPA is a mathematically proved and empirically good portfolio method for black-box noisy optimization, and our simple sharing tests did not provide clear improvements

Sharing information in portfolios of noisy optimization algorithms is not so easy. Our empirical results are mitigated.

A limitation of the present work is the lack of experiments on stochastic unit commitment problems using INOPA.

15 Differential evolution as a non-stationary portfolio method

In this chapter, we are interested in comparing an evolutionary approach, namely Differential Evolution (DE) [Storn and Price, 1997], to the classical bandit approach. DE is arguably one of the main evolutionary algorithms. It does not require heavy computations, performs well on many optimization testbeds [Ardia et al., 2011b, Ardia et al., 2011a, Poaík and Klema, 2012] and possesses many variants including self-adaptive parameters [Brest et al., 2006, Liu and Lampinen, 2005, Coelho and Mariani, 2006, Price et al., 2005]. Our contribution is a variation of DE that outperforms UCB-Discounted [Kocsis and Szepesvari, 2006b] and UCB-Discounted2 (see Section 15.2.1.2), both variants of UCB [Auer et al., 2002a] for the non-stationary testcase.

15.1 Introduction

We are interested in a situation where an agent faces K possible options of unknown distribution(s) and is given T sequential evaluations, where T is not necessarily known. The agent is then asked to output $\pi_{\tilde{REC}}$, a recommendation that corresponds to a probability distribution over K options, according to a performance criterion. In our case the performance criterion is the Simple Regret (SR) and we seek to maximize a reward.

This class of problems is well formalized through the bandit framework [Robbins, 1952] and more specifically our setting is similar to [Bubeck et al., 2009]. Typically this framework tackles effectively the tradeoff between exploration and exploitation.

During the evaluation process $t \in T$, the agent selects an option $k \in K$ according to its selection policy $\pi_{\tilde{SEL}}(\cdot)$. A selection policy $\pi_{\tilde{SEL}}(\cdot) \in$

K is an algorithm that selects an option based on the information at hand. A representative example of a selection policy is UCB [Auer et al., 2002a]. At the t^{th} evaluation, the option k is selected and a reward $r_{k,t}$ is computed. A detailed description of the selection policies and the distribution updates used in this section are provided in Section 15.2.

Such a process is repeated until the allocated number of evaluations T has been executed. Afterward, following the recommendation $\pi_{\tilde{REC}}$, an option $\tilde{k}$ is chosen. The pseudo code for a generic bandit algorithm up to the recommendation of $\tilde{k}$ is provided in Algorithm 18 and the recommendation of $\tilde{k}$ used in this section is provided in Section 15.2.1. In this chapter we focus on non-stationary distributions.

Algorithm 18 Generic Bandit Algorithm. The problem is described through the "get stochastic reward", a stochastic method and the option sets. The "return" method is formally called the recommendation policy. The selection policy is also commonly termed exploration policy.

Input: $T > 0$: Computational budget
Input: K: Set of options
Input: $\pi_{\tilde{SEL}}$: Selection policy
  for $t = 1$ to T **do**
    Select $k \in K$ based upon $\pi_{\tilde{SEL}}(\cdot)$
    Get stochastic reward $r_{k,t}$
    Update the information of k with $r_{k,t}$
  end for
  Return $\pi_{\tilde{REC}}$: Probability distribution over the set K

A non-stationary bandit problem implies that the distribution of each option $k \in K$ changes. In such cases, most of the literature is focused over a variation of distributions based upon the number of evaluations t [Kocsis and Szepesvari, 2006b, Audibert et al., 2007, Koulouriotis and Xanthopoulos, 2008]. In this chapter, we focus on a variation based upon the number of times we select an option. The reward distribution for an option improves as we select it more often. Such framework possesses a wide range of applications. The selection of an object from a portfolio is a good example and covers several testcases that justify such a definition.

In this chapter we are interested in comparing an evolutionary approach, namely Differential Evolution (DE) [Storn and Price, 1997], to the classical bandit approach. DE is arguably one of the main evolutionary algorithms. It does not require heavy computations, performs well on many optimization testbeds [Ardia et al., 2011b, Ardia et al., 2011a, Poaík and Klema, 2012] and possesses many variants including self-adaptive parameters [Brest et al., 2006, Liu and Lampinen, 2005, Coelho and Mariani, 2006, Price et al., 2005]. Our contribution is a variation of DE that outperforms UCB-Discounted [Kocsis and Szepesvari, 2006b] and UCB-Discounted2 (see Section 15.2.1.2), both variants of UCB [Auer et al., 2002a] for the non-stationary testcase.

Section 15.2 formulates the problem and introduces classic selection policies. Section 15.2.2 presents the DE algorithm and our variant. Section 15.2.4 shows the results and Section 14.4 concludes.

15.2 Problem Statement

In this section we formalize the bandit problem and introduce its related selection policy. Section 15.2.1 defines the non-stationary problem and presents two variants of UCB, one of the most popular selection policy for stationary bandit, UCB-Discounted, termed $\pi_{\tilde{SELUCBdt}}$, and our modified UCB-Discounted, termed $\pi_{\tilde{SELUCBdn}}$. These variants of UCB are specific for the non-stationary bandit problem. The first one is designed for problems where the reward distribution changes over time. The second one is a modification that we propose and where the distribution changes over the number of times an option is selected.

15.2.1 Non-Stationary Bandit Problem

In the non-stationary case under study, the reward distribution of the k^{th} option is given by $\mu_{k,n_{k,t}}$ and represents the expectation of rewards $r_{k,t}$, where $n_{k,t}$ is the number of times an option k has been selected after the t^{th} evaluation. $r_{k,t}$ is the reward of option k obtained at the t^{th} evaluation and given by $\sim Bernoulli(1, \mu_{k,n_{k,t}})$, with 1 being the number of Bernoulli trials.

At each time $t \in T$ the agent chooses an option $k \in K$ following a policy $\pi_{\tilde{SEL}}$ and obtains a reward $r_{k,t}$. The reward is modeled from distributions unknown to the user. As such, the best option after t evaluations is $k_t^* = \underset{k \in K}{\operatorname{argmax}}\, \mu_{k,t}$.

The recommendation policy $\pi_{\tilde{REC}}$ usually boils down to selecting the most played option

$$\tilde{k}_t = \underset{k \in K}{\operatorname{argmax}}\, n_{k,t},$$

yet there exist several other recommendation policies [Chou et al., 2012].

We use Simple Regret as optimization criterion. Simple Regret is the difference between the average reward of the best option and the average reward obtained by the recommendation. As we focus on the non-stationary bandit problem, the Simple Regret is thus computed by $\mu_{k^*_T,T} - \mu_{\tilde{k}_t,n_{k,t}}$ for $t \in T$.

15.2.1.1 UCB-Discounted

UCBdt [Kocsis and Szepesvari, 2006b] is an attempt to adapt UCB to the non-stationary bandit problem. The idea is to partially forget past information. As such, the estimated value computed from rewards is changed from:

$$\bar{r}_{k,t} = \frac{\sum_{i=1}^{t} r_{k_i} : k_i = k}{n_{k,t}}, \tag{15.1}$$

to:

$$\bar{r}_{k,t} = \frac{\sum_{i=1}^{t} (\gamma^{t-i} \cdot r_{k_i}) : k_i = k}{\sum_{i=1}^{t} \gamma^{t-i}}$$

where $\gamma \in (0, 1]$ and r_{k_t} is the reward of the selected option k_t obtained at the t^{th} evaluation. Note that if $\gamma = 1$ it becomes a UCB as defined for stationary bandit problems. The $\pi_{\tilde{SEL\,UCBdt}}$ is given by:

$$\underset{k \in K}{\operatorname{argmax}} \left(\bar{r}_{k,t} + \sqrt{C \frac{\ln t(\gamma)}{n_k(\gamma)}} \right) ,$$

where $C > 0$, $n_k(\gamma) = \sum_{i=1}^{t} \gamma^{t-i} : k_i = k$ and $t(\gamma) = \sum_{k \in K} n_k(\gamma)$. Other padding functions are considered in [Audibert et al., 2007, Garivier and Moulines, 2008]. The main issue in this definition is that the discount depends on t rather than $n_{k,t}$.

15.2.1.2 UCB-Discounted2

We propose a second version of discounted UCB named UCBdn. UCBdn is similar to UCBdt but the discount depends on $n_{k,t}$, the number of time an option is selected. The idea is still to partially forget past information. The update rule becomes:

$$\bar{r}_{k,t} = \frac{\sum_{i=1}^{n_{k,t}} (\gamma^{n_{k,t}-i} \cdot r_{k_i})}{\sum_{i=1}^{n_{k,t}} \gamma^{n_{k,t}-i}}$$

where $\gamma \in (0, 1]$. $\pi_{\tilde{SEL}\,UCBdn}$ is given by:

$$\underset{k \in K}{\operatorname{argmax}} \left(\bar{r}_{k,t} + \sqrt{C \frac{\ln t(\gamma)}{n_k(\gamma)}} \right) ,$$

where $C > 0$, $n_k(\gamma) = \sum_{i=1}^{n_{n,t}} \gamma^{n_{k,t}-i}$ and $t(\gamma) = \sum_{k \in K} n_k(\gamma)$.

15.2.2 Differential Evolution Algorithm

To represent a bandit problem into a framework that can be optimized through an evolutionary approach is far from trivial and is the subject of Section 15.2.2.1. Section 15.2.2.2 shows the protocol we use for the evaluation of the fitness and Section 15.2.3 presents a small variation proposed to greatly improve the performance of DE for bandit problem.

Let $f : \mathbb{R}^K \to \mathbb{R}$ be the fitness function to be maximized where K is the dimension of the problem. The DE algorithm takes as input a population $\mathcal{X}$ of individuals where each individual $x \in \mathbb{R}^K$ are candidate solutions. The goal can be viewed as finding a solution s such that $\forall\, x \in \mathbb{R}^K\ f(s) \geq f(x)$. Indeed this is for a maximization problem and the solution found is a global maximum. For a minimization, we can easily consider a fitness $h : -f$ instead. The variable t is increased every time a function evaluation is performed. The parameters $F \in [0, 2]$, $Cr \in [0, 1]$ and $|\mathcal{X}|$, the cardinality of $\mathcal{X}$, have a large impact on the performance of the optimization. Thus, the tuning process is an important part of the algorithm. Algorithm 19 presents our version of the algorithm.

15.2.2.1 Representation

In our DE algorithm, each individual $x \in \mathcal{X}$ represents a probability distribution over each option $k \in K$. Obviously, $\sum_{k \in K} x_k = 1$. These individuals are initialized with random probabilities as shown in Algorithm 19. Moreover, after each $t \in T$ evaluations we keep in memory $\bar{r}$, a vector of $[0,1]^K$, representing the different $\bar{r}_{k,t}$ as defined in (15.1). In this chapter, we add $\bar{r}$ as an individual in $\mathcal{X}$ and is updated at the end of each generation. In the following Section we further describe the evaluation of the fitness.

15.2.2.2 Evaluation of the fitness

When evaluating an individual x, we (i) optionally truncate (see Section 15.2.3) and renormalize, (ii) randomly draw an option $k \in K$ (k is selected with probability x_k) and (iii) evaluate the selected option by getting a reward $r_{k,t}$ through t' Bernoulli trial(s), where $t' > 0$ is the number of re-evaluations.

For the evaluation of the fitness $f(x)$, we decide to compute it following

$$\sum_{k \in K} (x_k \cdot \bar{r}_{k,t}).$$

Note that x_k and $\bar{r}_{k,t}$ are two values accessible without any call to the problem at hand, there is no Bernoulli trial executed during the fitness evaluation. As such, it does not increment the evaluation budget $t \in T$.

15.2.3 Truncation

In order to make DE more competitive, we propose a modification similar in its essence to Bernstein pruning [Mnih et al., 2008]. Since the recommendation $\pi_{\tilde{REC}}$ consists in choosing only one option $\tilde{k}$ and we have access to the current approximation of $\mu_{k,n_{k,t}}$ through $\bar{r}_{k,t}$, we can truncate the options that are not likely to be recommended. This ensure that DE is solely focusing on the best options. The rule we used in the algorithm is given by:

$$x_k = \begin{cases} x_k & \text{if } \bar{r}_{k,t} > c_1 \cdot \max_{k \in K} \bar{r}_{k,t} \text{ and } n_k > c_2 \cdot \max_{k \in K} n_{k,t} \\ 0 & \text{otherwise,} \end{cases}$$

where c_1, $c_2 \in (0, 1]$. The truncation process is executed during the normalization process when the current number of evaluations is bigger than a given $M \geq 0$, as described in Algorithm 19.

15.2.4 Experiments

In this section we describe the various experiments. Section 15.2.4.1 defines the non-stationary problem at hand, Section 15.2.4.2 presents the tuning of the UCB variants, Section 15.2.4.3 shows the tuning of the DE algorithm and finally Section 15.2.4.4 compares the two algorithms.

15.2.4.1 Non-stationary data

We use a portfolio of solvers over the benchmark Cart-Pole problem [Weinstein and Littman, 2012] as a testbed. Each option of the bandit problem represents a black-box Direct Policy Search (DPS) [Sutton and Barto, 1998]. More specifically for our case, we focus on three variants of black-box Noisy Optimization Algorithms (NOAs) combined with Neural Network (NN) (1, 2, 4, 6, 8 and 16 neurons). The three black-box NOAs are:

- a Self-Adaptive (μ, λ) Evolution Strategy with resamplings (RSAES) [Astete-Morales et al., 2013];
- a Fabian's stochastic gradient algorithm with finite differences [Fabian, 1967, Chen, 1988, Shamir, 2013];
- and a variant of Newton algorithm, adapted for noisy optimization, with gradient and Hessian approximated by finite differences and resamplings [Astest Morales et al.,], termed Noisy Newton.

Basically at iteration n, the NOA chooses one (or more) new point(s) z_n in the search domain and gets a reward $r_{k,t}$. RSAES and Noisy Newton use resamplings, i.e. re-evaluations, to reduce the noise. Note that the budget is consumed accordingly with the number of resamplings. Table 15.1 describes the specifics of the NOAs that we use in our experiments. λ_I represents the number of points selected (more generally termed individuals size) during one iteration, r_n is the number of resamplings of each individual and σ_n is the stepsize. We refer to [Cauwet et al., 2014] for more details on the settings and the different algorithms. As shown in Table 15.1, 24 solvers are used to resolve the Cart-Pole problem, thus resulting in 24 options in the non-stationary

bandit problem. Experiments are repeated 50 times and we present the mean of these results.

Table 15.1: Solvers used in the experiments, where d is dimension depending on the number of neurons; r_n is resampling number at iteration n for each individual. We refer to Chapter 14 for more details.

Number of neurons	Algorithm and parametrization
1, 2, 4, 6, 8, 16	RSAES with individual size $\lambda_I = 10d$, $\mu = 5d$, $r_n = 10n^2$.
1, 2, 4, 6, 8, 16	Fabian's solver with individual size $\lambda_I = 4$, stepsize $\sigma_n = 10/n^{0.1}$, $a = 100$, no resamplings.
1, 2, 4, 6, 8, 16	Noisy Newton's solver with individual size $\lambda_I = 4d^2 + 2d + 1$, stepsize $\sigma_n = 10/n$, $r_n = n^2$.
1, 2, 4, 6, 8, 16	Noisy Newton's solver with individual size $\lambda_I = 4d^2 + 2d + 1$, stepsize $\sigma_n = 100/n^4$, $r_n = n^2$.

15.2.4.2 Tuning of bandit

In this section we present the tuning of the different UCBs under study. For the baseline UCB, the only parameter to tune is the C constant, which control the tradeoff between exploration and exploitation. Figure 15.1 presents a summary of the best combinations of parameters. The x-axis represents the number of evaluations and the y-axis shows the Simple Regret. For each point in Figure 15.1, 50 independent trials are executed and we present the mean of these results.

UCB does not possess the inherent quality to tackle non-stationary bandit problem. Instead, to artificially induce the ability to keep adapting to the variation in the distributions we expect higher exploration rate C. This is exactly what we observe in Figure 15.1 where the best C constant is 200 when the number of evaluations is below 7×10^4 and diminishes to 20 as the number of evaluations increases. This value is much higher than on stationary problem where typical empirical values are usually within $[0, 2]$.

For UCBdn and UCBdt, we look for the best combination of parameters C and γ. Figure 15.2 presents different C values for the best $\gamma = 0.85$ found for UCBdn (Top) and the best $\gamma = 0.99$ found for UCBdt (Bottom). The rest of the setting is similar to Figure 15.1.

The best combination of parameters for UCBdn seems to be in the vicinity of $C = 2$ and $\gamma = 0.85$ for the given problem. The current γ value indicates that a fair amount of information is lost at each iteration but it is combined with a relatively small exploration factor $C = 2$. As opposed to UCBdn, UCBdt possesses a very high value for γ and C. Its best γ seems to indicate that UCBdt does not rely on forgetting

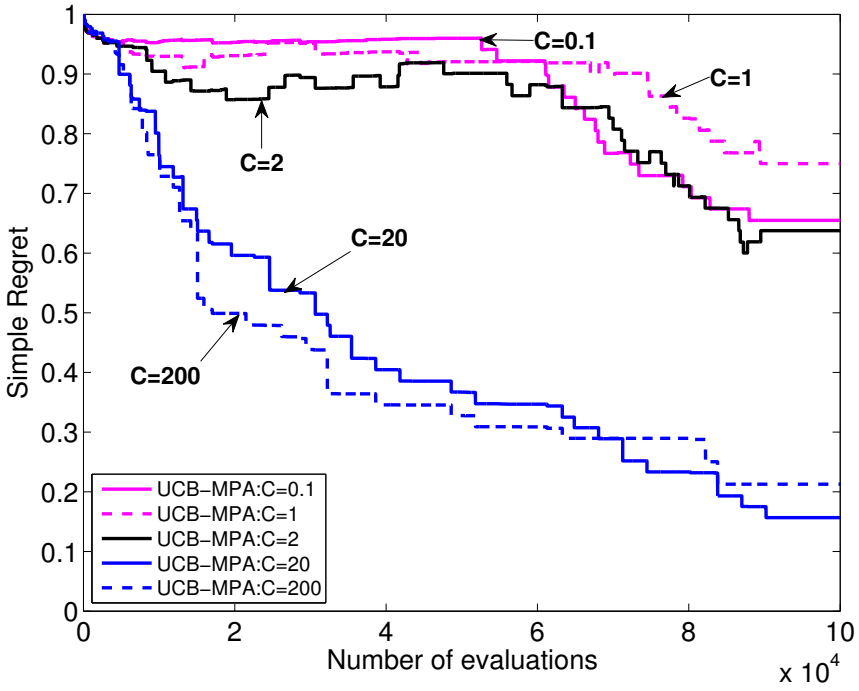
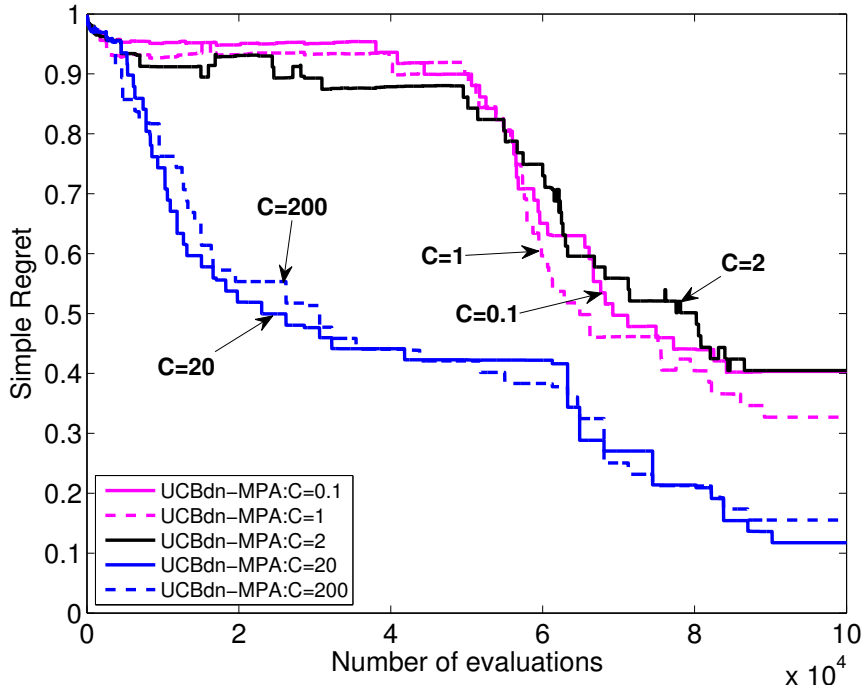


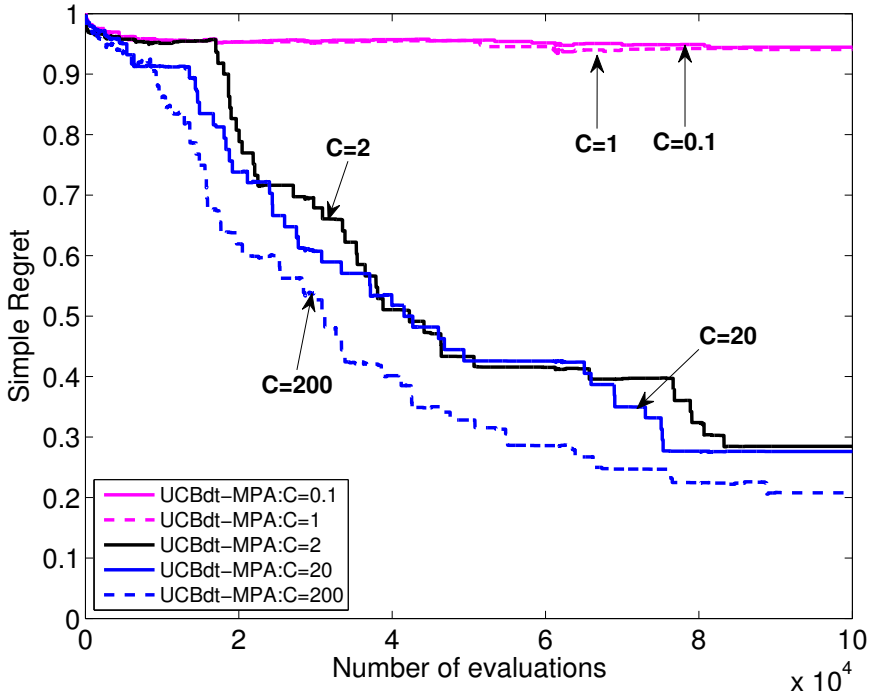
Figure 15.1: Simple Regret for UCB. Tested parameters are $C \in [10^{-3}, 200]$. For the clarity of graph, we remove some similar results and only show results for $C = 0.1$, $C = 1$, $C = 2$, $c = 20$ and $C = 200$. UCBdn converges faster and depends less on C than UCB or UCBdt. The recommendation policy $\pi_{\tilde{REC}}$ is Most Played Arm (MPA), where arm refer to option in the present chapter.

information throughout the iterations. As such, it is conceivable that it relies on a higher exploration factor C. It appears that because UCBdt has a forget rate that is not related with the way the distributions evolve its best combination of parameters boils down to almost a normal UCB.

From Figure 15.2, it clearly appears that UCBdn is the best variant of UCB for the settings describe in this chapter.



(a) UCB discounted2 (UCBdn) with $\gamma = 0.85$.



(b) UCB discounted (UCBdt) with $\gamma = 0.99$.

Figure 15.2: Top: UCBdn performance with $\gamma = 0.85$; Bottom: UCBdt performance with $\gamma = 0.99$. Several values of γ and C are tested for UCBdn and UCBdt, we remove some results for the clarity of graphs. Only the best parameterization of γ are presented. UCBdt relies on a higher exploration factor C. UCBdn depends less on C and converges faster than UCBdt using the same constant C.

15.2.4.3 Tuning of DE

In this section we tune different variants of DE and compare them. There are 3 parameters to tune for DE and its variants: λ, F and Cr. Different settings of $F \in [0, 2]$ and $Cr \in [0, 1]$ are presented in combination with the 2 best population sizes that are found which are $\lambda = 5$ and $\lambda = 10$.

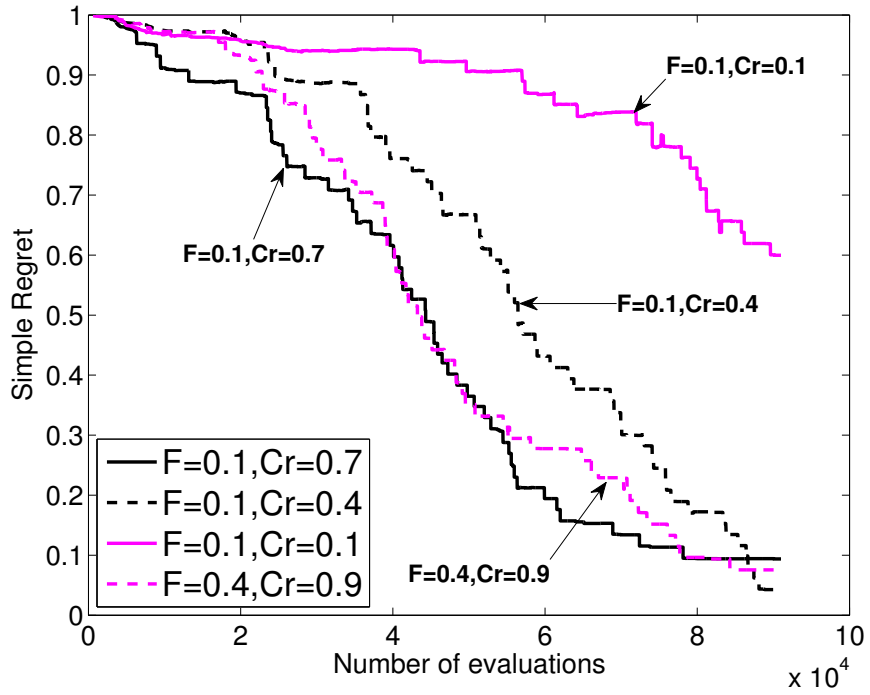
Figure 15.3 shows average performance of DE in terms of Simple Regret. The x-axis represents the number of evaluations and the y-axis shows the Simple Regret. For each point in Figure 15.3, 50 independent trials are executed and we present the mean of these results.

First, it seems that a smaller population yields better performance. It is understandable since each particle consumes part of the evaluation budget and does not directly share information between them. As such, there is a loss in the efficiency of sampling which increases as the size of the population grows. The best parameter setting for $\lambda = 10$ (Bottom) is $F = 0.1$ and $Cr = 0.7$, that is a relatively small differential weight F that prevents the distribution to move too fast in a direction but with a relatively high crossover probability Cr. For $\lambda = 5$ (Top), the combination $F = 0.1$ and $Cr = 0.7$ still yields good results but there exists several other combinations that give similar results. After 9×10^4 evaluations, none of the combination of F and Cr for a $\lambda = 10$ reaches a simple regret lower than 0.5 whereas for the same number of evaluations, the combination $\lambda = 5$, $F = 0.1$ and $Cr = 0.7$ reaches 0.23.

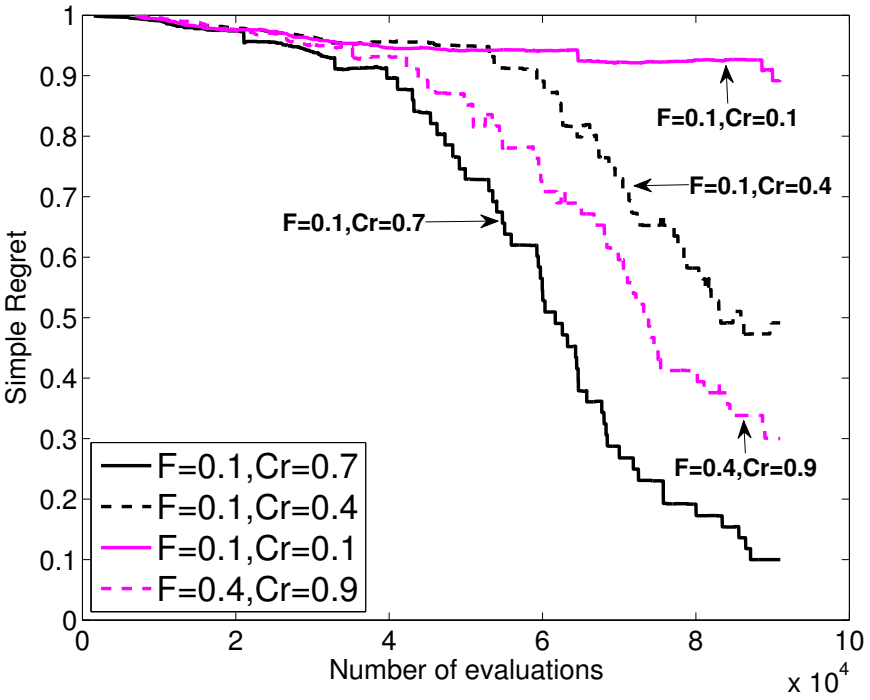
Figure 15.4 presents performance in terms of simple regret for different values of M for the truncated variant of DE presented in Section 15.2.3. We used $c_1 = 0.7$ and $c_2 = 0.7$. The rest of the settings are similar to Figure 15.3.

To facilitate the comparison of performance between normal DE and the truncated variant, we also plot the best normal DE found ($\lambda = 5$, $F = 0.1$, $Cr = 0.7$). Every tested variant of the truncated version outperform the normal DE. The best truncation factor for a population size $\lambda = 10$ appears to be $M = 0$ which indicates that early truncation are better.

From Figure 15.4, it clearly appears that smaller population size λ yields better results which is in line with previous findings. For a population size $\lambda = 5$, the best value of M is 500 in early iterations ($< 9 \times 10^4$) and, as the number of evaluations grows ($> 9 \times 10^4$) the best M is 2 000. These results indicates that truncating early yields good results yet if the budget is large enough it is better to gather more

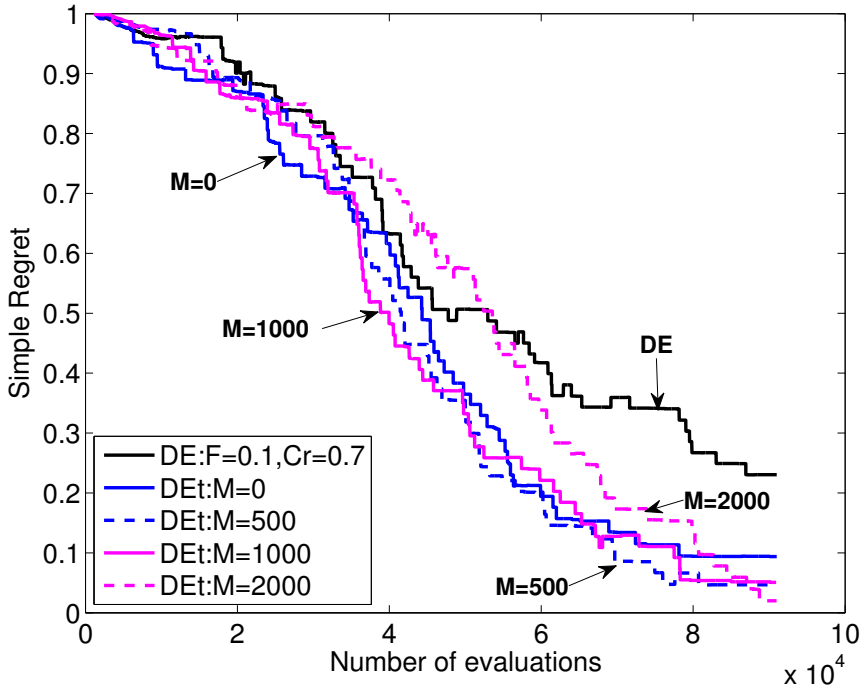


(a) Normal DE, $\lambda = 5$.

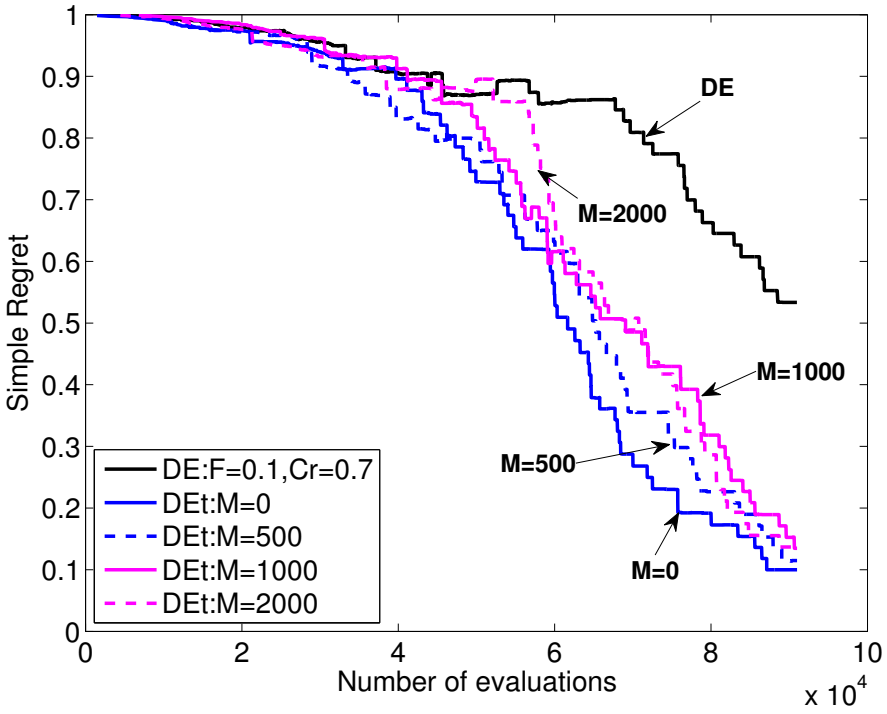


(b) Normal DE, $\lambda = 10$.

Figure 15.3: Performance of different variants of DE in terms of Simple Regret. Number of options $K = 24$, population size $\lambda = 5$ (Top) and $\lambda = 10$ (Bottom). Here we only show a summary of results for the parameters. When $F = 0.1$, bigger Cr gets better performance. Normal DE can not converge when near to optimum, which is to be expected from an evolutionary algorithm.



(a) Truncated-DE compared to normal DE, $\lambda = 5$.



(b) Truncated-DE compared to normal DE, $\lambda = 10$.

Figure 15.4: Performance of different variants of DE in terms of Simple Regret. Number of options $K = 24$, population size $\lambda = 5$ (Top) and $\lambda = 10$ (Bottom). Here we only show a summary of results for the parameters. Black solid curve is the best DE found in Figure 15.3. Truncated-DE clearly outperforms the normal DE. M influences slightly on the convergence rate.

information before starting the truncation process.

15.2.4.4 DE vs Bandit

This section compares the best UCBs with the best variants of DE. It is important to note that each algorithm are tuned individually instead of using generic constant for a family of algorithms. In this section, we also add another baseline which consists in choosing an option randomly and always pulling the same one. We call this baseline *Random*.

Figure 15.5 shows the average performance in terms of Simple Regret of the best UCBs and DEs. The x-axis represents the number of evaluations and the y-axis shows the Simple Regret. For each point, 50 independent trials are executed and we present the mean of these results. Standard deviation for each curve is of order of magnitude of 0.015.

Given a decent number of evaluations, every algorithm outperforms the baseline *Random*. As expected, the best of the UCB variant (UCBdn) is the best algorithm up to a budget of 5.7×10^4. Afterward, truncated-DE for $\lambda = 5$, $M = 500$ is the best available algorithm.

There are two important conclusions that we can infer from these experiments. First, the variant of UCB (UCBdn) presented in this chapter is the best UCB algorithm for non-stationary bandit problem where the distribution varies with the number of times an option is selected. Second, Evolutionary Algorithm such as DE, given some adaptation, can outperform bandit specific algorithms on bandit problem.

15.2.5 Conclusion: UCBdn is preferable for small budget, truncated-DE is recommended for huge budget

In this chapter, we compare an evolutionary algorithm, namely DE, to bandit algorithms on a black-box portfolio selection problem. Such problems can be formalized into a non-stationary bandit problems, where each option increases their performance as the number of times we evaluate it increases.

First, we introduce our definition of the non-stationary bandit problem and justify such an approach through a direct application (portfolio of solvers for the Cart-Pole problem). Second, we present the current state-of-the-art bandit algorithms for the given problems and propose our own variant (UCBdn) that outperforms all the other UCB-like algo-

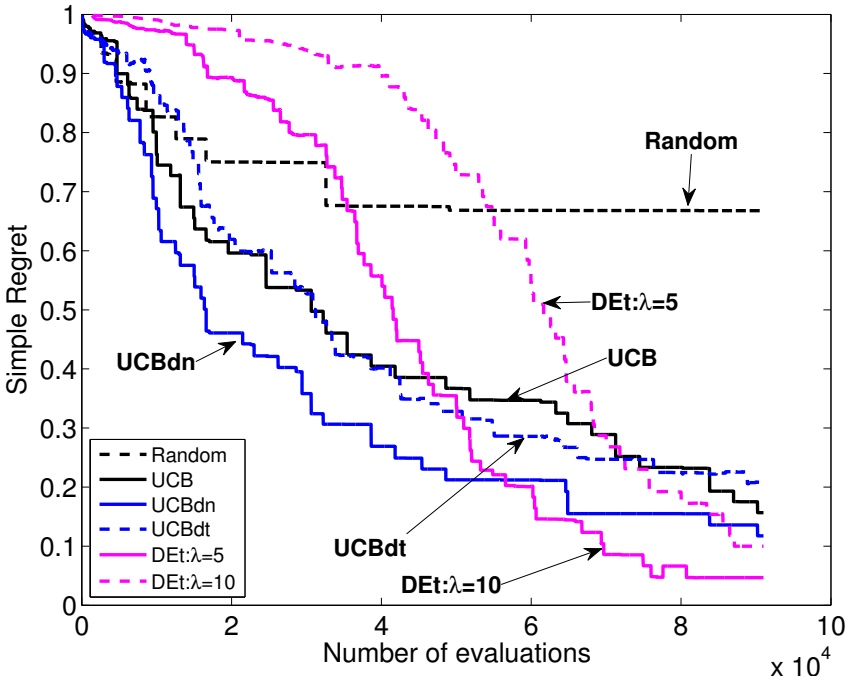


Figure 15.5: Comparison of truncated-DE, UCB, UCBdn, UCBdt and *Random*. UCBdn outperforms UCB and UCBdt and is clearly the best algorithm if the budget is small. However, as the number of evaluations grows (5.7×10^4), truncated-DE gets better (smaller) Simple Regret. Standard deviation for each curve is of order of magnitude of 0.015.

rithm on non-stationary distributions where each distribution changes according to the number of times the option is selected.

Third, we introduce DE, an evolutionary algorithm, and apply it over this specific non-stationary bandit problem. We also present our own variant of DE which outperforms the classic version.

Fourth and last, we compare the best variant of each category (UCBdn and truncated-DE). If the budget is smaller than 5.7×10^4 we propose the use of UCBdn and, as the number of evaluations increases, truncated-DE is better.

For future research, we intend to apply truncated-DE on a wider range of problems. Moreover, we would like to explore the idea of generic parameter tuning in the case of DE. As for the bandit algo-

rithm part, to compute theoretical bound for UCBdn would provide interesting information on its generic performance.

Algorithm 19 DE/best/1.

Input: K: Dimension
Input: T: Computational budget
Input: $Truncation$: 1 if truncated, 0 otherwise
Input: M: Threshold if truncated
Input: $|\mathcal{X}|$: The size of the population $\mathcal{X}$

```
Initialize each x ∈ 𝒳 randomly ∈ (0,1)^K
Normalize the values x_{k∈K}
Initialize t ← 1
Compute f(x) for each x ∈ 𝒳
Set a : f(a) ≥ f(x) ∈ 𝒳
while t ≤ T do
    if Truncation and t > M then
        Execute truncation
    end if
    for each x ∈ 𝒳 do
        Evaluate x t' times
        t ← t + t'
        Pick randomly 2 distinct individuals b and c from 𝒳
        Pick a random index R ∈ K
        Get a new position y for the individual x as follows:
        for each k ∈ K do
            Pick a random number rand ≡ 𝒰(0,1)
            if rand < Cr or k == R then
                Set y_k = a_k + F(b_k − c_k)
            else
                y_k = x_k
            end if
        end for
        Normalize the values y_k, ∀k ∈ K
        Compute f(y)
        if f(y) > f(x) then
            replace x by y
        end if
        if f(x) > f(a) then
            replace a by x
        end if
    end for
end while
Pick the agent x̃ ∈ 𝒳 using recommendation policy
Return k̃
```

Part V

Conclusion and Perspectives

16 Overview and conclusion

16.1 Contributions in noisy optimization

The first body of work in this manuscript is around noisy optimization, in particular with resampling methods, and with portfolio methods.

Noisy optimization algorithms have various rates, usually in the *log-log* scale (chapter 4). Our results show a fast rate as soon as the estimator of the location of the optimum has squared error $O(\sigma^{2z-2}/r)$, when using r points sampled adequately within distance $O(\sigma)$ of the optimum - with $z = 0$ for constant variance noise, and z greater corresponds to noise decreasing to zero in the neighborhood of the optimum.

Evolution strategies and differential evolution can become consistent in the strongly noisy case, when resamplings are applied (Chapters 5 and 6). We have proved mathematically that a non-adaptive rule with exponential number of resamplings can lead to *log-log* convergence, i.e. the logarithm of the distance to the optimum typically scales linearly with the logarithm of the number of evaluations, and have also shown experimentally that a non-adaptive rule with polynomial number of resamplings can lead to the same *log-log* convergence.

16.2 Conclusions in noisy optimization by portfolio methods

These population-based methods, in the case of noisy optimization, can often be improved a lot by slightly "grey box" optimization (Chapter 7). The impact on rates depends on the specific nature of the noise, and can be detrimental in some cases.

The slowest algorithms have, on the other hand, strong advantages in terms of robustness to multimodal settings (evolution strategies).

Also, the best algorithm depends on the kind of noise we have. The best algorithm strongly depends on the context:

- With large noise ($z = 0$, i.e. constant variance noise), Fabian's algorithm is the fastest, whereas with small noise other algorithms are competitive.
- In multimodal cases, RSAES is by far the most efficient solver among the solvers used in our experiments.
- On stochastic unit commitment problems, RSAES is usually the best NOA for small dimensions and variants of Fabian for large dimension.

Therefore it makes sense to combine several solvers into one single optimizer, with portfolio methods (part IV). We have theoretical guarantees that a portfolio reaches the optimal rate among its solvers: the computational overhead due to testing and comparing many methods is negligible. In both artificial and real world testcases, given a same budget, even a portfolio of identical solvers sometimes outperforms its components.

Sharing information in portfolios of noisy optimization algorithms is not trivial. Our setting of sharing used in NOPA and INOPA is rather preliminary and the experimental results did not produce significant improvements. As a summary, we get the same rate as the best algorithm, with a method (INOPA) which has an asymptotically negligible computational overhead.

16.3 Conclusions on computing Nash equilibria

16.3.1 Conclusions on the modeling of decision under uncertainty thanks to Nash equilibria

Nash equilibria are a specific case of portfolio: we design a stochastic combination of deterministic policy, which is optimal in terms of expected rewards in the worst case. The method is validated experimentally in Chapter 10, and relies on the computation of Nash equilibria in large matrix games. On the first hand, we provide a new model, different from Wald's criterion or Savage's criterion, for making decision under uncertainty. We argue that the Nash criterion makes sense for

decision under uncertainty, as well as other classical criteria, and leads to smaller computation time.

16.3.2 Conclusions on the computational aspects of Nash equilibria

Bandit algorithms for adversarial problems are efficient tools for computing Nash equilibria. In particular, for a matrix game, we do not have to compute all the entries of the matrix for approximating the Nash equilibrium [Grigoriadis and Khachiyan, 1995]. We provide improved rates, thanks to an a posteriori pruning, in Chapter 9.

16.4 Contribution in portfolio methods for adversarial problems.

Another body of work is the application of portfolio methods to artificial intelligences in games (Chapter 11). This work has been performed in several settings:

- with a portfolio of variants of a given AI (different parametrizations of a game-playing program);
- with variants generated by testing several random seeds of a stochastic AI.

The latter is based on the key principle that random seeds have a significant impact on the performance. Typically, a seed which leads to the right move in 2 or 3 critical situations (usually in the opening book) is much better than a seed which does not. Optimizing random seeds is therefore useful. This has no impact on the computation time.

It is much better, in terms of robustness against a learning opponent, to have a probability distribution on random seeds (see Figures 11.6, 11.7, 11.8); we could get such a probability distribution thanks to the use of Nash equilibria. For all these methods based on random seeds, the improvement is particularly big for small time settings.

Such an optimization of random seeds, or more generally the combination of deterministic methods into one single stochastic method optimal in a Nash-sense, involves computing Nash equilibria. The optimization of random seeds for portfolio of game AIs is then tested on

several games (Domineering, Atari-go, Breakthrough and Go) in Chapter 11.

In combinatorial optimization, sometimes, portfolio methods change their choice dynamically. We do the same, but in games, in Chapter 12. We get a method with a better asymptotic behaviour than Monte Carlo Tree Search (Figure 12.8). This has been validated on the game of Go, by comparing the original MCTS and our improved version with position-specific random seeds. Interestingly, we outperform MCTS, for a same number of simulations, whereas our simulations are faster and more parallel.

17 Further work

For further research, we would like to explore the following ideas:

- **Resamplings rules.**
  The main further work is the mathematical analysis of the polynomial number of resamplings in the non-adaptive case.

  Adaptive resampling rules are sensitive to parameters and special cases, but they can save up fitness evaluations. A combination of adaptive and non-adaptive rules might be interesting; adaptive rules are intuitively satisfactory, but non-adaptive polynomial rules provide simple efficient solutions, with empirically easy (no tuning) results. A work in progress is to use Bernstein races or resampling numbers depending on step-sizes and non-adaptive methods studied in Chapter 6.

  A related issue is comparing resampling to some unproved but interesting state of the art approaches: "large population" and "mutate large, but inherit small" (MLIS). Some works in progress in the team suggest that a rate $SR_n = O(1/n)$ is possible for evolutionary algorithms applying MLIS.

- **Variance reduction techniques.**
  Other variance reduction techniques are possible. A nice challenge for future research is to find algorithms protecting variance reduction techniques from their possible detrimental effects. In particular importance sampling with optimal allocation per stratum (though we need variance estimates for that, which is difficult in a noisy optimization setting), quasi Monte Carlo (more difficult in a nearly black-box setting), or quantization [Defourny, 2010, Dupacová et al., 2000].

- **Portfolio methods in noisy optimization.**
  A further work consists in identifying relevant information for

sharing in portfolios of noisy optimization algorithms; 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, then we can safely stop its run and save up computational power.

- **Expansion of portfolio concept.**
  The portfolio concept is not limited to a combination of algorithms. [Berthier et al., 2015] has combined parametric DPS policies with a more generic neural network. Another analogue is some evolutionary algorithms which use μ nonidentical parents. This can be regarded as a portfolio of different start points.

- **Portfolio methods in games.**
  Chapter 11 has shown that our boosted AIs significantly outperform the baselines on some games. The main further work is the use of bandit algorithms, e.g. as [Lai and Robbins, 1985] for BestSeed and [Auer et al., 1995] for Nash, in order to decrease the offline computational cost. We can basically decrease the computational cost to its square root, up to logarithmic factors (see [Grigoriadis and Khachiyan, 1995]) and with a minor cost in terms of precision.

Noisy optimization algorithms

We present briefly several noisy optimization algorithms. Algorithm 20 is a classical Self Adaptive-(μ,λ)-Evolution Strategy, with noise handled by resamplings. Algorithm 21 is a stochastic gradient method, with gradient estimated by finite differences; it is known to converge with simple regret $O(1/n)$ on smooth enough functions corrupted by additive noise [Fabian, 1967, Shamir, 2013]. Algorithm 22 extends Fabian's algorithm by adding second-order information, by approximating the Hessian [Fabian, 1971].

Algorithm 20 Self-Adaptive Evolution Strategy with resamplings. $\mathcal{N}$ denotes some independent standard Gaussian random variable and $c = mod(a, b)$ for $b > 0$ means $\exists k \in \mathbb{Z}, (a - c) = bk$ and $0 \leq c < b$.

Input: dimension $d \in \mathbb{N}^*$

Input: population size $\lambda \in \mathbb{N}^*$ and number of parents $\mu \in \mathbb{N}^*$ with $\lambda \geq \mu$

Input: function $N_{resample} : \cdot \mapsto \mathbb{N}^*$ ▷ Function used to compute resampling number

Input: an initial parent $x_{1,i} \in \mathbb{R}^d$ and an initial $\sigma_{1,i} = 1$, $i \in \{1, \ldots, \mu\}$

$n \leftarrow 1$

$\tilde{x} \leftarrow x_{1,1}$ ▷ recommendation

while (true) **do**

Generate λ individuals i_j, $j \in \{1, \ldots, \lambda\}$, independently using ▷ offspring

$$\sigma_j = \sigma_{n,mod(j-1,\mu)+1} \times \exp\left(\frac{\mathcal{N}}{2d}\right) \tag{17.1}$$

$$i_j = x_{n,mod(j-1,\mu)+1} + \sigma_j \mathcal{N} \tag{17.2}$$

Evaluate each of them $\lceil N_{resample} \rceil$ times and average their fitness values

Define $j_1, \ldots, j_\lambda$ so that ▷ ranking

$$\hat{\mathbb{E}}_{\lceil N_{resample} \rceil}[f(i_{j_1})] \leq \hat{\mathbb{E}}_{\lceil N_{resample} \rceil}[f(i_{j_2})] \cdots \leq \hat{\mathbb{E}}_{\lceil N_{resample} \rceil}[f(i_{j_\lambda})]$$

where $\hat{E}_m$ denotes the average over m resamplings

Compute $x_{n+1,k}$ and $\sigma_{n+1,k}$ using ▷ update

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

$\tilde{x} = i_{j_1}$ ▷ update recommendation

$n \leftarrow n + 1$

end while

Algorithm 21 Fabian's stochastic gradient algorithm with finite differences. Fabian, in [Fabian, 1967], proposes various rules for the parametrization; in the present paper, we use the following parameters. s is as in Remark 5.2 in [Fabian, 1967], i.e., s is the minimal even number $\geq \frac{1}{2\gamma} - 1$. The scales u_i are $u_i = \frac{1}{i}, \forall i \in \{1, \ldots, \frac{s}{2}\}$; this generalizes the choice in Example 3.3 in [Fabian, 1967]. The w_i are computed as given in Lemma 3.1 in [Fabian, 1967]. e_i is the i^{th} vector of the standard orthonormal basis of $\mathbb{R}^d$.

Input: dimension $d \in \mathbb{N}^*$

Input: $\frac{1}{2} > \gamma > 0$, $a > 0$, $c > 0$, even number of samples per axis s

Input: scales $1 \geq u_1 > \cdots > u_{\frac{s}{2}} > 0$, weights $w_1 > \cdots > w_{\frac{s}{2}}$ summing to 1

Input: an initial $x_1 \in \mathbb{R}^d$

$n \leftarrow 1$

$\tilde{x} \leftarrow x_1$ ▷ recommendation

while (true) **do**

  Compute $\sigma_n = c/n^\gamma$ ▷ step-size

  Evaluate the gradient g at x_n by finite differences, averaging over s samples per axis: $\forall i \in \{1, \ldots, d\}, \forall j \in \{1, \ldots, \frac{s}{2}\}$ ▷ gradient estimation

$$x_n^{(i,j)+} = x_n + u_j \sigma_n e_i \text{ and } x_n^{(i,j)-} = x_n - u_j \sigma_n e_i$$

$$g_i = \frac{1}{2\sigma_n} \sum_{j=1}^{s/2} w_j \left(f(x_n^{(i,j)+}) - f(x_n^{(i,j)-}) \right)$$

  Apply $x_{n+1} = x_n - \frac{a}{n} g$ ▷ next search point

  $\tilde{x} \leftarrow x_{n+1}$ ▷ update recommendation

  $n \leftarrow n + 1$

end while

Algorithm 22 An adaptation of Newton's algorithm for noisy objective functions, with gradient and Hessian approximated by finite differences and revaluations. The recommendations are the x_n's. e_i is the i^{th} vector of the standard orthonormal basis of $\mathbb{R}^d$.

Input: dimension $d \in \mathbb{N}^*$
Input: $A > 0$, $B > 0$, $\alpha > 0$, $\beta > 0$
Input: an initial $x_1 \in \mathbb{R}^d$

$n \leftarrow 1$
$\tilde{x} \leftarrow x_1$ ▷ recommendation
$\hat{h} \leftarrow$ identity matrix
while (true) **do**
    Compute $\sigma_n = A/n^\alpha$ ▷ step-size
    for $i = 1$ to d **do**
        Evaluate g_i by finite differences at $x_n + \sigma_n e_i$ and $x_n - \sigma_n e_i$, averaging each evaluation over $\lceil Bn^\beta \rceil$ resamplings.
    end for
    for $i = 1$ to d **do**
        Evaluate $\hat{h}_{i,i}$ by finite differences at $x_n + \sigma_n e_i$, x_n and $x_n - \sigma_n e_i$, averaging each evaluation over $\lceil Bn^\beta \rceil$ resamplings
        for $j = 1$ to d, $j \neq i$ **do**
            Evaluate $\hat{h}_{i,j}$ by finite differences thanks to evaluations at each of $x_n \pm \sigma_n e_i \pm \sigma_n e_j$, averaging over $\lceil Bn^\beta/10 \rceil$ resamplings
        end for
    end for
    $\delta \leftarrow$ solution of $\hat{h}\delta = -g$ ▷ possible next search point
    if $||\delta|| > \frac{1}{2}\sigma_n$ **then**
        $\delta = \frac{1}{2}\sigma_n \frac{\delta}{||\delta||}$ ▷ trust region style
    end if
    Apply $x_{n+1} = x_n + \delta$
    $\tilde{x} \leftarrow x_{n+1}$ ▷ update recommendation
    $n \leftarrow n + 1$
end while

Summary of notations

π	=	policy
t	=	time step
T	=	budget (time steps or computational time)
a_t	=	action (decision) at t
A	=	action (decision) space
s_t	=	state at t
S	=	state space
r	=	reward
$\mathcal{N}$	=	a Gaussian standard noise
		General notations:
$\mathbb{E}_w$	=	expectation with respect to random variable w.
$\hat{E}_k X$	=	average over k independent realizations of random variable X.
		Notation for solvers:
x_n	=	search point used by the solver for the n^{th} evaluation.
$\tilde{x}_n$	=	recommendation given by the solver after the n^{th} evaluation.
SR_n	=	$\mathbb{E}\left(f(\tilde{x}_n) - f(x^*)\right)$. (simple regret)
		Notation for AS algorithms:
i^*	=	index of the solver chosen by the AS algorithm.
$\tilde{x}_{i,n}$	=	recommendation given by the solver i after the n^{th} evaluation.
$SR_{i,n}$	=	$\mathbb{E}\left(f(\tilde{x}_{i,n}) - f(x^*)\right)$.
M	=	number of solvers in portfolio.
$\Delta_{i,n}$	=	$SR_{i,n} - \min_{j \in \{1,\dots,M\}} SR_{j,n}$.
$SR_n^{Solvers}$	=	$\min_{i \in \{1,\dots M\}} SR_{i,n}$.
$SR_n^{Selection}$	=	$\mathbb{E}\left(f(\tilde{x}_{i^*,n}) - f(x^*)\right)$.

Acronyms

AI: Artificial Intelligence
BS: Best Seed
CR: Cumulative Regret
CRN: Common Random Number
DE: Differential Evolution
DP: Dynamique Programming
DPS: Direct Policy Search
INOPA: Improved Noisy Optimization Portfolio Algorithm
LP: Linear Programming
MAB: Multi-Armed Bandit
MC: Monte-Carlo
MCTS: Monte-Carlo Tree Search
MDP: Markov Decision Processes
ML: Machine Learning
NE: Nash Equilibrium
NOPA: Noisy Optimization Portfolio Algorithm
RL: Reinforcement learning
RSAES: Self-Adaptive Evolution Strategy with resamplings
SAES: Self-Adaptive Evolution Strategy
SDP: Stochastic Dynamique Programming
SR: Simple Regret

Index

List of Algorithms

List of Figures

List of Tables

Bibliography

[Aha, 1992] Aha, D. W. (1992). Generalizing from case studies: A case study. In *Proceedings of the 9th International Workshop on Machine Learning*, pages 1–10. Morgan Kaufmann Publishers Inc.

[Anderson and Ferris, 2001] Anderson, E. J. and Ferris, M. C. (2001). A direct search algorithm for optimization with noisy function evaluations. *SIAM Journal on Optimization*, 11:837–857.

[Ardia et al., 2011a] Ardia, D., Boudt, K., Carl, P., Mullen, K., and Peterson, B. (2011a). Differential evolution with deoptim: An application to non-convex portfolio optimization. *The R Journal*, 3(1):27–34.

[Ardia et al., 2011b] Ardia, D., David, J., Arango, O., and Gómez, N. D. G. (2011b). Jump-diffusion calibration using differential evolution. *Wilmott*, 2011(55):76–79.

[Armstrong et al., 2006] Armstrong, W., Christen, P., McCreath, E., and Rendell, A. P. (2006). Dynamic algorithm selection using reinforcement learning. In *International Workshop on Integrating AI and Data Mining*, pages 18–25.

[Arnold and Beyer, 2000] Arnold, D. V. and Beyer, H.-G. (2000). Efficiency and mutation strength adaptation of the (μ/μ i, λ)-es in a noisy environment. In *Parallel Problem Solving from Nature PPSN VI*, pages 39–48. Springer.

[Arnold and Beyer, 2006] Arnold, D. V. and Beyer, H.-G. (2006). A general noise model and its effects on evolution strategy performance. *IEEE Transactions on Evolutionary Computation*, 10(4):380–391.

[Astest Morales et al.,] Astest Morales, S., Cauwet, M.-L., Liu, J., and Teytaud, O. Noisy optimization rates. *Theoretical Computer Science*, page Accepted.

[Astete-Morales et al., 2015] Astete-Morales, S., Cauwet, M.-L., and Teytaud, O. (2015). Evolution strategies with additive noise: A convergence rate lower bound. In *Foundations of Genetic Algorithms*, page 9, Aberythswyth, United Kingdom.

[Astete-Morales et al., 2013] Astete-Morales, S., Liu, J., and Teytaud, O. (2013). log-log convergence for noisy optimization. In *Proceedings of EA 2013*, LLNCS, page accepted. Springer.

[Aubin, 1991] Aubin, J.-P. (1991). *Viability Theory*. Birkhauser Boston Inc., Cambridge, MA, USA.

[Audibert et al., 2007] Audibert, J.-Y., Munos, R., and Szepesvári, C. (2007). Tuning bandit algorithms in stochastic environments. In *Algorithmic Learning Theory*, pages 150–165. Springer.

[Audibert et al., 2009] Audibert, J.-Y., Munos, R., and Szepesvári, C. (2009). Exploration–exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 410(19):1876–1902.

[Audibert et al., 2006] Audibert, J.-Y., Munos, R., Szepesvari, C., et al. (2006). Use of variance estimation in the multi-armed bandit problem. *NIPS 2006 Workshop on On-line Trading of Exploration and Exploitation*.

[Auer, 2003] Auer, P. (2003). Using confidence bounds for exploitation-exploration trade-offs. *The Journal of Machine Learning Research*, 3:397–422.

[Auer et al., 2002a] Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002a). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256.

[Auer et al., 1995] Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. E. (1995). 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.

[Auer et al., 2002b] Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. E. (2002b). The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77.

[Auger, 2005] Auger, A. (2005). Convergence results for (1,λ)-SA-ES using the theory of φ-irreducible Markov chains. *Theoretical Computer Science*, 334(1-3):35–69.

[Auger et al., 2010] Auger, A., Finck, S., Hansen, N., and Ros, R. (2010). BBOB 2009: Comparison Tables of All Algorithms on All Noisy Functions. Technical Report RT-0384, INRIA.

[Auger et al., 2005] Auger, A., Jebalia, M., and Teytaud, O. (2005). Xse: quasi-random mutations for evolution strategies. In *Proceedings of Evolutionary Algorithms, 12 pages.*

[Baudiš and Pošík, 2014] Baudiš, P. and Pošík, P. (2014). Online black-box algorithm portfolios for continuous optimization. In *Parallel Problem Solving from Nature–PPSN XIII*, pages 40–49. Springer.

[Baudiš and Gailly, 2011] Baudiš, P. and Gailly, J. (2011). PACHI: State of the Art Open Source Go Program. In *Proc. 13th Int. Conf. Adv. Comput. Games, LNCS 7168*, pages 24–38, Tilburg, The Netherlands. Springer.

[Bayer et al., 2008] Bayer, A., Bump, D., Daniel, E. B., Denholm, D., Dumonteil, J., Farnebäck, G., Pogonyshev, P., Traber, T., Urvoy, T., and Wallin, I. (2008). Gnu go 3.8 documentation. Technical report, Free Software Fundation.

[Bellman, 1957] Bellman, R. (1957). *Dynamic Programming.* Princeton Univ. Press.

[Bengio, 1997] Bengio, Y. (1997). Using a financial training criterion rather than a prediction criterion. *International Journal of Neural Systems*, 8(4):433–443. Special issue on noisy time-series.

[Berthier et al., 2015] Berthier, V., Couetoux, A., and Teytaud, O. (2015). Combining policies: the best of human expertise and neuro-control. In *Proceedings of EA 2015*, LLNCS, page accepted. Springer.

[Beyer, 2001] Beyer, H.-G. (2001). *The Theory of Evolution Strategies.* Natural Computing Series. Springer, Heideberg.

[Beyer and Finck, 2008] Beyer, H.-G. and Finck, S. (2008). On the performance of evolution strategies on noisy pdqfs:progress rate analysis. In *IEEE Congress on Evolutionary Computation*, pages 495–502. IEEE.

[Billings et al., 1999] Billings, D., Castillo, L. P., Schaeffer, J., and Szafron, D. (1999). Using probabilistic knowledge and simulation to play poker. In *AAAI/IAAI*, pages 697–703.

[Billingsley, 1986] Billingsley, P. (1986). *Probability and Measure*. John Wiley and Sons.

[Borrett and Tsang, 1996] Borrett, J. and Tsang, E. P. K. (1996). Towards a formal framework for comparing constraint satisfaction problem formulations. Technical report, University of Essex, Department of Computer Science.

[Bottou and Bousquet, 2011] Bottou, L. and Bousquet, O. (2011). The tradeoffs of large scale learning. In Sra, S., Nowozin, S., and Wright, S. J., editors, *Optimization for Machine Learning*, pages 351–368. MIT Press.

[Bourki et al., 2010] Bourki, A., Coulm, M., Rolet, P., Teytaud, O., Vayssiere, P., et al. (2010). Parameter tuning by simple regret algorithms and multiple simultaneous hypothesis testing. *ICINCO 2010*.

[Bouzy, 2004] Bouzy, B. (2004). Associating shallow and selective global tree search with Monte Carlo for 9x9 Go. In *4rd Computer and Games Conference, Ramat-Gan*.

[Breiman, 1996] Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

[Brenner and Cummings, 1972] Brenner, J. and Cummings, L. (1972). The Hadamard maximum determinant problem. *Amer. Math. Monthly*, 79:626–630.

[Brest et al., 2006] Brest, J., Greiner, S., Boskovic, B., Mernik, M., and Zumer, V. (2006). Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation*, 10(6):646–657.

[Breuker et al., 2000] Breuker, D., Uiterwijk, J., and van den Herik, H. (2000). Solving 8x8 domineering. *Theoretical Computer Science*, 230(1-2):195 – 206.

[Broyden, 1970] Broyden, C. G. (1970). The convergence of a class of double-rank minimization algorithms 2. *The New Algorithm. J. of the Inst. for Math. and Applications*, 6:222–231.

[Bruegmann, 1993] Bruegmann, B. (1993). Monte-carlo Go (unpublished draft http://www.althofer.de/bruegmann-montecarlogo.pdf).

[Bubeck and Cesa-Bianchi, 2012] Bubeck, S. and Cesa-Bianchi, N. (2012). Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *arXiv preprint arXiv:1204.5721*.

[Bubeck et al., 2009] Bubeck, S., Munos, R., and Stoltz, G. (2009). Pure exploration in multi-armed bandits problems. In *Algorithmic Learning Theory*, pages 23–37. Springer.

[Bubeck et al., 2011] Bubeck, S., Munos, R., and Stoltz, G. (2011). Pure exploration in finitely-armed and continuous-armed bandits. *Theoretical Computer Science*, 412(19):1832–1852.

[Bullock, 2002] Bullock, N. (2002). Domineering: Solving large combinatorial search spaces. *ICGA Journal*, 25(2):67–84.

[Burnetas and Katehakis, 1996] Burnetas, A. N. and Katehakis, M. N. (1996). Optimal adaptive policies for sequential allocation problems. *Advances in Applied Mathematics*, 17(2):122–142.

[Busa-Fekete et al., 2010] Busa-Fekete, R., Kégl, B., et al. (2010). Fast boosting using adversarial bandits. In *Proceedings of the 27th International Conference on Machine Learning*, pages 143–150.

[Cappé et al., 2013] Cappé, O., Garivier, A., Maillard, O.-A., Munos, R., and Stoltz, G. (2013). Kullback-Leibler Upper Confidence Bounds for Optimal Sequential Allocation. *Annals of Statistics*, 41(3):1516–1541. Accepted, to appear in Annals of Statistics.

[Cauwet et al., 2014] Cauwet, M., Liu, J., and Teytaud, O. (2014). Algorithm portfolios for noisy optimization: Compare solvers early. In *Learning and Intelligent Optimization - 8th International Conference, Lion 8, Gainesville, FL, USA, February 16-21, 2014. Revised Selected Papers*, pages 1–15.

[Cazenave, 2006] Cazenave, T. (2006). A phantom-go program. In van den Herik, H. J., Hsu, S.-C., Hsu, T.-S., and Donkers, H. H. L. M., editors, *Proceedings of Advances in Computer Games*, volume 4250 of *Lecture Notes in Computer Science*, pages 120–125. Springer.

[Cazenave, 2009] Cazenave, T. (2009). Nested monte-carlo search. In Boutilier, C., editor, *IJCAI*, pages 456–461.

[Cazenave and Borsboom, 2007] Cazenave, T. and Borsboom, J. (2007). Golois wins phantom go tournament. *ICGA Journal*, 30(3):165–166.

[Cazenave and Saffidine, 2011] Cazenave, T. and Saffidine, A. (2011). Score bounded monte-carlo tree search. In *Proceedings of the 7th International Conference on Computers and Games*, CG'10, pages 93–104, Berlin, Heidelberg. Springer-Verlag.

[Chapel and Deffuant, 2006] Chapel, L. and Deffuant, G. (2006). SVM viability controller active learning. In *Kernel machines and Reinforcement Learning Workshop - ICML 2006*, United States.

[Chaslot et al., 2008] Chaslot, I. S. G., Winands, M. H., and van den Herik, H. (2008). Parameter tuning by the cross-entropy method. In *European workshop on reinforcement learning*.

[Chaudry et al., 2014] Chaudry, M., Jenkins, N., Qadrdan, M., and Wu, J. (2014). Combined gas and electricity network expansion planning. 113(C):1171–1187.

[Chen, 1988] Chen, H. (1988). Lower rate of convergence for locating the maximum of a function. *Annals of statistics*, 16:1330–1334.

[Chen et al., 1996] Chen, H. F., Duncan, T. E., and Pasik-Duncan, B. (1996). A stochastic approximation algorithm with random differences. In *Proceedings of the 13th IFAC World Congress*, volume H, pages 493–496.

[Chou et al., 2013] Chou, C.-W., Chou, P.-C., Christophe, J.-J., Couetoux, A., De, P., Galichet, N., Lee, C.-S., Liu, J., Sebag, M., Teytaud, O., et al. (2013). Strategic choices in optimization. *Journal of Information Sciences and Engineering*.

[Chou et al., 2012] Chou, C.-W., Chou, P.-C., Lee, C.-S., Saint-Pierre, D. L., Teytaud, O., Wang, M.-H., Wu, L.-W., and Yen, S.-J. (2012). Strategic choices: Small budgets and simple regret. In *Technologies and Applications of Artificial Intelligence (TAAI), 2012 Conference on*, pages 182–187. IEEE.

[Coelho and Mariani, 2006] Coelho, L. S. and Mariani, V. C. (2006). Combining of chaotic differential evolution and quadratic programming for economic dispatch optimization with valve-point effect. *Power Systems, IEEE Transactions on*, 21(2):989–996.

[Couetoux, 2013] Couetoux, A. (2013). *Monte Carlo Tree Search for Continuous and Stochastic Sequential Decision Making Problems*. Theses, Université Paris Sud - Paris XI.

[Coulom, 2006] Coulom, R. (2006). Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. *In P. Ciancarini and H. J. van den Herik, editors, Proceedings of the 5th International Conference on Computers and Games, Turin, Italy*, pages 72–83.

[Coulom, 2007] Coulom, R. (2007). Computing Elo ratings of move patterns in the game of Go. *ICGA Journal*, 30(4):198–208.

[Coulom, 2012] Coulom, R. (2012). Clop: Confident local optimization for noisy black-box parameter tuning. In *Advances in Computer Games*, pages 146–157. Springer Berlin Heidelberg.

[Coulom et al., 2011] Coulom, R., Rolet, P., Sokolovska, N., and Teytaud, O. (2011). Handling expensive optimization with large noise. In *Foundations of Genetic Algorithms, 11th International Workshop, FOGA 2011, Schwarzenberg, Austria, January 5-8, 2011, Proceedings*, pages 61–68.

[Cranley and Patterson, 1976] Cranley, R. and Patterson, T. (1976). Randomization of number theoretic methods for multiple integration. *SIAM J. Numer. Anal.*, 13(6):904,1914.

[Dantzig and Thapa, 2003] Dantzig, G. and Thapa, M. (2003). *Linear Programming 2: Theory and Extensions*. Springer.

[Das et al., 2005] Das, S., Konar, A., and Chakraborty, U. K. (2005). Improved differential evolution algorithms for handling noisy optimization problems. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1691–1698 Vol. 2.

[de Matos et al., 2012] de Matos, V., Philpott, A., and Finardi, E. (2012). Improving the performance of stochastic dual dynamic programming. *Applications – OR and Management Sciences (Scheduling)*.

[Decock et al., 2015] Decock, J., Liu, J., and Tetaud, O. (2015). Variance reduction in population-based optimization: Application to unit commitment. In *Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference*, GECCO Companion '15, pages 1377–1378, New York, NY, USA. ACM.

[Decock and Teytaud, 2013] Decock, J. and Teytaud, O. (2013). Noisy optimization complexity under locality assumption. In *Proceedings of the twelfth workshop on Foundations of genetic algorithms XII*, FOGA XII '13, pages 183–190, New York, NY, USA. ACM.

[Defourny, 2010] Defourny, B. (2010). *Machine Learning Solution Methods for Multistage Stochastic Programming*. PhD thesis, Institut Montefiore, Université de Liège.

[Dowell and Jarratt, 1972] Dowell, M. and Jarratt, P. (1972). The "pegasus" method for computing the root of an equation. *BIT Numerical Mathematics*, 12(4):503–508.

[Dubois and Prade, 2012] Dubois, D. and Prade, H. (2012). Possibility theory. In Meyers, R. A., editor, *Computational Complexity*, pages 2240–2252. Springer New York.

[Dupacová et al., 2000] Dupacová, J., Gröwe-Kuska, N., and Römisch, W. (2000). Scenario reduction in stochastic programming: An approach using probability metrics. Number 20 in Stochastic Programming E-Print Series. Institut für Mathematik. published; Springer; Berlin [u.a.]; Mathematical Programming; 95; 2003; 3.

[Dupač, 1957] Dupač, V. (1957). O Kiefer-Wolfowitzově aproximační Methodě. *Časopis pro pěstování matematiky*, 082(1):47–75.

[Fabian, 1967] Fabian, V. (1967). Stochastic Approximation of Minima with Improved Asymptotic Speed. *Annals of Mathematical statistics*, 38:191–200.

[Fabian, 1971] Fabian, V. (1971). *Stochastic Approximation*. SLP. Department of Statistics and Probability, Michigan State University.

[Feng, 2014] Feng, Y. (2014). *Scenario Generation and Reduction for Long-term and Short-term Power System Generation Planning under Uncertainties*. PhD thesis.

[Fey et al., 1996] Fey, M., McKelvey, R. D., and Palfrey, T. R. (1996). An experimental study of constant-sum centipede games. *International Journal of Game Theory*, 25(3):269–287.

[Finck et al., 2011] Finck, S., Beyer, H.-G., and Melkozerov, A. (2011). Noisy optimization: a theoretical strategy comparison of es, egs, spsa & if on the noisy sphere. In Krasnogor, N. and Lanzi, P. L., editors, *GECCO*, pages 813–820. ACM.

[Fletcher, 1970] Fletcher, R. (1970). A new approach to variable-metric algorithms. *Computer Journal*, 13:317–322.

[Flory and Teytaud, 2011] Flory, S. and Teytaud, O. (2011). Upper confidence trees with short term partial information. In *Procedings of EvoGames 2011*, page accepted. Springer.

[Fournier and Teytaud, 2011] Fournier, H. and Teytaud, O. (2011). Lower bounds for comparison based evolution strategies using vc-dimension and sign patterns. *Algorithmica*, 59(3):387–408.

[Gagliolo and Schmidhuber, 2005] Gagliolo, M. and Schmidhuber, J. (2005). A neural network model for inter-problem adaptive online time allocation. In *15th International Conference on Artificial Neural Networks: Formal Models and Their Applications*, pages 7–12. Springer.

[Gagliolo and Schmidhuber, 2006] Gagliolo, M. and Schmidhuber, J. (2006). Learning dynamic algorithm portfolios. volume 47, pages 295–328.

[Gale and Tucker, 1951] Gale, K. and Tucker (1951). Linear programming and the theory of games. In Koopmans, editor, *Activity Analysis of Production and Allocation*, chapter XII. Wiley.

[Gardner and Ball, 1974] Gardner, M. and Ball, W. (1974). Mathematical games. *Scientific American*, 231:187–191.

[Garivier and Cappé, 2011] Garivier, A. and Cappé, O. (2011). The kl-ucb algorithm for bounded stochastic bandits and beyond. *arXiv preprint arXiv:1102.2490*.

[Garivier and Moulines, 2008] Garivier, A. and Moulines, E. (2008). On upper-confidence bound policies for non-stationary bandit problems. *arXiv preprint arXiv:0805.3415*.

[Gaudel et al., 2010] Gaudel, R., Hoock, J.-B., Pérez, J., Sokolovska, N., and Teytaud, O. (2010). A Principled Method for Exploiting Opening Books. In *International Conference on Computers and Games*, pages 136–144, Kanazawa, Japon.

[Gelly and Silver, 2007] Gelly, S. and Silver, D. (2007). Combining online and offline knowledge in uct. In *Proceedings of the 24th international conference on Machine learning*, pages 273–280. ACM.

[Gelly et al., 2006] Gelly, S., Yizao, W., Munos, R., and Teytaud, O. (2006). Modification of uct with patterns in monte-carlo go. Technical report.

[Goldfarb, 1970] Goldfarb, D. (1970). A family of variable-metric algorithms derived by variational means. *Mathematics of Computation*, 24:23–26.

[Grigoriadis and Khachiyan, 1995] Grigoriadis, M. D. and Khachiyan, L. G. (1995). A sublinear-time randomized approximation algorithm for matrix games. *Operations Research Letters*, 18(2):53–58.

[Hadamard, 1893] Hadamard, J. (1893). Résolution d'une question relative aux déterminants. *Bull. Sci. Math.*, 17:240–246.

[Hamadi, 2013] Hamadi, Y. (2013). *Search: from Algorithms to Systems (HDR)*. Habilitation à diriger des recherches, Université Paris-Sud.

[Hamzaçebi and Kutay, 2007] Hamzaçebi, C. and Kutay, F. (2007). Continuous functions minimization by dynamic random search technique. *Applied Mathematical Modelling*, 31(10):2189–2198.

[Hansen et al., 2009] Hansen, N., Niederberger, S., Guzzella, L., and Koumoutsakos, P. (2009). A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *IEEE Transactions on Evolutionary Computation*, 13(1):180–197.

[Heidrich-Meisner and Igel, 2009] Heidrich-Meisner, V. and Igel, C. (2009). Hoeffding and bernstein races for selecting policies in evolutionary direct policy search. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 401–408, New York, NY, USA. ACM.

[Hoeffding, 1963] Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30.

[Huang et al., 2010] Huang, S.-C., Coulom, R., and Lin, S.-S. (2010). Monte-carlo simulation balancing in practice. In van den Herik, H. J., Iida, H., and Plaat, A., editors, *Computers and Games*, volume 6515 of *Lecture Notes in Computer Science*, pages 81–92. Springer.

[Jebalia and Auger, 2008] Jebalia, M. and Auger, A. (2008). On multiplicative noise models for stochastic search. In et a.l., G. R., editor, *Conference on Parallel Problem Solving from Nature (PPSN X)*, volume 5199, pages 52–61, Berlin, Heidelberg. Springer Verlag.

[Jebalia et al., 2010] Jebalia, M., Auger, A., and Hansen, N. (2010). Log-linear convergence and divergence of the scale-invariant (1+1)-ES in noisy environments. *Algorithmica*, pages 1–36. online first.

[Jones et al., 1998] Jones, D., Schonlau, M., and Welch, W. (1998). Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13(4):455–492.

[Kadioglu et al., 2011] Kadioglu, S., Malitsky, Y., Sabharwal, A., Samulowitz, H., and Sellmann, M. (2011). Algorithm selection and scheduling. In *17th International Conference on Principles and Practice of Constraint Programming*, pages 454–469.

[Katehakis and Veinott Jr, 1987] Katehakis, M. N. and Veinott Jr, A. F. (1987). The multi-armed bandit problem: decomposition and computation. *Mathematics of Operations Research*, 12(2):262–268.

[Kiefer and Wolfowitz, 1952] Kiefer, J. and Wolfowitz, J. (1952). Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics 23*, 3:462–466.

[Kim and Zhang, 2010] Kim, S. and Zhang, D. (2010). Convergence properties of direct search methods for stochastic optimization. In

Proceedings of the Winter Simulation Conference, WSC '10, pages 1003–1011. Winter Simulation Conference.

[Kleinman et al., 1999] Kleinman, N. L., Spall, J. C., and Naiman, D. Q. (1999). Simulation-based optimization with stochastic approximation using common random numbers. *Management Science*, 45(11):1570–1578.

[Knuth and Moore, 1975] Knuth, D. and Moore, R. (1975). An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326.

[Kocsis and Szepesvari, 2006a] Kocsis, L. and Szepesvari, C. (2006a). Bandit based Monte-Carlo planning. In *15th European Conference on Machine Learning (ECML)*, pages 282–293.

[Kocsis and Szepesvari, 2006b] Kocsis, L. and Szepesvari, C. (2006b). Discounted-UCB. In *2nd Pascal-Challenge Workshop*.

[Korf, 1985] Korf, R. E. (1985). Depth-first iterative-deepening: an optimal admissible tree search. *Artif. Intell.*, 27(1):97–109.

[Kotthoff, 2012] Kotthoff, L. (2012). Algorithm selection for combinatorial search problems: A survey. *CoRR*, abs/1210.7959.

[Koulouriotis and Xanthopoulos, 2008] Koulouriotis, D. and Xanthopoulos, A. (2008). Reinforcement learning and evolutionary algorithms for non-stationary multi-armed bandit problems. *Applied Mathematics and Computation*, 196(2):913–922.

[Kozak, 2004] Kozak, M. (2004). Optimal stratification using random search method in agricultural surveys. *Statistics in Transition*, 6(5):797–806.

[Krink et al., 2004] Krink, T., Filipic, B., Fogel, G., and Thomsen, R. (2004). Noisy optimization problems - a particular challenge for differential evolution? In *In Proceedings of 2004 Congress on Evolutionary Computation*, pages 332–339. IEEE Press.

[Lacca et al., 2012] Lacca, G., Neri, F., and Mininno, E. (2012). Noise analysis compact differential evolution. *Intern. J. Syst. Sci.*, 43(7):1248–1267.

[Lai, 2015] Lai, M. (2015). Giraffe: Using deep reinforcement learning to play chess. Master's thesis, Imperial College London, London.

[Lai and Robbins, 1985] Lai, T. and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6:4–22.

[Lavallée and Hidiroglou, 1988] Lavallée, P. and Hidiroglou, M. (1988). On the stratification of skewed populations. *Survey Methodology*, 14(1):33–43.

[Lee et al., 2009] Lee, C.-S., Wang, M.-H., Chaslot, G., Hoock, J.-B., Rimmel, A., Teytaud, O., Tsai, S.-R., Hsu, S.-C., and Hong, T.-P. (2009). The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments. *IEEE Transactions on Computational Intelligence and AI in games*.

[Lew et al., 2013] Lew, D., Bird, L., Milligan, M., Speer, B., Wang, X., Carlini, E. M., Estanqueiro, A., Flynn, D., Gomez-Lazaro, E., Menemenlis, N., et al. (2013). Wind and solar curtailment. In *Int. Workshop on Large-Scale Integration of Wind Power Into Power Systems*, pages 1–9.

[Linderoth et al., 2006] Linderoth, J., Shapiro, A., and Wright, S. (2006). The empirical behavior of sampling methods for stochastic programming. *Annals OR*, 142(1):215–241.

[Liu et al., 2008] Liu, B., Zhang, X., and Ma, H. (2008). Hybrid differential evolution for noisy optimization. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 587–592.

[Liu and Lampinen, 2005] Liu, J. and Lampinen, J. (2005). A fuzzy adaptive differential evolution algorithm. *Soft Comput.*, 9(6):448–462.

[Liu et al., 2014] Liu, J., Saint-Pierre, D. L., Teytaud, O., et al. (2014). A mathematically derived number of resamplings for noisy optimization. In *Genetic and Evolutionary Computation Conference (GECCO 2014)*.

[Liu and Teytaud, 2014] Liu, J. and Teytaud, O. (2014). Meta online learning: Experiments on a unit commitment problem. In *22th European Symposium on Artificial Neural Networks, ESANN 2014, Bruges, Belgium, April 23-25, 2014*.

[Lucidi and Sciandrone, 2002] Lucidi, S. and Sciandrone, M. (2002). A derivative-free algorithm for bound constrained optimization. *Comp. Opt. and Appl.*, 21(2):119–142.

[Mascagni and Chi, 2004] Mascagni, M. and Chi, H. (2004). On the scrambled halton sequence. *Monte-Carlo Methods Appl.*, 10(3):435–442.

[Mnih et al., 2008] Mnih, V., Szepesvári, C., and Audibert, J.-Y. (2008). Empirical Bernstein stopping. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 672–679, New York, NY, USA. ACM.

[Morokoff, 1998] Morokoff, W. J. (1998). Generating quasi-random paths for stochastic processes. 40(4):765–788.

[Nagarajan et al., 2015] Nagarajan, V., Marcolino, L. S., and Tambe, M. (2015). Every team deserves a second chance: Identifying when things go wrong (student abstract version). In *29th Conference on Artificial Intelligence (AAAI 2015), Texas, USA*.

[Nalimov et al., 2000] Nalimov, E. V., Haworth, G. M., Heinz, E. A., et al. (2000). Space-efficient indexing of chess endgame tables. *ICGA Journal*, 23(3):148–162.

[Niederreiter, 1992] Niederreiter, H. (1992). *Random Number Generation and Quasi-Monte Carlo Methods*.

[Pedersen, 2010] Pedersen, M. E. H. (2010). *Tuning & simplifying heuristical optimization*. PhD thesis, University of Southampton.

[Pereira and Pinto, 1991] Pereira, M. V. F. and Pinto, L. M. V. G. (1991). Multi-stage stochastic optimization applied to energy planning. *Math. Program.*, 52(2):359–375.

[Pinson, 2013] Pinson, P. (2013). Renewable energy forecasts ought to be probabilistic. In *WIPFOR seminar*.

[Poaík and Klema, 2012] Poaík, P. and Klema, V. (2012). Jade, an adaptive differential evolution algorithm, benchmarked on the bbob noiseless testbed. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, pages 197–204. ACM.

[Powell, 2004] Powell, M. J. D. (2004). The newuoa software for unconstrained optimization with derivatives. Technical Report NA2004/08, Dept. of Applied Math. and Theoretical Physics.

[Powell, 2008] Powell, M. J. D. (2008). Developments of newuoa for minimization without derivatives. *IMA J Numer Anal*, pages drm047+.

[Price et al., 2005] Price, K. V., Storn, R. M., and Lampinen, J. A. (2005). Differential evolution a practical approach to global optimization.

[Price et al., 2006] Price, K. V., Storn, R. M., and Lampinen, J. A. (2006). *Differential Evolution - A Practical Approach to Global Optimization*. Natural Computing. Springer-Verlag. ISBN 540209506.

[Pulina and Tacchella, 2009] Pulina, L. and Tacchella, A. (2009). A self-adaptive multi-engine solver for quantified boolean formulas. *Constraints*, 14(1):80–116.

[Reuchlin, 1978] Reuchlin, M. (1978). Vicariant processes and individual differences. *Journal de Psychologie Normale et Pathologique*.

[Richards and Hart, 2006] Richards, D. and Hart, T. (2006). The alpha-beta heuristic (aim-030). *Massachusetts Institute of Technology. Retrieved on*, pages 12–21.

[Robbins, 1952] Robbins, H. (1952). Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 55.

[Robbins and Monro, 1951] Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics 23*, 22:400–407.

[Robson, 1983] Robson, J. M. (1983). The complexity of go. In *IFIP Congress*, pages 413–417.

[Rolet and Teytaud, 2009] Rolet, P. and Teytaud, O. (2009). Bandit-based estimation of distribution algorithms for noisy optimization: Rigorous runtime analysis. In *Proceedings of Lion4 (accepted); presented in TRSH 2009 in Birmingham*, pages 97–110.

[Rolet and Teytaud, 2010] Rolet, P. and Teytaud, O. (2010). Adaptive noisy optimization. In Di Chio, C., Cagnoni, S., Cotta, C., Ebner, M., Ekárt, A., Esparcia-Alcazar, A., Goh, C.-K., Merelo, J., Neri, F., Preuß, M., Togelius, J., and Yannakakis, G., editors, *Applications of Evolutionary Computation*, volume 6024 of *Lecture Notes in Computer Science*, pages 592–601. Springer Berlin Heidelberg.

[Ros, 2009] Ros, R. (2009). Benchmarking the NEWUOA on the BBOB-2009 Noisy Testbed. In *GECCO*, Montréal, Canada.

[Rosenthal, 1981] Rosenthal, R. (1981). Games of perfect information, predatory pricing, and the chain store. *Journal of Economic Theory*, 25:92–100.

[RTE-ft, 2008] RTE-ft (2008). Rte forecast team: Electricity consumption in France : Characteristics and forecast method.

[Russell and Wolfe, 2005] Russell, S. and Wolfe, J. (2005). Efficient Belief-State AND-OR Search, with Application to Kriegspiel. In *IJCAI*, pages 278–285.

[Saint-Pierre and Teytaud, 2014] Saint-Pierre, D. L. and Teytaud, O. (2014). Nash and the Bandit Approach for Adversarial Portfolios. In *CIG 2014 - Computational Intelligence in Games*, Computational Intelligence in Games, pages 1–7, Dortmund, Germany. IEEE, IEEE.

[Saisirirat et al., 2013] Saisirirat, P., Chollacoop, N., Tongroon, M., Laoonual, Y., and Pongthanaisawan, J. (2013). Scenario analysis of electric vehicle technology penetration in thailand: Comparisons of required electricity with power development plan and projections of fossil fuel and greenhouse gas reduction. *Energy Procedia*, 34(0):459 – 470. 10th Eco-Energy and Materials Science and Engineering Symposium.

[Sakrison, 1964] Sakrison, D. (1964). A continuous Kiefer-Wolfowitz procedure for random process. *Ann. Math. Statist.*, 35:590–599.

[Samulowitz and Memisevic, 2007] Samulowitz, H. and Memisevic, R. (2007). Learning to solve qbf. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, pages 255–260. AAAI.

[Savage, 1951] Savage, L. J. (1951). The theory of statistical decision. *Journal of the American Statistical Association*, 46(253):55–67.

[Schwartz, 1996] Schwartz, P. (1996). *The Art of the Long View: Paths to Strategic Insight for Yourself and Your Company*. Random House.

[Sehnke et al., 2010] Sehnke, F., Osendorfer, C., Rückstiess, T., Graves, A., Peters, J., and Schmidhuber, J. (2010). Parameter-exploring policy gradients. *Neural Networks*, (23(4)):551–559. Intelligent Autonomous Systems.

[Sethi, 1963] Sethi, V. (1963). A note on optimum stratification of populations for estimating the population means. *Australian Journal of Statistics*, 5(1):20–33.

[Shahryar et al., 2006] Shahryar, R., Hamid R., T., and Magdy M. A., S. (2006). Opposition-based differential evolution for optimization of noisy problems. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1865–1872.

[Shahryar et al., 2008] Shahryar, R., Hamid R., T., and Magdy M. A., S. (2008). Opposition-based differential evolution. In Chakraborty, U., editor, *Advances in Differential Evolution*, volume 143 of *Studies in Computational Intelligence*, pages 155–171. Springer Berlin Heidelberg.

[Shamir, 2013] Shamir, O. (2013). On the complexity of bandit and derivative-free stochastic convex optimization. In *COLT 2013 - The 26th Annual Conference on Learning Theory, June 12-14, 2013, Princeton University, NJ, USA*, pages 3–24.

[Shanno, 1970] Shanno, D. F. (1970). Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24:647–656.

[Shannon, 1988] Shannon, C. E. (1988). Computer chess compendium. chapter Programming a computer for playing chess, pages 2–13. Springer-Verlag New York, Inc., New York, NY, USA.

[Shapire et al., 1997] Shapire, R., Freund, Y., P.Bartlett, and Lee, W. (1997). Boosting the margin: a new explanation for the effectiveness of voting methods. In *Proceedings of the 14^{th} International Conference on Machine Learning*, pages 322–330. Morgan Kaufmann.

[Shapiro et al., 2013] Shapiro, A., Tekaya, W., da Costa, J. P., and Soares, M. P. (2013). Risk neutral and risk averse stochastic dual

dynamic programming method. *European Journal of Operational Research*, 224(2):375–391.

[Silver and Tesauro, 2009] Silver, D. and Tesauro, G. (2009). Monte-carlo simulation balancing. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 945–952, New York, NY, USA. ACM.

[Siqueira et al., 2006] Siqueira, T., Zambelli, M., Cicogna, M., Andrade, M., and Soares, S. (2006). Stochastic dynamic programming for long term hydrothermal scheduling considering different streamflow models. In *Probabilistic Methods Applied to Power Systems, 2006. PMAPS 2006. International Conference on*, pages 1–6. IEEE.

[Spall, 2000] Spall, J. (2000). Adaptive stochastic approximation by the simultaneous perturbation method. *Automatic Control, IEEE Transactions on*, 45(10):1839–1853.

[Spall, 2009] Spall, J. (2009). Feedback and weighting mechanisms for improving jacobian estimates in the adaptive simultaneous perturbation algorithm. *Automatic Control, IEEE Transactions on*, 54(6):1216–1229.

[Spall, 2003] Spall, J. C. (2003). *Introduction to stochastic search and optimization: Estimation, simulation, and control*. John Wiley and Sons.

[St-Pierre and Liu, 2014] St-Pierre, D. L. and Liu, J. (2014). Differential Evolution Algorithm Applied to Non-Stationary Bandit Problem. In *2014 IEEE Congress on Evolutionary Computation (IEEE CEC 2014)*, Beijing, China.

[Stoltz et al., 2011] Stoltz, G., Bubeck, S., and Munos, R. (2011). Pure exploration in finitely-armed and continuous-armed bandits. *Theoretical Computer Science*, 412(19):1832–1852.

[Storn, 1996] Storn, R. (1996). On the usage of differential evolution for function optimization. In *Fuzzy Information Processing Society, 1996. NAFIPS. 1996 Biennial Conference of the North American*, pages 519–523. IEEE.

[Storn and Price, 1997] Storn, R. and Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359.

[Strens et al., 2002] Strens, M., Lx, H. G., Moore, A., Brodley, E., and Danyluk, A. (2002). Policy search using paired comparisons. *Journal of Machine Learning Research*, 3:921–950.

[Strens and Moore, 2001] Strens, M. and Moore, A. (2001). Direct policy search using paired statistical tests. In *Proceedings of the 18th International Conference on Machine Learning*, pages 545–552. Morgan Kaufmann, San Francisco, CA.

[Suganthan et al., 2005] Suganthan, P. N., Hansen, N., Liang, J. J., Deb, K., Chen, Y. P., Auger, A., and Tiwari, S. (2005). Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. Technical report, Nanyang Technological University, Singapore.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction.* MIT Press., Cambridge, MA.

[Takagi and Pallez, 2009] Takagi, H. and Pallez, D. (2009). Paired comparison-based interactive differential evolution. In *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pages 475–480. IEEE.

[Tesauro and Galperin, 1996] Tesauro, G. and Galperin, G. R. (1996). On-line policy improvement using monte-carlo search. In *NIPS*, volume 96, pages 1068–1074.

[Teytaud and Decock, 2013] Teytaud, O. and Decock, J. (2013). Noisy Optimization Complexity. In *FOGA - Foundations of Genetic Algorithms XII - 2013*, Adelaide, Australie.

[Teytaud and Flory, 2011] Teytaud, O. and Flory, S. (2011). Upper confidence trees with short term partial information. In *Applications of Evolutionary Computation*, pages 153–162. Springer.

[Teytaud et al., 2014] Teytaud, O., Saint-Pierre, D. L., Ruette, S., Liu, J., and Auger, D. (2014). Sparse binary zero-sum games. In *Asian Conference on Machine Learning*, volume 29, page 16.

[Tvrdık, 2006] Tvrdık, J. (2006). Competitive differential evolution. In *MENDEL 2006, 12th International Conference on Soft Computing*, pages 7–12. Brno: University of Technology.

[Uiterwijk, 2013] Uiterwijk, J. W. H. M. (2013). Perfectly solving domineering boards. In Cazenave, T., Winands, M. H. M., and Iida, H., editors, *Computer Games - Workshop on Computer Games, CGW 2013, Held in Conjunction with the 23rd International Conference on Artificial Intelligence, IJCAI 2013, Beijing, China, August 3, 2013, Revised Selected Papers*, volume 408 of *Communications in Computer and Information Science*, pages 97–121. Springer.

[Utgoff, 1988] Utgoff, P. E. (1988). Perceptron trees: A case study in hybrid concept representations. In *National Conference on Artificial Intelligence*, pages 601–606.

[v. Neumann, 1928] v. Neumann, J. (1928). Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320.

[Vapnik, 1995] Vapnik, V. N. (1995). *The Nature of Statistical Learning*. Springer Verlag.

[Vassena et al., 2003] Vassena, S., Mack, P., Rousseaux, P., Druet, C., and Wehenkel, L. (2003). A probabilistic approach to power system network planning under uncertainties. In *IEEE Bologna Power Tech Conference Proceedings*.

[Vassilevska et al., 2006] Vassilevska, V., Williams, R., and Woo, S. L. M. (2006). Confronting hardness using a hybrid approach. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1–10. ACM.

[Villemonteix et al., 2008] Villemonteix, J., Vazquez, E., and Walter, E. (2008). An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, page 26 pages.

[von Stengel, 2002] von Stengel, B. (2002). Computing equilibria for two-person games. In Aumann, R. and Hart, S., editors, *Handbook of Game Theory*, volume 3, pages 1723 – 1759. Elsevier, Amsterdam.

[Wald, 1939] Wald, A. (1939). Contributions to the theory of statistical estimation and testing hypotheses. *Ann. Math. Statist.*, 10(4):299–326.

[Wang and Hickernell, 2000] Wang, X. and Hickernell, F. (2000). Randomized halton sequences. *Math. Comput. Modelling*, 32:887–899.

[Weinstein and Littman, 2012] Weinstein, A. and Littman, M. L. (2012). Bandit-based planning and learning in continuous-action markov decision processes. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*.

[Wolpert and Macready, 1997] Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.

[Xu et al., 2008] Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2008). Satzilla: Portfolio-based algorithm selection for sat. *J. Artif. Int. Res.*, 32(1):565–606.

[Xu et al., 2011] Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). 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)*.

[Yang et al., 2010] Yang, M., Guan, J., Cai, Z., and Wang, L. (2010). Self-adapting differential evolution algorithm with chaos random for global numerical optimization. In Cai, Z., Hu, C., Kang, Z., and Liu, Y., editors, *Advances in Computation and Intelligence*, volume 6382 of *Lecture Notes in Computer Science*, pages 112–122. Springer Berlin Heidelberg.

[Yang et al., 2005] Yang, X., Birkfellner, W., and Niederer, P. (2005). Optimized 2d/3d medical image registration using the estimation of multivariate normal algorithm (EMNA). In *Biomedical engineering*.

[Zabinsky, 2009] Zabinsky, Z. B. (2009). Random search algorithms. *Wiley Encyclopedia of Operations Research and Management Science*.