Hybridization of dynamic optimization methodologies

Jérémie Decock

Université Paris-Sud 11
École Doctorale d'Informatique, ED 427
Inria Saclay / LRI

Ph.D. Thesis in Computer Science

Hybridization of dynamic optimization methodologies

By
Jérémie DECOCK

advised by
Olivier TEYTAUD

Presented and publicly defended on November 28 2014
in Orsay, France

With the following jury :

Pierre-Olivier Malaterre,	HDR, CR Irstea Montpellier, France	(Reviewer)
Liva Ralaivola,	Professor, University of Aix-Marseille, France	(Reviewer)
Yacine Chitour,	Professor, University of Paris-Sud 11, France	(Examiner)
Damien Ernst,	Associate Professor, University of Liège, Belgium	(Examiner)
Pierre Martinon,	CR Inria Saclay, CMAP Polytechnique, France	(Examiner)
Marc Schoenauer,	DR Inria Saclay, LRI/TAO, France	(Thesis Advisor)
Olivier Teytaud,	HDR, CR Inria Saclay, LRI/TAO, France	(Thesis Advisor)

Reviewers :

Pierre-Olivier Malaterre,	HDR, CR Irstea Montpellier, France
Liva Ralaivola,	Professor, University of Aix-Marseille, France

Inria Saclay - Île-de-France – TAO Project Team
Bât 660 Claude Shannon, Université Paris Sud, Rue Noetzlin, 91190 Gif-sur-Yvette, FRANCE

UNIVERSITÉ PARIS SUD
Comprendre le monde, construire l'avenir®

Inria
INVENTORS FOR THE DIGITAL WORLD

Université Paris-Sud 11
École Doctorale d'Informatique, ED 427
Inria Saclay / LRI

Thèse de doctorat en Informatique

L'hybridation de Méthodes d'optimisation dynamique

Par

Jérémie DECOCK

sous la direction
d'Olivier TEYTAUD

Présentée et soutenue publiquement le 28 Novembre 2014
à Orsay, France

Devant le jury ci-dessous :

Pierre-Olivier Malaterre,	HDR, CR Irstea Montpellier, France	(Rapporteur)
Liva Ralaivola,	Professor, University of Aix-Marseille, France	(Rapporteur)
Yacine Chitour,	Professor, University of Paris-Sud 11, France	(Examinateur)
Damien Ernst,	Associate Professor, University of Liège, Belgium	(Examinateur)
Pierre Martinon,	CR Inria Saclay, CMAP Polytechnique, France	(Examinateur)
Marc Schoenauer,	DR Inria Saclay, LRI/TAO, France	(Directeur de Thèse)
Olivier Teytaud,	HDR, CR Inria Saclay, LRI/TAO, France	(Directeur de Thèse)

Rapporteurs :

Pierre-Olivier Malaterre,	HDR, CR Irstea Montpellier, France
Liva Ralaivola,	Professor, University of Aix-Marseille, France

Inria Saclay - Île-de-France – TAO Project Team
Bât 660 Claude Shannon, Université Paris Sud, Rue Noetzlin, 91190 Gif-sur-Yvette, FRANCE

Abstract

This thesis is dedicated to sequential decision making (also known as multistage optimization) in uncertain complex environments. Studied algorithms are essentially applied to electricity production ("Unit Commitment" problems) and energy stock management (hydropower), in front of stochastic demand and water inflows. The manuscript is divided in 7 chapters and 4 parts: Part I, "General Introduction", Part II, "Background Review", Part III, "Contributions" and Part IV, "General Conclusion".

The first chapter (Part I) introduces the context and motivation of our work, namely energy stock management. "Unit Commitment" (UC) problems are a classical example of "Sequential Decision Making" problem (SDM) applied to energy stock management. They are the central application of our work and in this chapter we explain main challenges arising with them (e.g. stochasticity, constraints, curse of dimensionality, ...). Classical frameworks for SDM problems are also introduced and common mistakes arising with them are discussed. We also emphasize the consequences of these - too often neglected - mistakes and the importance of not underestimating their effects. Along this chapter, fundamental definitions commonly used with SDM problems are described. An overview of our main contributions concludes this first chapter.

The second chapter (Part II) is a background review of the most classical algorithms used to solve SDM problems. Since the applications we try to solve are stochastic, we there focus on resolution methods for stochastic problems. We begin our study with classical Dynamic Programming methods to solve "Markov Decision Processes" (a special kind of SDM problems with Markovian random processes). We then introduce "Direct Policy Search", a widely used method in the Reinforcement Learning community. A distinction is made between "Value Based" and "Policy Based" exploration methods.

The third chapter (Part II) extends the previous one by covering the most classical algorithms used to solve UC's subtleties. It contains a state of the art of algorithms commonly used for energy stock management, mainly "Model Predictive Control",

"Stochastic Dynamic Programming" and "Stochastic Dual Dynamic Programming". We briefly overview distinctive features and limitations of these methods.

The fourth chapter (Part III) presents our main contribution: a new algorithm named "Direct Value Search" (DVS), designed to solve large scale unit commitment problems. We describe how it outperforms classical methods presented in the third chapter. We show that DVS is an "anytime" algorithm (users immediately get approximate results) which can handle large state spaces and large action spaces with non convexity constraints, and without assumption on the random process. Moreover, we explain how DVS can reduce modelling errors and can tackle challenges described in the first chapter, working on the "real" detailed problem without "cast" into a simplified model.

Noisy optimisation is a key component of DVS algorithm; the fifth chapter (Part III) is dedicated to it. In this chapter, some theoretical convergence rate are studied and new convergence bounds are proved - under some assumptions and for given families of objective functions. Some variance reduction techniques aimed at improving the convergence rate of graybox noisy optimization problems are studied too in the last part of this chapter.

Chapter sixth (Part III) is devoted to non-quasi-convex optimization. We prove that a variant of evolution strategy can reach a log-linear convergence rate with non-quasi-convex objective functions.

Finally, the seventh chapter (Part IV) concludes and suggests some directions for future work.

Résumé en Français

Dans ce manuscrit de thèse, mes travaux portent sur la comparaison et la combinaison de méthodes pour la prise de décision séquentielle (plusieurs étapes de décision corrélées) dans des environnements complexes et incertains. Ces travaux ont été exercés dans le cadre d'un partenariat (« Inria Innovation Lab ») entre l'équipe TAO (LRI-Inria) et l'entreprise Artelys, spécialisée dans l'optimisation de systèmes complexes. Les méthodes mises au point sont essentiellement appliquées à des problèmes de gestion et de production d'électricité (« unit commitment ») tels que l'optimisation de la gestion des stocks d'énergie (réservoirs d'eau dans les centrales hydrauliques, etc.) dans un parc de production afin d'anticiper au mieux la fluctuation de la consommation des clients.

Le manuscrit comporte 7 chapitres regroupés en 4 parties : Partie I, « Introduction générale », Partie II, « État de l'art », Partie III, « Contributions » et Partie IV, « Conclusion générale ».

Le premier chapitre (Partie I) introduit le contexte et les motivations de mes travaux, à savoir la résolution de problèmes d' « Unit commitment », c'est à dire l'optimisation des stratégies de gestion de stocks d'énergie dans les parcs de production d'électricité. Les particularités et les difficultés sous-jacentes à ces problèmes sont décrites ainsi que le cadre de travail et les notations utilisées dans la suite du manuscrit.

Le second chapitre (Partie II) dresse un état de l'art des méthodes les plus classiques utilisées pour la résolution de problèmes de prise de décision séquentielle dans des environnements incertains. Ce chapitre introduit des méthodes générales (i.e. non spécifiques à notre domaine d'application) et des concepts nécessaires à la bonne compréhension des chapitres suivants (notamment le chapitre 4). Les méthodes de programmation dynamique classiques (algorithmes d'itération de la valeur et d'itération de la politique) et les méthodes de recherche de politique directe y sont présentées.

Le 3e chapitre (Partie II) prolonge le précédent en dressant un état de l'art des principales méthodes d'optimisation spécifiquement adaptées à la gestion des parcs de production d'énergie et à leurs subtilités. Ce chapitre présente entre autre les

méthodes MPC (Model Predictive Control), SDP (Stochastic Dynamic Programming) et SDDP (Stochastic Dual Dynamic Programming) avec pour chacune leurs particularités, leurs avantages et leurs limites. Ce chapitre complète le précédent en introduisant d'autres concepts nécessaires à la bonne compréhension de la suite du manuscrit.

Le 4e chapitre (Partie III) contient la principale contribution de ma thèse : un nouvel algorithme appelé « Direct Value Search » (DVS) créé pour résoudre des problèmes de prise de décision séquentielle de grande échelle en milieu incertain avec une application directe aux problèmes d' « Unit commitment ». Ce chapitre décrit en quoi ce nouvel algorithme dépasse les méthodes classiques présentées dans le 3e chapitre. Cet algorithme innove notamment par sa capacité à traiter des grands espaces d'actions contraints dans un cadre non-linéaire, avec un grand nombre de variables d'état et sans hypothèse particulière quant aux aléas du système optimisé (c'est à dire applicable sur des problèmes où les aléas ne sont pas nécessairement Markovien).

Le 5e chapitre (Partie III) est consacré à un concept clé de DVS : l'optimisation bruitée. Ce chapitre expose une nouvelle borne théorique sur la vitesse de convergence des algorithmes d'optimisation appliqués à des problèmes bruités vérifiant certaines hypothèses données. Des méthodes de réduction de variance sont également étudiées et appliquées à DVS pour accélérer sensiblement sa vitesse de convergence.

Le 6e chapitre (Partie III) décrit un résultat mathématique sur la vitesse de convergence linéaire d'un algorithme évolutionnaire appliqué à une famille de fonctions non quasi-convexes. Dans ce chapitres, il est prouvé que sous certaines hypothèses peu restrictives sur la famille de fonctions considérée, l'algorithme présenté atteint une vitesse de convergence linéaire.

Le 7e chapitre (Partie IV) conclut ce manuscrit en résumant mes contributions et en dressant quelques pistes de recherche intéressantes à explorer.

Acknowledgements

First and foremost I want to thank my advisors, Olivier and Marc, for a rich and friendly scientific environment they have created.

I'm glad to have worked with so highly competent colleagues; many thanks to all the member of the INRIA-TAO team and to Artelys' employees.

Once again, I want to thanks a lot Olivier for his consideration, his kindness and his great patience. I am very grateful for everything he has taught me over the last 3 years. I am also very grateful I had the pleasure to work *with* him, and not *for* him.

A lot of thanks to all others PhD students too for the great atmosphere and for the wonderful work environment they all contributed to build. I also thank Marie Carol and Olga for their outstanding work as the secretary of the team.

A lot of thanks to the NUTN-OASE team, for their warm welcome and their help during my stay in Taiwan.

Also a lot of thanks to Olivier Teytaud, Liva Ralaivola and Michele Sebag for the proofreading of this manuscript. I also thank Yacine Chitour, Damien Ernst, Pierre-Olivier Malaterre, Pierre Martinon, Liva Ralaivola and Marc Schoenauer for being members of the comity.

I wouldn't get results without Grid5000 and Inria's SIC staff, a lot of thanks to them.

Also, a lot of thanks to my family and my friends for their presence and their support. And finally, a huge thank to my wife and my daughter for giving me the courage to move forward.

To Laura, the love of my life and Léna, my adored daughter.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

ADP Approximate Dynamic Programming

DP Dynamic Programming

DPS Direct Policy Search

DVS Direct Value Search

ES Evolution Strategy

CMA-ES Covariance Matrix Adaptation Evolution Strategy

SA-ES Self-Adaptive Evolution Strategy

GA Genetic Algorithm

LP Linear Programming

MDP Markov Decision Process

MILP Mixed Integer Linear Programming

MPC Model Predictive Control

QP Quadratic Programming

RL Reinforcement Learning

SDP Stochastic Dynamic Programming

SDDP Stochastic Dual Dynamic Programming

RBF Radial Basis Function

SVM Support Vector Machine

Part I

General Introduction

Chapter 1

Introduction

1.1 Thesis Outline

This first chapter introduces the context and motivation of our work, namely energy stock management. *Unit Commitment* problems are a classical example of *Sequential Decision Making* problem applied to energy stock management. They are the central application of our work and we will explain main challenges arising with them (e.g. stochasticity, constraints, curse of dimensionality, ...). In this chapter, classical frameworks for Sequential Decision Making problems will be introduced and common mistakes arising with them will be discussed. We will emphasize the consequences of these – too often neglected – mistakes and the importance of not underestimating their effects. Along this chapter, fundamental definitions commonly used with Sequential Decision Making problems will be described. An overview of our main contributions will conclude this first chapter.

The second chapter is a background review of the most classical algorithms used to solve Sequential Decision Making problems. Since the applications we try to solve are stochastic, we there focus on resolution methods for stochastic problems. We begin our study with classical *Dynamic Programming* methods to solve *Markov Decision Processes* (a special kind of Sequential Decision Making problems with Markovian random processes). We then introduce *Direct Policy Search*, a widely used method in the Reinforcement Learning community. A distinction will be made between *value based* and *policy based* exploration methods.

The third chapter extends the previous one by covering the most classical algorithms used to solve Unit Commitment's subtleties. It contains a state of the art of algorithms commonly used for energy stock management, mainly Model Predictive Control, Stochastic Dynamic Programming and Stochastic Dual Dynamic Program-

ming. We will briefly overview distinctive features and limitations of these methods.

The fourth chapter presents our main contribution: a new algorithm named *Direct Value Search*, designed to solve large scale unit commitment problems. We will describe how it outperforms classical methods presented in the third chapter. We will show that Direct Value Search is an *anytime* algorithm (users immediately get approximate results) which can handle large state spaces and large action spaces with non convexity constraints, and without assumption on the random process. Moreover, we will explain how Direct Value Search can reduce modelling errors and can tackle challenges described in the first chapter, working on the "real" detailed problem without "cast" into a simplified model.

Noisy optimisation is a key component of Direct Value Search algorithm; the fifth chapter is dedicated to it. In this chapter, some theoretical convergence rate will be studied and new convergence bounds will be proved – under some assumptions and for given families of objective functions. Some variance reduction techniques aimed at improving the convergence rate of graybox noisy optimization problems will be studied too in the last part of this chapter.

Chapter sixth is devoted to non-quasi-convex optimization. We prove that a variant of evolution strategy can reach a log-linear convergence rate with non-quasi-convex objective functions.

Finally, the seventh chapter concludes and suggests some directions for future work.

1.2 Context and Motivation

Energy management problems have received a considerable attention in recent years. For instance, in France, the government recently undertook the *French Environment and Energy Management Agency* (ADEME) to work on several strategic and priority project on this thematic (as part of the "Investissement d'avenir" program[1]).

As explained in [1] and [2], among the likely reasons behind this rise of interest, we could mention the economic weight of energy market and the strategic and political importance of this sector for many countries (especially since 1970s energy crisis). Indeed, electricity is an essential consumer goods of which the global demand and production is rising sharply for some years[2]. Because of the increasing demand in newly industrialized countries, the global energy demand should grow by more than

[1] `http://investissement-avenir.gouvernement.fr/`

[2] For instance, China's electricity production increased from 621TWh to 4716TWh between 1990 and 2012 (respectively 5% and 21% of the world electricity production) as reported in [3]

one-third over the period to 2035 according to the *International Energy Agency* (IEA) [4]. As fossil fuels remain the main primary energy sources for the world electricity production nowadays [3], the foreseeable scarcity and the geographical concentration of these fossil fuels cause a growing rivalry between states to access energy.

Another explanation to this rise of interest for energy management is the sudden awareness of the global warming and other deterioration of the environment and global health caused by fossil fuels exploitation [5, 6]. As a result, new environmental constraints have appeared for energy producers. Kyoto Protocol[3] is probably the most famous one. Adopted in 1997 by 84 countries and joined by 191 states since, the Protocol's major feature is that it has mandatory targets on reducing greenhouse-gas[4] emissions for the world's leading economies which have accepted it. As a result, these constraints pushed the development of "green"[5] renewable and sustainable energies. Since hydroelectric is almost already developed at its full capacity in many countries and as there exists a noticeable pressure of public opinion against nuclear power plants (especially since the Fukushima nuclear disaster in Japan in 2011 [7, 8]), the global trend in many countries [9, 10] is to develop solar power plants and wind turbines. But these fast-growing energy sources make the exploitation strategy more difficult to define for energy producers. Indeed wind and solar energy production depend on weather conditions. Their production level is often difficult to predict and can vary a lot.

Another good reason to be preoccupied with energy management is the growing interconnection of electrical networks which increases the negative impact of potentials blackout (by the domino effect). Avoiding these blackout requires more attention and better investment and exploitation strategies by energy producers.

Finally, the worldwide industrial and economical competition prompts towards a cheap energy and therefore further optimized exploitation and investment strategies.

To sum up, electricity makers have to face new difficult problematics and the environmental, economical and political consequences of inappropriate choices are more and more serious. Within this context, prospective models are a very valuable decision support. They are commonly used to forecast and optimise the short and mid term electricity production likewise long-term investment strategies.

Our motivation in this thesis is to create new decision aids tools fitted for the latest political, economical, environmental and sociological constraints. Our goal is to help electricity managers to find better exploitation and investment strategies on

[3] See `http://unfccc.int/2860.php`

[4] Carbon dioxide (C02), Methane (CH4), Nitrous oxide (N20), Hydrofluorocarbons (HFCs), Perfluorocarbons (PFCs) and Sulphur hexafluoride (SF6)

[5] Greenhouse gas emission-free

different scenarios, with larger, more detailed and more realistic models (section 1.5).

1.3 Power Systems

The problems we try to solve in this thesis are mainly energy management problems and in particular *Unit Commitment* problems (see [11]).

There exist many others classic problems for energy management like *thermal economic dispatch*, *coordination of hydroelectric and thermal (and sometimes pumped-storage)*, *proper commitment*, *optimal economic maintenance scheduling*, ... We will not consider them in this manuscript as they are more specific and less important than Unit Commitment to achieve our goal. But for the interested readers, some of these problems are defined in [11].

1.3.1 Unit Commitment and Related Problems

Unit Commitment problems are *Sequential Decision Making* problems [6] (or *multi-stage optimization problems*) where an energy producer (the *decision maker*) has to meet a certain energy demand (unknown in advance). In order to satisfy this demand, the decision maker has several power plants like:

- hydroelectric power stations (dams),
- thermal plants (coal power plants, fossil-fuel power stations, gas turbine plants, nuclear plants, geothermal power plants, ...),
- solar power plants,
- wind turbines,
- ...

These power plants are composed of production units (also named generators). All these plants and production units have different properties, different constraints and different exploitation costs making them more or less appropriate in a given situation [11]. The goal for the decision maker is to find the optimal operating strategy (or *policy*) which satisfy consumers energy demand for a given period of time at the lowest production cost possible. To sum up, Unit Commitment is the art and science of choosing which power plants to switch on and off over time to satisfy consumers demand at the lowest cost. These choices are made considering the features and the

[6] Sequential Decision Making problems are described in section 1.4. Classical methods to solve these problems are studied in chapter 2 and 3.

cost of each power plants, but also endogenous factors (e.g. the *state* of power plants, the energy stocks in dams, ...), varied constraints (laws, energy transportation, ...) and uncertain exogenous factors with more or less limited forecast (e.g. consumers demand, weather, ...).

Usually, the following constraints have to be satisfied during the optimization process [12, 13, 14]:

- System power balance: demand, losses, exports.
- System reserve requirement: safety stocks required in case of technical incident.
- Unit initial production level: the level at which it is operating before the first time period of our schedule.
- Unit minimum and maximum generation level (in MW).
- Unit rate limits: generation level inertia.
- Unit minimum-up time: this is the minimum number of hours a generator must be on once turned on.
- Unit minimum-down time: this is the minimum number of hours a generator must be off once turned off.
- Unit start-up and shut-down ramps: this is the maximum amount that a generator can increase or decrease production in an hour.
- Unit status restrictions: must-run, fixed-MW, unavailable, available.
- Unit fuel availability (or for instance stock volume for hydro and wind for wind turbine).
- Plant crew constraints: the number of units that can start at the same time in a particular plant.
- Electricity transmission constraints (electricity transportation is constrained by the transmission network topology, transmission lines have limited capacities and are subject to losses).
- Various plant type and unit type specific constraints (fuel, coal, nuclear, hydro, wind turbine, ...). E.g. for fuel and coal thermal plants:
  - Unit flame stabilization fuel mix (dual fuels are used for flame stabilization when the units operates at low output levels, e.g. when start-up).
  - Unit dual or alternate fuel usage.
  - Unit start-up fuel consumption coefficient.
  - Unit fuel consumption formula.

- System environmental constraints (e.g. allowance of CO2 emissions)
- ...

Thus, generic Unit Commitment problems can be formulated as the following mathematical optimization problem:

$$\begin{aligned} \min \quad & \text{Operational Costs} \\ \text{s.t.} \quad & \text{constraints} \end{aligned} \tag{1.1}$$

Operational costs to minimize generally consist of a sum of following terms (formulations are taken from [12, 13, 14]):

- Energy production costs: for instance Eq. (1.2) could be applied in the case of thermal plants;
- Start-up costs: expressed as a function of the number of hours the unit has been down (exponential when cooling and linear when banking) (Eq. (1.3));
- Shut-down costs: a fixed price for each unit per shut-down;
- Maintenance costs: calculated using base maintenance cost and incremental maintenance cost (Eq. (1.4))

$$FC_{i,t}(P_{i,t}) = F_{i,t}.H_{i,t}(P_{i,t}) \tag{1.2}$$

where

$FC_{i,t}(P_{i,t})$	is the fuel cost (€/h)
$P_{i,t}$	is the power level (MW)
$F_{i,t}$	is the fuel cost (€/MBTu)
$H_{i,t}(P_{i,t})$	is the heat rate curve (MBTu/h)

for unit i at time step t with BTu the British Thermal Unit (1 BTu $\simeq$ 1055 Joules).

$$ST_i = TS_iF_i + \left(1 - \exp^{-D_i/AS_i}\right) BS_iF_i + MS_i \tag{1.3}$$

where

ST_i	is the startup cost
TS_i	is the turbine startup energy (MBTu)
D_i	is the number of hours down
AS_i	is the boiler cool-down coefficient
BS_i	is the boiler startup energy (MBTu)
MS_i	is the startup maintenance cost

$$MC_i(P_i) = BM_i + IM_iP_i \tag{1.4}$$

where

$MC_i(P_i)$	is the maintenance cost
BM_i	is the base maintenance cost
IM_i	is the incremental maintenance cost

Decision variables (free variables) are generally [13, 14]:

- the list of units to be on and off;
- the power generation for each unit;
- the reserve margins;
- plant or unit specific variables.

Depending on the type of plants, the model choices and the resolution horizon, the following random variables may be considered:

- the electricity demand;
- the electricity production market price (fuel price, ...);
- the weather, which influences the electricity demand and the electricity production (especially for hydroelectric power stations, solar plants and wind turbines).

Depending on the type of power plants considered and the model choices, optimization is performed for very different horizons, going from the short term (some days) to the long-term (several years) and using very different time step resolutions (from hours to months).

The complexity of the problem also depends on the number of units optimized (up to hundreds units for some large scale models like the one used in [15]).

Our motivation is to design numerical methods (algorithms) to help the decision maker to choose the best possible decisions. As we saw in section 1.2, these problems are very important and bad choices may have serious consequences. We will see in section 1.3.2 that these problems can also be very challenging, and are commonly used as benchmark in the *Operations Research* (OR) and *Reinforcement Learning* (RL) community (as we do here).

1.3.2 Challenges

Power systems problems like Unit Commitment have existed for several decades. They are complex problems. Resolution methods exist but in most cases, with realistic models, the optimal solutions cannot be reached due to the excessive computation time requirement [16]. As a result, these resolution methods aim at finding efficiently a nearly optimal solution. As any slight improvement on proposed solutions may save millions of euros for decision makers [11], there is still a relevance for research on power systems algorithms. Moreover, new challenges recently appeared like those described in section 1.2: the recent and profound mutation on the operation of electric systems due to deregulation, the increase in the interconnections and the distribution networks size, the irregular and hardly predictable production with renewable energies, ... With these new needs, some power systems problems became much more challenging and classical algorithms are unsuited. In this section, main (old and new) power systems challenges are described.

State Space and Action Space Dimensions

Unit Commitment is a large-scale optimisation problem, often dealing with hundreds of generating units in a region[7], making it difficult to find the optimal solution in an acceptable amount of time [15]. Thus, the number of state variables, i.e. variables necessary for describing the current state of the system, can be large. The physical system has already tenths of variables for describing water levels in dams, just for France alone. A detailed model must include the variables accounting for the current power output of thermal plants. Taking into account costs associated to humans operating the machine introduces additional variables. The variables corresponding to the demand forecasts and wind/solar power should also be taken into account. The number of state variables also increases with the number of time steps.

As far as that goes, the action space, i.e. the set of possible decisions, is huge. As for state space, its size is proportional to the number of generators and the number of time steps. Many methods aggregate several time steps into an operational horizon. This means that the number of time steps is reduced, and it helps a lot for ensuring that constraints are satisfied, but the action space becomes bigger - the number of actions is multiplied by the number of aggregated time steps.

Some of these state and decision variables are integers or binaries (e.g. the on/off status of generators). Thus, the problem to be optimized becomes *combinatorial* and much more difficult to solve. This challenge is often addressed by simplifying the

[7] For instance, in France, there were 630 production units in 2011 [17, 18].

model using *relaxation methods* but this approximation yields suboptimal solutions which may cost millions of euros per year to decision makers[8].

Stochasticity

Unit Commitment are stochastic problems that are often solved like deterministic problems with a predicted scenario in order to make it simpler to solve. But ignoring uncertainties may lead to hazardous results as exposed in [20]. Unit Commitment have different sources of randomness [2].

Variations in electricity demand. The demand curve is made of daily, weekly and seasonal cycles (roughly corresponding to the global activity of consumers). But this demand is very sensitive to weather conditions and unpredicted weather changes may cause unpredicted demand changes. For example, in France, a drop in temperature of 1°C during winter usually leads to an increased consumption of about 1400 MW. Similarly, the difference in cloud cover can cause variations of about 6000 MW [21].

Variations in electricity production. The electricity production can vary more or less for various reasons. Firstly, technical hitch in production units or transmission network can cause a sudden and unexpected variation of production (for instance a failure in a generator or a power line breakdown). Then, some production methods, like most of those based on renewable energies, are naturally irregular (and hardly predictable). As the generation mix shifts to include more and more of these intermittent renewable energy, the production level is more and more affected by variations [20]. This variability involves different time constants [22] depending on the type of unit:

- multiannual variations for biomass;
- seasonal and annual variations for hydroelectricity;
- hourly variations (or less) for wind turbines and solar power plants.

The latter is the most problematic. Indeed, these variations are less regular and more difficult to forecast [20]. Moreover, significant variations in the production level may occur for smaller periods of time than the model's time resolution (often a hourly resolution) and thus are ignored in simulations.

[8] see the case of PJM [19], a regional transmission organization (RTO) in USA, whose annual bid costs savings were estimated at $60 million since they switched to *Mixed Integer Programming* solvers.

Decision makers usually deal with the variations in electricity production by increasing their reserve capacity and use it whenever it is necessary. This is rather safe but economically inefficient. Another approach is to solve Unit Commitment problems using *robust decision making* methods like *stochastic programming* [23]. A large number of scenarios is necessary to have confidence in the approach, and this significantly increases the complexity of the problem.

Variations in interconnected networks. Regional or national electric networks are more and more interconnected (for economical and technical reasons). For instance, European countries are connected to each others. As a result, any big technical hitch in a region can propagate to the other and cause cascading failure by domino effect. For instance, on September 28, 2003, an incident on a transmission line in Switzerland ensued an almost total blackout in Italy (see Fig.1.1).

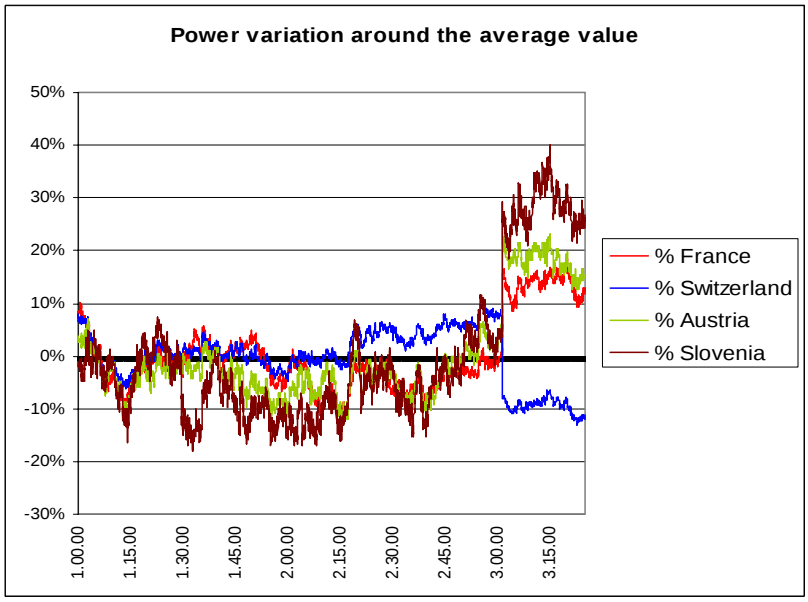

Figure 1.1: Consequences of the cascading failure on September 28th 2003 between 1:00 AM and 3:30 AM. The power went off at about 3:00 AM in Italy. This figure shows the power variation around the average level in interconnections with each neighboring country of Italy. These countries had to face a sudden and unexpected increase in the power demand. Source: *Autorita per l'Energia Elettrica e il Gas* and *Commission de Régulation de l'Electricité* [24]

.

Constraints and objective function

As we saw in section 1.3.1, many different constraints apply in Unit Commitment problems. Some of these constraints may be non-linear (with respect to decision variables) or non-convex[9]. Similarly, the objective function may be non-linear or non-convex. For instance:

- the input-output curve of steam turbine generators is non-linear [11] (see Fig. 1.2);
- the input-output curve of hydroelectric unit is non-linear [11];
- the input-output curve of steam turbine generators with multiple admission valves is non-convex [11] (see Fig. 1.3);
- the input-output curve of *combined cycle units* is non-convex [11];
- some hydroelectric units are non-convex with respect to the stock level (as the output efficiency can depends on the water drop height on turbines).

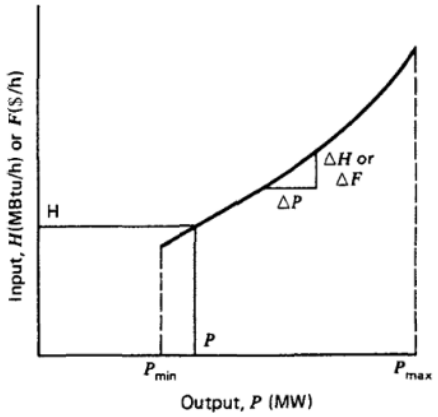

Figure 1.2: Input-output curve of a typical steam turbine generator (coal, fuel, ...). with H = British thermal unit (Btu) per hour heat input to the unit (or MBtu/h) and P = Output power (in MW). Source: *Power Generation, Operation and Control* [11].

[9] Conventionally, the term *convex* (or *non-convex*) objective function implicitly means *convex* (or *non-convex*) function ***within minimization problems*** and *concave* (or *non-concave*) function ***within maximization problems***.

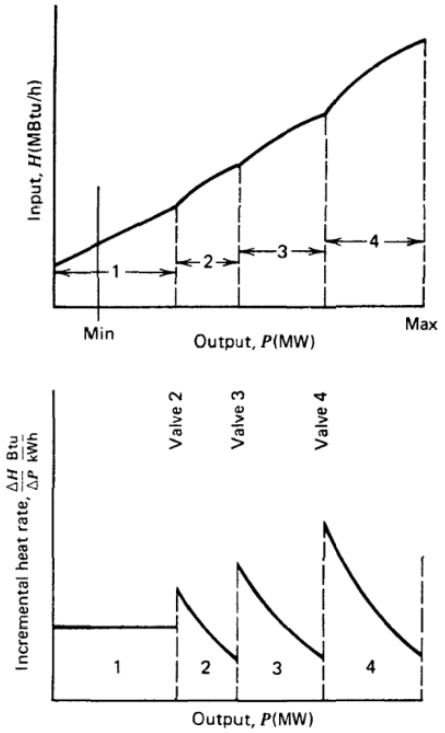

Figure 1.3: Characteristics of a steam turbine generator with four steam admission valves, with H = British thermal unit (Btu) per hour heat input to the unit (or MBtu/h) and P = Output power (in MW). Source: *Power Generation, Operation and Control* [11].

These non-linear and non-convex constraints and objective functions make the problem even more difficult to solve. A classical solution adopted by most decision makers is to transform the non-linear problem into a linear one or the non-convex problem into a convex one. But we will see in section 1.5 that good solutions for the simplified model can be bad solutions for the actual one and thus, directly solving the non transformed system may be more desirable.

1.3.3 Investment Problems

So far, we have considered how to optimize the exploitation of power plants. An interesting related problem is the investment problem. It attempts to answer the following question: when, where and how much should we invest on new capacities

(plants, transmission lines, ...) in order to optimize profits, keep the system in a reasonable condition and adapt it to the future changes of its environment in the long run? In the case of long-term investment studies (several decades), there are non-stochastic uncertainties, which are usually handled by considering several scenarios. These uncertainties are typically:

- political constraints: nuclear power or not, real CO_2 penalization or not, massive renewable sources or not, geopolitical events (for instance the Russian gas), ...
- technological breakthroughs;
- climate change uncertainty (green gas effect, ...);
- uncertainties about the evolution of demand and usage (expansion of electric cars, ...);
- prices (oil, gas) which highly depend on the international context;

Moreover, the investment part in Unit Commitment strongly increases the state space and action space dimensions, as state variables should describe the current financial energy portfolio, including contracts at the scale of days, weeks, months, years.

These long-term investment studies are extremely important, as they involve investments in hundreds of billions of euros – and they have to be made with huge uncertainties. We will, in this work, develop algorithms suitable for the former challenges but the non-stochastic uncertainties are not considered.

1.4 Modelling Sequential Decision Making Problems

Sequential Decision Making is a class of optimization problems (also named *multi-stage optimization problems* or *dynamic programs* in [25]) where an *agent* (also named *controller*) interacts with a given *environment* (also named *system*) over time.

In Sequential Decision Making, the time is discretized and interactions between the agent and the environment occur for a certain number of *time steps* (or *stages*). The environment starts in a given initial *state* and the agent can modify this state by taking *actions* (or *decisions*) on the environment. The result of successive actions is correlated as actions taken at a given time step influence future actions. Thus,

Sequential Decision Making problems are more complex than a series of multiple independent decision making problems since the agent has to consider the long-term effect of his decisions. At each time step, the actions of the agent alter the state of the environment according to a stochastic *transition model*. To choose among the possible actions, the agent uses information given by the environment. Each time an action is executed, a resulting reward is transmitted to the agent. This reward usually depends on the new state of the environment but it sometimes may also depend on the former state and the executed action. The goal for the agent is to take actions such that its global *utility* is maximized. This utility is either the sum of all rewards gained over time or a discounted sum of these rewards.

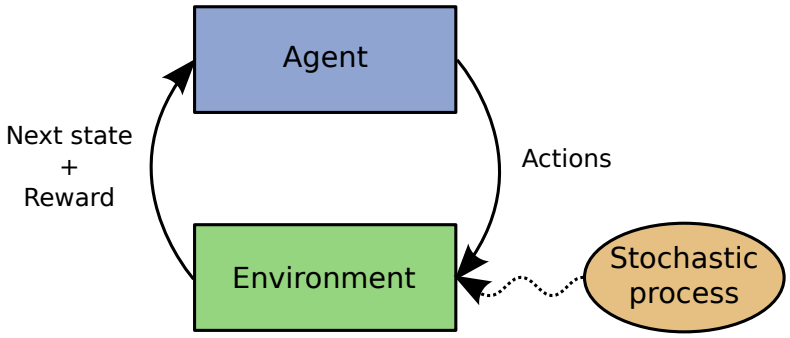

Figure 1.4: Sequential Decision Making Problems

This problem is depicted in Fig.1.4: starting from a given initial state, the environment sends information about its state to the agent; the agent uses this information to alter the environment with an action; this action updates the environment's state according to a (stochastic) *transition model*; the environment sends back information about its new state and the immediate reward resulting in this action; the agent uses those information to take another action on the environment; this action updates the environment's state according to the transition model; the environment sends back information about its new state and the immediate reward resulting in this action; and so forth until the environment is in a *terminal state* (i.e. no further action is required).

Later in this manuscript, we will refer to Markov Decision Processes. This is a particular case of Sequential Decision Making problems where the transition model is *Markovian* (i.e. the probability of reaching state $\boldsymbol{s}'$ from the state $\boldsymbol{s}$ and an action $\boldsymbol{a}$ depends only on $\boldsymbol{s}$ and $\boldsymbol{a}$, not on the history of earlier states).

Unless specified otherwise, we assume a fully observable environment (i.e. the environment does not "hide" information to the agent) and a Markovian transition

model.

In following sections, we give further details on definitions and chosen notations for the different elements composing a Sequential Decision Making problem. These definitions and notations are mainly taken from [26] and [25] for the control and *Operations Research* community, and from [27] and [28] for the *Reinforcement Learning* community.

1.4.1 Decision epochs

Decision epochs are the times at which a decision has to be made by the agent, or in other words, the times at which an action has to be executed by the agent. In this manuscript, we only consider the case of discrete decision epochs (thus called *time steps*). For the interested readers, [26] gives additional information about the continuous time setting, where decisions are made continuously. Note that discrete time steps don't necessarily mean equally spaced decision epochs; time steps simply define time when the agent has to execute an action.

When relevant, we will index variables with their corresponding time step t, using convention from [25]: a variable X_t indexed by t is deterministic at time t. While at time t, variable X_{t+1} could be both deterministic or stochastic, depending on the context.

When relevant, variables are indexed with the iteration index i in order to make the distinction between the actual time step t and the iteration index i of iterative resolution methods.

1.4.2 State variables

The state variables could contain information about the physical state of the agent, pure information received from the environment or information about history. To make things as clear as possible, we borrow the following definition from [25]:

State variable. A state variable is the minimally dimensioned function of history that is necessary and sufficient to compute the decision function, the transition function, and the contribution function.

According to this definition, the state variable includes enough of the observable information to compute the optimal decision (if this exists with respect to the observable information). Moreover, this definition makes the distinction between Markovian (i.e. history independent) processes and non-Markovian (i.e. history dependent) processes almost useless. Note that this is true from the modelling point

of view, but actually, this distribution still have a big impact on the computational cost when solving or simulating problems. If the history information is required to make decision, to compute the next state, or to compute rewards resulting from an action, then it should be included in the state variable. Otherwise, there is no reason to include it.

Similarly, if the time is required to make decision, to compute the next state, or to compute rewards, then it should be included in the state variable. Otherwise, there is no reason to include it.

We will generally refer to state variables as $\boldsymbol{x} \in \mathcal{X}$, with $\mathcal{X}$ referring to the set of possible states. $\mathcal{X}$ can be finite, infinite but countable, or infinite and uncountable. It is also common to use $\boldsymbol{s}$ and $\mathcal{S}$ respectively as a notation for the state variable and the state space, but it tends to imply that the state space is finite, as it is often used in the framework of finite Markov Decision Process (like in chapter 2). This set is determined by the environment, that is to say by the nature of the problem and by the choices of the model. Some resolution methods require to have a discrete state space, and will thus need to first discretize a continuous state space before doing anything else.

We use a bold font for the state variable as in the general case $\boldsymbol{x}$ and $\boldsymbol{s}$ are vectors.

1.4.3 Action variables

As defined previously, actions (or decisions or controls) are the way for the agent to act on the environment in order to achieve its goal. They are the free variables to be optimized by the agent.

Usually, actions are denoted $\boldsymbol{a} \in \mathcal{A}$ (or sometimes $\boldsymbol{u} \in \mathcal{U}$), where $\mathcal{A}$ is the set of possible actions. The notation $\boldsymbol{d}$ is also used when we talk about decisions. As for the state space, $\mathcal{A}$ can be finite, infinite but countable, or infinite and uncountable. We use the bold notation for the action variable as in the general case $\boldsymbol{a}$ is a vector.

Feasible actions (or *legal actions*) are actions that are possible from a certain state $\boldsymbol{x}$ and/or at some time step t. In this case, the action set is denoted $A_t(\boldsymbol{x})$ or $A(\boldsymbol{x})$ depending on whether the time determines the action set or not. Usually, environments are defined such that there is one unique action set regardless of the current state and time step. In this case, the action set is simply denoted $\mathcal{A}$ for clarity. This space is determined by the environment, that is to say by the nature of the problem and by the choices of the model. As for the state variables, some resolution methods require to have a discrete action space, and will thus need to first discretize a continuous action space before doing anything else.

1.4.4 Policies

When we are solving a Sequential Decision Making problem, we are looking for an optimal (or almost optimal) *policy* (or *strategy* or *controller*) – directly or not depending on the resolution algorithm (more on this in chapter 2). A policy is a function that maps from states to actions. As Sequential Decision Making problems are stochastic, we cannot rely on a simple list of actions to execute as a solution to the problem; because of the uncertainty, the best action to each possible future state should be considered. An optimal policy tells us which action should be executed, for each possible state, in order to maximize (or minimize) the expected reward (or cost) it has been designed for. Usually, policies are denoted

$$\pi : \mathcal{S} \to \mathcal{A}$$

$$\boldsymbol{s} \mapsto \pi(\boldsymbol{s})$$

with $\pi \in \Pi$ and optimal policies are denoted π^*. A policy can be *stationary* if it does not depends on time (i.e. the same policy π is used for every time steps of the problem) or *non-stationary* if it depends on time (i.e. a distinct policy π_t has to be made for each time steps t of the problem).

1.4.5 Random processes

Random processes refer to variables that take random values. These variables exert an influence on the transition function outcome (that is to say the next state). According to [25], we consider that this flow of information arrive continuously to the agent. We note $\boldsymbol{\omega}_t$ the information available at time t.

These random variables are defined by a probability density function denoted $p_{\boldsymbol{\omega}_t}(\boldsymbol{\omega}_t = \boldsymbol{\omega})$ (sometimes written $p_{\boldsymbol{\omega}_t}(\boldsymbol{\omega})$ for short). These probability distributions are usually unknown in practice and estimating the random variables distribution may be a big challenge for many realistic problems. The set of realizations is noted Ω_t. Like for $\mathcal{X}$ and $\mathcal{A}$, the realization set Ω can be finite, discrete and countable, or uncountable.

Depending on the problem, random processes, the sets of possible realizations, and the related probability distributions may depend on the time, the states and the executed actions. In this case, the density function could be written $p_{\boldsymbol{\omega}_t}(\boldsymbol{\omega}_t = \boldsymbol{\omega}|\boldsymbol{x}, \boldsymbol{a})$, and the set of realizations noted $\Omega_t(\boldsymbol{x}, \boldsymbol{a})$. Note that, according to our state definition in section 1.4.2, if $\boldsymbol{\omega}_t$ depends on the history of the process $(\boldsymbol{\omega}'_t)_{1 \leq t' \leq t-1}$, then this history should be included in the state variable $\boldsymbol{x}_t$.

The way the knowledge about this random process is made available can vary from one application to the other. In data driven applications, one might be given a fixed amount of past realizations, and try to infer the hidden distribution. In model based approaches, one needs to have a generative model of the random process, able to generate new realizations whenever prompted.

We will later see that the way the optimizer deals with the random process of the problem is important. Some methods choose to bypass the randomness entirely, optimizing their actions in the case of one fixed series of realizations. Others choose to optimize their action as if the random process would always take values equal to the expectation of this process. And others are trying to make decisions in order to minimize a given risk criterion, like the costs incurred in the 5% worst cases. Finally, some methods are simply trying to optimize the expected cost, computed with respect to the full distribution of the random process or a compromise between the expected cost and the variance. All these approaches have their advantages and drawbacks. Generally speaking, one needs to make a trade-off between being theoretically consistent and optimal, and having a fast and scalable method.

As many random processes in real applications are not Markovian (e.g. weather, prices, ...), people like to consider series of random variables rather than isolated ones. It is then convenient to talk about random scenarios (or time series) to refer to the realizations of these series of random variables.

1.4.6 Transition function

As previously stated, this function describes how the system evolves, from a given state to another, as a result of a given agent's action and the outcome of the random processes. In other words, this *state* transition function defines the dynamics of the environment.

Thus, for a given state $\boldsymbol{x}_t$, action $\boldsymbol{a}_t$, and random process variable $\boldsymbol{\omega}_{t+1}$[10], the next state $\boldsymbol{x}_{t+1}$ is given by the transition function T, so that $\boldsymbol{x}_{t+1} = T(\boldsymbol{x}_t, \boldsymbol{a}_t, \boldsymbol{\omega}_{t+1})$. Sometimes in this document, the transition function is equivalently denoted f.

When the transition function is stochastic, the probability density function of $\boldsymbol{x}_{t+1}$ is sometimes used to define it. In such cases we note it $p(.|\boldsymbol{x}_t, \boldsymbol{a}_t)$. Formally this notation means that for all $\boldsymbol{x} \in \mathcal{X}$, $p(\boldsymbol{x}|\boldsymbol{x}_t, \boldsymbol{a}_t) = p(T(\boldsymbol{x}_t, \boldsymbol{a}_t, \boldsymbol{\omega}_{t+1}) = \boldsymbol{x})$. This probabilistic notation is not systematically used in this manuscript.

At this point, we can illustrate the definition of state variables introduced earlier. If the transition function T requires any observable information that is not contained in $\boldsymbol{a}_t$ or in the possible realization of $\boldsymbol{\omega}_{t+1}$, then it must be included in $\boldsymbol{x}_t$. Moreover

[10] this is the random information that will become available between t ant $t+1$

if all past actions $(\boldsymbol{a}_{t'})_{0\leq t'\leq t-1}$ are needed to compute T, then they should be included in $\boldsymbol{x}_t$. Similarly, if the probability distribution of $\boldsymbol{\omega}_{t+1}$ depends on $(\boldsymbol{\omega}_{t'})_{0\leq t'\leq t}$, then it should be included in $\boldsymbol{x}_t$. Of course, if for some computational reasons, one does not want to include all that information in $\boldsymbol{x}_t$, it is possible to truncate the state to a more manageable size. But then, one needs to be aware that the sampled realizations of $\boldsymbol{\omega}_{t+1}$ are very likely to be biased, and that the result of T may be far from the true model's result.

Stationary and non stationary transitions. So far we have written the transition function assuming it does not depend on time (it is usually the case in practice). This kind of transition functions is called *stationary transition*.

However, sometimes, the parameters or the structure of the function depend on time. These functions are said to be *non stationary*; they should not be mistaken with stationary transitions functions which depends on data that depends on time.

According to our definition of state variables, non stationary transition functions can be rewritten to become stationary. Indeed, in this case, the time can simply be included in the state variable. Nevertheless, sometimes ones want to make more obvious and explicit the time dependence of the transition function, and still overload the notations using non stationary transitions.

1.4.7 Reward function and cost function

Depending on the problem, either a cost function or a reward function is used to quantify the preferences (or *utility*) of the agent. This function denotes the agent's goal and influences its actions choices. A cost function is used while minimizing and a reward function is used while maximizing; these functions are interchangeable by taking their respective opposite values. In the Reinforcement Learning community, problems are usually defined such that rewards are used instead of costs. By default, we will use this convention, and the following definitions are given considering rewards but can similarly be applied to costs.

In this document, cost functions are denoted c and reward functions are denoted r. The *immediate reward* is computed at each time step by the environment, and is given by $r_t = r(\boldsymbol{x}_t, \boldsymbol{a}_t, \boldsymbol{\omega}_{t+1})$, $r_t = r(\boldsymbol{x}_t, \boldsymbol{a}_t, \boldsymbol{x}_{t+1})$ or simply $r_t = r(\boldsymbol{x}_t)$ depending on the problem and the resolution method considered. Using this notation, the objective of the agent is to maximize the long-term reward (also named global or total reward) usually defined as the sum of immediate rewards $R_T = \sum_{0\leq t\leq T} r_t$. Another frequently used long-term reward function is the discounted sum of immediate rewards $R_T = \sum_{0\leq t\leq T} \gamma^t r_t$ where γ is the *discount factor*, which allocates different weights to

immediate rewards according to their distance to the present time. For the problem to be properly defined, we need to have $R_T < \infty$. When R_N is a random quantity, the question mentioned before arise: what are we trying to achieve ? We may try to maximize the expected rewards, maximize the "worst case scenario" rewards (when this even makes sense), or maximize the rewards assuming one given scenario will happen.

Non stationary reward functions. As for transition functions, reward functions might depend on the time t. In this case, following our definition of state variables, $\boldsymbol{x}_t$ should include the time.

1.5 Optimization and Model Errors

The purpose of this thesis is not to provide detailed models for Unit Commitment problems but to provide efficient methods to solve Unit Commitment problems defined with such detailed model. It is important to understand that simplifications on the model usually contribute to the suboptimality of obtained solutions.

1.5.1 Optimization error

Because of limited computation time or because of the use of approximate algorithms, the optimal solution may not be found. Typically:

- most *value-based* methods will assume that the random processes involved can be reduced to a small Markovian Random Process (see chapter 2);
- *Stochastic Dual Dynamic Programming* (see chapter 3) will assume that the value function is convex. This is not the case e.g. with precise models of hydroelectricity;
- *Stochastic Dynamic Programming* (see chapter 3) will often discretize the state space;
- *Direct Policy Search* (see chapter 2) assumes an arbitrary parametric policy;

1.5.2 Objective function and constraint errors

For various reasons, the objective function and the constraints used for optimisation may not be realistic.

Simplification error: as mentioned above, often, the model is simplified to comply with algorithms assumption [29, 2, 20]: terms are removed or replaced by simple linear terms, time steps are aggregated, a less precise time resolution is used, some constraints are ignored (e.g. geographical distribution), ... But the obtained optimal solution may be far from the actual optimum.

Anticipativity error: this occurs when the system is optimized as if it were possible to predict the future (the optimisation is performed on chosen scenarios). For example, we might decide the capacity of a connection, using simulations of a power grid in which all operational decisions are made with perfect knowledge of the future.

Statistical error: when some data are obtained through a finite sample, which might be too small (e.g. archive of 40 years of climate data) and/or biased (e.g. climate change).

1.6 Main Contributions

1.6.1 Direct Value Search

Our main contribution, presented in the fourth chapter, is an algorithm designed to solve large scale unit commitment problems. With this algorithm, named *Direct Value Search* (DVS), we propose a new method that couples the main ideas and advantages of the two families of algorithms presented in Chapter 2 and 3, namely *Dynamic Programming* (*Bellman Methods*) and *Direct Policy Search*.

Thanks to this new coupling, Direct Value Search have interesting features to tackle challenges listed in section 1.3.2. Indeed, we will see that Direct Value Search is an anytime algorithm (users immediately get approximate results) which can handle large state spaces and large action spaces with non-convex constraints, and without assumptions on the random process.

Moreover, as it can work with non-convex constraints and without assumption on the random process, Direct Value Search is capable to solve problems defined with more detailed and realistic models, reducing then the modelling errors presented in section 1.5, and thus improving the quality of proposed solutions.

Concrete examples of Unit Commitment problems we have worked on

As discussed in section 1.3.2, there are important and challenging Unit Commitment problems at all scales, from the very local scale (e.g. the *Citines* project in table

1.1), to transcontinental problems and very long-term planning (e.g. *Post* project in table 1.1 or MedGrid[11]).

Various Unit Commitment problems with different characteristics have been used to test our algorithms. The main features of these problems are summarized in table 1.1.

	time steps	state variables	decisions	loss	constraints	horizon
short term unit commitment (*Citines* project)	10 min	(local) thermal plants and stocks state	bidding and stock choices	hydro and thermal costs	operational constraints	2 days
hydroelectric scheduling (*IOMCA* project)	days(*)	hydroelectric stock levels	hydroelectric power plants output	cost of thermal power plants	hydro constraints and maintenance (in such a case thermal plants are often simplified)	5 years
nuclear power plants maintenance	weeks(*)	hydro level and maintenance status	maintenance and hydro operation	production cost	operational constraints	8 years
financial planning (*POST* project)	weeks(*)	hydro level and financial status	financial decisions	production cost	operational and investment constraints	40 years

Table 1.1: (*) stands for cases in which time steps are usually not contiguous, which is beyond the scope of this work.

In table 1.1, Citines is a FP7[12] project conducted by Artelys[13] where a production strategy (under uncertainties) has to be found for cities, factories, ... IOMCA

[11] `http://www.medgrid-psm.com/en/`

[12] European-funded project

[13] `http://www.artelys.com`, a company specialized in optimization, statistics and decision-support

is an ANR[14] project. This is a classical hydrothermal coordination problem (a Unit Commitment problem involving hydroelectric power stations and thermal plants) with long-term planning (several years), taking into account stochasticity and uncertainties in demand and production. See [30] for further details. POST[15] is one of the 8 projects "réseaux électriques intelligents innovent" (intelligent electric network) in *Programme d'Investissements d'Avenir* (PIA), financed by ADEME[16]. This is a high scale (Europe and North Africa) and long-term (horizon 2030 to 2050) investment problem with huge (non-stochastic) uncertainties about, inter alia, future technologies, laws, and clients demand.

1.6.2 Noisy Optimization

The fifth chapter is dedicated to Noisy optimisation, a key component of the Direct Value Search algorithm. In this chapter, we have studied some theoretical convergence rate and we have proved new convergence bounds for given families of objective functions – under some reasonable assumptions. Besides, in section 5.3 we have studied some variance reduction techniques aimed at improving the convergence rate of graybox noisy optimization problems.

1.6.3 Non quasi-convex functions

As many users of Direct Policy Search variants, we use evolutionary algorithms. All convergence proofs of evolution strategies, with log-linear rates, are based on convex functions. We provide a proof in the non-quasi-convex setting. The proof is given for a simplified evolutionary algorithm with an ad hoc mutation which is more convenient for proofs.

[14] *Agence nationale de la Recherche*, a French agency which finance public research in France since 2005

[15] *Plateforme d'Optimisation des Supergrids Transcontinentaux*

[16] `http://www.presse.ademe.fr/2013/06/les-reseaux-electriques-intelligents-innovent.html`

Part II

Background Review

Chapter 2

Classical Algorithms for Sequential Decision Making

In this chapter, we present some classical algorithms for "general purpose" Sequential Decision Making under uncertainty and particularly *Markov Decision Process* (or MDP) (see section 1.4). Some other Sequential Decision Making algorithms – rather general but more widely used in energy management – are described in chapter 3.

The purpose of this chapter is to introduce some classic concepts like *Dynamic Programming* and *Bellman's Principle of Optimality*, *Bellman equations*, *value search* and *policy search*, ... as these concepts will be required for a better understanding of our main contribution – Direct Value Search – described in chapter 4.

Of course, there exist many other classical methods to solve Sequential Decision Making problems and methods described in this chapter have been chosen to be the most relevant according to our goals (section 1.2) and to out main contribution (chapter 4). Also, we focus on stochastic methods, thus we will not talk here about mathematical programming or Lagrangian relaxation.

There exist many algorithms to compute a Markov Decision Process's optimal policy. In the following sections, we briefly describe the most popular ones. The notation used in this chapter (and in the following ones) has been introduced in section 1.4.

2.1 Backward Induction

Backward Induction is a basic *Dynamic Programming* method [31]. Like other Dynamic Programming algorithms, it uses the *Bellman's Principle of Optimality* [31] for accelerating computation (compared to an exhaustive search) with problems which

exhibit a compatible structure (i.e. a problem defined with *overlapping subproblems* and an *optimal substructure* [31]). Actually, this acceleration is obtained by breaking problems down into simpler subproblems in such a manner that redundant computations are avoided.

Backward Induction computes non-stationary policy: one different policy is computed for each time step. Thus the number of time steps used to solve the problem is set in advance. Like most methods presented in this chapter, *Backward Induction* algorithms solve Sequential Decision Making problems defined with discrete actions and state spaces.

The *value* (or *utility*) V^* for each state $\boldsymbol{s}$ at the latest time step T is

$$V_T^*(\boldsymbol{s}) = r(\boldsymbol{s}) \tag{2.1}$$

where r is the immediate reward function defined in section 1.4.7. The best expected value V^* for each state $\boldsymbol{s}$ at the t^{th} time step is

$$V_t^*(\boldsymbol{s}) = r(\boldsymbol{s}) + \max_{\boldsymbol{a} \in \mathcal{A}} \left[\sum_{\boldsymbol{s}' \in \mathcal{S}} T(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') V_{t+1}^*(\boldsymbol{s}') \right] \tag{2.2}$$

and the t^{th} optimal action (or decision) $d_t^*(\boldsymbol{s})$ among the set of possible actions $\mathcal{A}$ is

$$d_t^*(\boldsymbol{s}) = \arg\max_{\boldsymbol{a} \in \mathcal{A}} \left[\sum_{\boldsymbol{s}' \in \mathcal{S}} T(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') V_{t+1}^*(\boldsymbol{s}') \right] \tag{2.3}$$

where T is the transition function defined in section 1.4.6.

The main idea is to compute the expected value of each state (Eq. 2.2) and then to use it to select the best action for any given state (Eq. 2.3).

Eq. 2.2 cannot be solved analytically because the system of equations to compute V contains non-linear terms (due to the "max" operator). As an alternative, Eq. 2.2 is usually computed using Dynamic Programming method, as described in algorithm 1.

2.2 Value Iteration

Value Iteration [31] is one of the most famous algorithm to compute MDP's optimal policy. Like for *Backward Induction* (section 2.1), the main idea implemented by Value Iteration is to compute the best expected value of each state and then to use these values to select the best action from any given state.

Algorithm 1 Backward Induction

Input:

$mdp = \langle \mathcal{S}, \mathcal{A}, T, r \rangle$, a Markov Decision Process

T, the resolution horizon (ie. the number of time steps)

Local variables:

$V_t^* \quad \forall t \in \{1, ..., T\}$, vectors of utilities for states in $\mathcal{S}$

$V_T^*[\boldsymbol{s}] \leftarrow r(\boldsymbol{s}) \quad \forall \boldsymbol{s} \in \mathcal{S}$

for all $t \in \{T-1, T-2, ..., 1\}$ **do**

  for all $\boldsymbol{s} \in \mathcal{S}$ **do**

    if $\boldsymbol{s}$ is a final state **then**

      $V_t^*[\boldsymbol{s}] \leftarrow r(\boldsymbol{s})$

    else

      $$V_t^*[\boldsymbol{s}] \leftarrow r(\boldsymbol{s}) + \max_{\boldsymbol{a} \in \mathcal{A}} \left[\sum_{\boldsymbol{s}' \in \mathcal{S}} T(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') V_{t+1}^*[\boldsymbol{s}'] \right]$$

    end if

  end for

end for

return $V_t^* \quad \forall t \in \{1, ..., T\}$

The main difference with the Backward Induction algorithm is that Value Iteration computes a stationary policy. Indeed, the same resulting policy is used for each time step and thus there is no assumption about the number of time steps to consider for the solution.

The expected value V^π for each state $\boldsymbol{s}$ when the agent follow a given (stationary) policy π is

$$V^\pi(\boldsymbol{s}) = E\left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t)|\pi, \boldsymbol{s}_0 = \boldsymbol{s}\right] \tag{2.4}$$

The optimal (stationary) policy π^* is defined using the best expected value V^{π^*} and using the principle of *Maximum Expected Utility* as follow

$$\pi^*(\boldsymbol{s}) = \arg\max_{\boldsymbol{a}\in\mathcal{A}} \left[\sum_{\boldsymbol{s}'\in\mathcal{S}} T(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')V^{\pi^*}(\boldsymbol{s}')\right] \tag{2.5}$$

Eq. 2.6 is commonly named the *Bellman equation*; it gives the best value we can expect for any given state (assuming optimal policy π^* is followed). There are $|\mathcal{S}|$ Bellman equations, one for each state. As for the Backward Induction method, this system of equations cannot be solved analytically because Bellman equations contains non-linear terms (due to the "max" operator). As an alternative, Eq. 2.6 can be computed iteratively using a Dynamic Programming method named Value Iteration and described in Algorithm 2.

$$V(\boldsymbol{s}) := V^{\pi^*}(\boldsymbol{s}) = \begin{cases} r(\boldsymbol{s}) & \text{if } \boldsymbol{s} \text{ is a final state} \\ r(\boldsymbol{s}) + \gamma \max_{\boldsymbol{a}\in\mathcal{A}} \left[\sum_{\boldsymbol{s}'\in\mathcal{S}} T(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')V(\boldsymbol{s}')\right] & \text{otherwise} \end{cases} \tag{2.6}$$

Equation 2.7 – called *Bellman update* – is used in the iterative method described in Algorithm 2, to update V at each iteration.

$$V_{i+1}(\boldsymbol{s}) \leftarrow \begin{cases} r(\boldsymbol{s}) & \text{if } \boldsymbol{s} \text{ is a final state} \\ r(\boldsymbol{s}) + \gamma \max_{\boldsymbol{a}\in\mathcal{A}} \left[\sum_{\boldsymbol{s}'\in\mathcal{S}} T(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')V_i(\boldsymbol{s}')\right] & \text{otherwise} \end{cases} \tag{2.7}$$

Convergence The convergence of Value Iteration has been proved, but this convergence is asymptotic [31]. However, each iteration is easy and fast to compute.

Algorithm 2 Value Iteration

Input:

$mdp = \langle \mathcal{S}, \mathcal{A}, T, r \rangle$, a Markov Decision Process

γ, the discount factor

ϵ, the maximum error allowed in the utility of any state in an iteration

Local variables:

V, V', vector of utilities for states in $\mathcal{S}$, initially zero

δ, the maximum change in the utility of any state in an iteration

repeat
  $V \leftarrow V'$
  $\delta \leftarrow 0$
  for all $\boldsymbol{s} \in \mathcal{S}$ **do**
    if $\boldsymbol{s}$ is a final state **then**
      $V'[\boldsymbol{s}] \leftarrow r[\boldsymbol{s}]$
    else
      $V'[\boldsymbol{s}] \leftarrow r[\boldsymbol{s}] + \gamma \max_{\boldsymbol{a} \in \mathcal{A}} \left[\sum_{\boldsymbol{s}' \in \mathcal{S}} T(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') V[\boldsymbol{s}'] \right]$
      if $|V'[\boldsymbol{s}] - V[\boldsymbol{s}]| > \delta$ **then**
        $\delta \leftarrow |V'[\boldsymbol{s}] - V[\boldsymbol{s}]|$
      end if
    end if
  end for
until $\delta < \epsilon(1 - \gamma)/\gamma$
return V

2.3 Policy Iteration

Policy Iteration [32] is another very popular algorithm to compute MDP's optimal policy. In practice, it is often faster than Value Iteration.

The Policy Iteration algorithm alternates the following two steps, beginning from some initial policy π_0:

1. Policy Evaluation: given a policy π_i, it calculates $V^{\pi_i}(\boldsymbol{s})\ \forall \boldsymbol{s} \in \mathcal{S}$, the expected value of each state when π_i is followed.

2. Policy Improvement: it calculates a new policy π_{i+1}, using one-step look-ahead based on V^{π_i} and using the principle of *Maximum Expected Utility* as follow

$$\pi_{i+1}(\boldsymbol{s}) = \arg\max_{\boldsymbol{a} \in \mathcal{A}} \left[\sum_{\boldsymbol{s}' \in \mathcal{S}} T(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') V^{\pi_i}(\boldsymbol{s}') \right] \tag{2.8}$$

Algorithm 3 describes the two-step procedure. The algorithm terminates when the *Policy Improvement* step yields no change in the utilities.

Solving the POLICY-EVALUATION routine is much simpler than solving the standard Bellman equations (which is what Value Iteration does). Indeed, the action in each state is fixed by the policy, thus the "max" operator disappears and Bellman equations become linear. As a result, V^{π_i} can be computed by solving the linear system of these *simplified Bellman equations* (Eq. 2.9) for each state.

$$V^{\pi_i}(\boldsymbol{s}) = \begin{cases} r(\boldsymbol{s}) & \text{if } \boldsymbol{s} \text{ is a final state} \\ r(\boldsymbol{s}) + \gamma \sum_{\boldsymbol{s}' \in \mathcal{S}} T(\boldsymbol{s}, \pi_i(\boldsymbol{s}), \boldsymbol{s}') V^{\pi_i}(\boldsymbol{s}') & \text{otherwise} \end{cases} \tag{2.9}$$

2.3.1 Convergence

As the number of states and policies is finite, and as the policy is improved at each iteration, Policy Iteration converges in a finite number of iterations (often small in practice). However, within each iteration, solving the POLICY-EVALUATION routine may cost a lot (its complexity is $O(|\mathcal{S}|^3)$).

2.3.2 Notable Policy Iteration variants

- Modified Policy Iteration [33] and [34], speed-up the POLICY-EVALUATION routine with approximated evaluations of V using $ValueIteration$.

Algorithm 3 Policy Iteration

Input:

$mdp = \langle \mathcal{S}, \mathcal{A}, T, r \rangle$, an MDP

Local variables:

V, vector of utilities for states in $\mathcal{S}$, initially zero

π, a policy vector indexed by state, initially random

repeat

$V \leftarrow$ POLICY-EVALUATION(π, V, mdp)

unchanged? $\leftarrow$ *true*

for all state $\boldsymbol{s} \in \mathcal{S}$ **do**

if $\max_{\boldsymbol{a} \in \mathcal{A}} \left[\sum_{\boldsymbol{s}' \in \mathcal{S}} T(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') V[\boldsymbol{s}'] \right] > \sum_{\boldsymbol{s}' \in \mathcal{S}} T(\boldsymbol{s}, \pi[\boldsymbol{s}], \boldsymbol{s}') V[\boldsymbol{s}']$ **then**

$\pi[\boldsymbol{s}] \leftarrow \arg\max_{\boldsymbol{a} \in \mathcal{A}} \left[\sum_{\boldsymbol{s}' \in \mathcal{S}} T(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') V[\boldsymbol{s}'] \right]$

unchanged? $\leftarrow$ *false*

end if

end for

until *unchanged*?

return π

- Asynchronous Policy Iteration [35], speed-up the Policy Iteration by considering only a (well-chosen) subset of $\mathcal{S}$ when updating V and π.

2.4 Direct Policy Search

Direct Policy Search (DPS) methods (or simply *Policy Search*) are widely used in reinforcement learning. As the name indicates, these methods search in the space of policy Π for an optimal policy π^*. Actually, they optimize – on simulations – the parameter $\boldsymbol{\theta}$ of a given parametric policy $\pi_{\boldsymbol{\theta}} \in \Pi$. For instance, $\pi_{\boldsymbol{\theta}}$ can be a neural network, $\boldsymbol{\theta}$ being the weights of neurons.

Policy $\pi_{\boldsymbol{\theta}}$ can be differentiable with respect to $\boldsymbol{\theta}$ or not; in the first case, a *Gradient-Based Policy Search* method is generally used whereas in the second one a *Gradient-Free Policy Search* is used. In this thesis, when we evoke Direct Policy Search, we implicitly refer to the *Gradient Free* case. This method considers the problem of finding a good parameter $\boldsymbol{\theta}$ as a noisy (gradient-free) optimization problem. Existing techniques, such as evolutionary optimization (see section 4.1.2) or cross-entropy methods can be used to optimize $\boldsymbol{\theta}$. The optimisation procedure is defined in Algorithm 4 and the objective function considered for the maximization of $\boldsymbol{\theta}$ is given in Algorithm 5. Using human expertise for defining the π function can make this approach quite efficient and anytime [36]. However, it is difficult to ensure that $\pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t)$ verifies decision constraints.

The key advantages of DPS include:

- Direct Policy Search does not need a particular linear, quadratic or analytical model of the system. The only requirement is a generative model (a black-box simulator).
- There is no need for a decomposable criterion neither (e.g. medians or quantiles of delayed costs can be optimized).
- The simulations naturally focus on likely parts of the state space, without the tedious backward induction on all the state space as in backwards Bellman-based tools.
- There is no need for a particular Markovian Random Process; any Random Process (Markovian or not) can be used within the black-box simulator.

There are also some drawbacks of using DPS:

- A huge size of the action space implies, for many parametric policies, a huge size of the parameter space, leading to prohibitory resolution times.
- Constraints on the action space are difficult to handle in Direct Policy Search.

Algorithm 4 The Direct Policy Search procedure: $\hat{\boldsymbol{\theta}} \leftarrow$Direct Policy Search$(\pi_{\boldsymbol{\theta}}, MNMDP, \boldsymbol{x})$. The non linear optimization tool used for finding $\hat{\boldsymbol{\theta}}$ is not specified this general definition.

Input:
a parametric policy $\pi_{\boldsymbol{\theta}}(.)$,
where $\pi_{\boldsymbol{\theta}}$ is a mapping from the state space
to the decision space,
a *Markovian or Non-Markovian Decision Process* MNMDP,
an initial state $\boldsymbol{x}$

Output:
a parameter $\hat{\boldsymbol{\theta}}$ leading to a policy $\pi_{\hat{\boldsymbol{\theta}}}(.)$

Find $\hat{\boldsymbol{\theta}}$ maximizing the expectation of
$\boldsymbol{\theta} \mapsto$ Simulate$(\pi_{\boldsymbol{\theta}}, \text{MNMDP}, \boldsymbol{x})$
with a given non-linear noisy optimization algorithm like the one discussed in section 4.1.2

return $\hat{\boldsymbol{\theta}}$

Algorithm 5 The *Simulate* function: $r \leftarrow$Simulate$(\pi, MNMDP, \boldsymbol{x})$

Input:
a policy π,
a Markovian or Non-Markovian Decision Process MNMDP,
an initial state $\boldsymbol{x}$

Output:
a reward r

Simulate MNMDP from state $\boldsymbol{x}$ with policy π until a terminal state and return the obtained reward r

return r

Chapter 3

The Most Classical Solutions for Energy Stock Management

As explained in chapter 1, Unit Commitment are complex multistage optimization problems with both integer and continuous variables. The exact solution to the problem can not be obtained with realistic power system models, because of its excessive computation time requirement.

Due to the interpretability of Bellman value functions (Chap. 2), and due to the elegance of the Bellman optimality principle, many practitioners prefer value-based methods, which provide simple rules for the pricing of resources. For example, a value function for water provides – thanks to the derivative with respect to stock – the marginal price of water in dams. This is very helpful for optimizing locally the dispatch of units. Dual versions provide a significant improvement in terms of speed, admittedly with a cost in terms of modelling – not all power systems can be rewritten with dual dynamic programming.

Classical backwards Dynamic Programming is expensive and tricky. Therefore, *Model Predictive Control* (section 3.1) is sometimes preferred. The only but strong assumption of Model Predictive Control is the perfect forecast over some *tactical horizon* (section 3.1).

This chapter reviews the most classical methodologies for power systems optimization:

- *Stochastic Dynamic Programming* [31] (including dual versions), using scenario tree generation [37]. It is briefly presented in Section 3.5. The limits in the performance of such methods are the discretization and, in many cases, simplifying assumptions made on the modeling, for accelerating the solving (see section 1.5), e.g.:

  - the convexity of Bellman values, which, for instance, is not verified in some unit commitment problems where the water turbines efficiency increases as a function of stocked water quantity (see section 1.3.2);
  - the nice properties of the random process as explained above;
  - sometimes the separability of the Bellman values over state variables (i.e. $V(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = \sum_{i \in \{1,\ldots,k\}} V_i(\boldsymbol{x}_i)$).

  These assumptions are justified by the huge computational cost when trying to solve a more realistic model. For example, with unit commitment problems, optimizing over a group of 250 hydroelectric stocks by SDP is impossible without such approximations and is still quite hard.

- *Approximate Dynamic Programming* [38], using other assumptions such as the good learnability of Bellman values by an approximate model (e.g. *Support Vector Machines* [39] or *Neural Networks*). A crucial drawback is the loss of the polynomial *Decision Making Runtime*[1] or DMR when the approximate Bellman values can not be encoded in a linear problem (more on this below).

- *Model Predictive Control*, replacing the uncertain random variables by a deterministic forecast. This deterministic approximation of uncertainties may be the expected value of the random variables or more commonly a pessimistic forecast [40]. This methodology does not aim at tackling highly stochastic random processes [20]; it is nonetheless quite practical, with moderate *Decision Making Runtime* and nearly null *Solving Runtime*[2] (or SR).

3.1 Model Predictive Control

Model Predictive Control (MPC) is a very simple algorithm. For making a decision, it replaces all stochastic parts by a deterministic approximation, and then, it optimizes the decisions as if reality was this deterministic approximation. In other words, it simplify a stochastic problem into a deterministic one, much easier to solve. This approach is also known as *anticipativity*. Thus, Model Predictive Control applies an *open loop control* (also termed *conformant planning*): the decisions for all time steps are made once and for all, independently of the state variables. This simple principle can be improved in several ways:

[1] The computation time for making one decision with a given policy, i.e. the online computational cost.

[2] The computation time for building the policy, i.e. the offline computational cost.

- *Receding horizon* (also termed *shrinking horizon* or *rolling planning*): after having made decisions, we wait a given number h of time steps, and then we re-optimize the decisions; h is then termed the *operational horizon*. This means that we perform many optimizations instead of just one; if we work on T time steps, then we have $\lceil T/h \rceil$ optimization problems. The advantage of this trick is that observations of the state are then used to adjust the deterministic approximation of all stochastic parts (in other words MPC applies mixture of open-loop control and closed-loop control). The principle of receding horizon is illustrated in Fig. 3.1.

- *Tactical horizon* (Fig. 3.2): optimizing over the next T time steps might be a challenge in terms of computational power, in particular in case of huge power systems with many constraints. Therefore, it is a common practice to optimize only the costs corresponding to the k next time steps. In such a case, k is termed the *tactical horizon*, to be distinguished from the total number T of time steps, termed *strategic horizon*. This has the obvious drawback that long-term effects are not taken into account. On the other hand, it has two advantages:

  - due to limited tactical horizon, the results are less biased by an assumption of perfect forecast;
  - the computational cost is much lower as the number of decision variables becomes usually linear in the tactical horizon k instead of the strategic horizon T.

Fig. 3.3 illustrates the case where both *Receding horizon* and *Tactical horizon* are used.

3.2 Stochastic Dynamic Programming

Richard Bellman [31] has shown that, in the Markov Decision Process case and for an optimization of the average reward, an optimal policy π for solving Eqs. 3.1-3.3

$$(r_t, \boldsymbol{s}_{t+1}) = f(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\omega}_t) \tag{3.1}$$

$$R = \sum_{t \in \{0,\ldots,T-1\}} r_t \tag{3.2}$$

$$\boldsymbol{\omega}_t = Random \tag{3.3}$$

$$\boldsymbol{a}_t = \pi(\boldsymbol{s}_t) \tag{3.4}$$

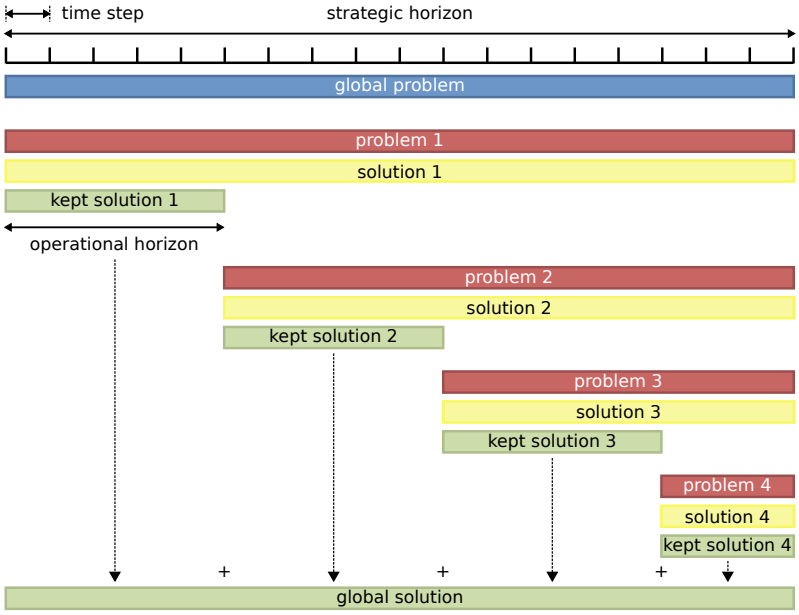

Figure 3.1: Receding horizon.

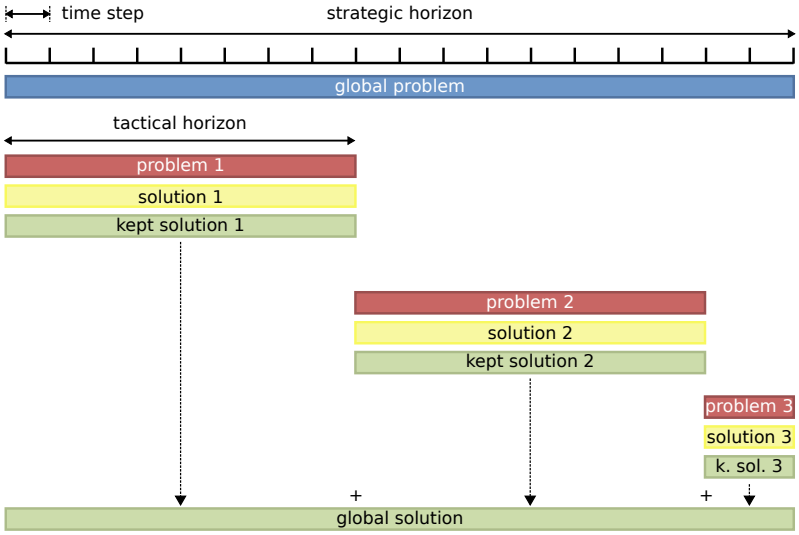

Figure 3.2: Tactical horizon.

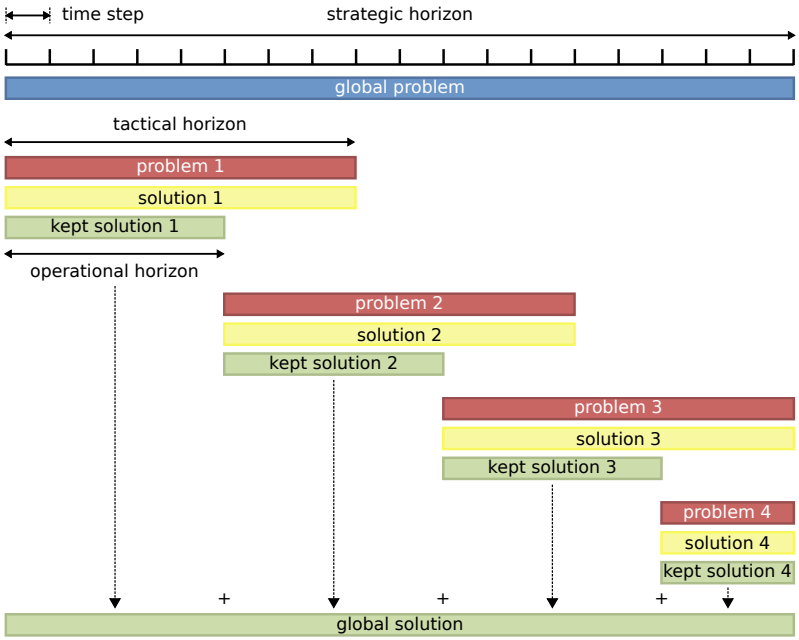

Figure 3.3: Tactical horizon with Receding.

is defined as follows:

$$\pi(\boldsymbol{s}_t) = \arg\max_{\boldsymbol{a}_t \in \mathcal{A}} [r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\omega}_t)] + V(\boldsymbol{s}_{t+1}, t+1), \tag{3.5}$$

where $r_t := r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\omega}_t)$ is the instantaneous reward associated to decision $\boldsymbol{a}_t$ and

$$V(\boldsymbol{s}, t) = \sup_{\boldsymbol{a}} \mathbb{E}\left[r_t + r_{t+1} + \cdots + r_T\right], \tag{3.6}$$

with $\boldsymbol{s} := \boldsymbol{s}_t$ and where π ranges over legal policies for each time steps $t' \geq t$. Note that the notation has slightly changed since Chapter 2; we switched form a simple notation for general algorithms to a more common notation in Power Systems community[3]. We will keep this new notation in chapter 4.

A variant of Eq. 3.6 with tactical horizon k (assuming that we can predict $\boldsymbol{\omega}_t, \ldots, \boldsymbol{\omega}_{t+k}$ with good precision) is possible:

$$V(\boldsymbol{s}, t) = \sup_{\boldsymbol{a}_t, \ldots, \boldsymbol{a}_{t+k-1}} \mathbb{E}\left[r_t + \cdots + r_{t+k-1}\right] + V(\boldsymbol{s}_{t+k}, t+k) \tag{3.7}$$

[3] Here the time is explicitly put as an argument of V, but t could equivalently be hidden in the state vector (as explained in chapter 1). Also we consider here that the reward r depends on the action $\boldsymbol{a}$; in order to simplify equations, it was not the case in chapter 2. Finally the transition function f now receives the exogenous random part as an argument.

with $(r_t, \boldsymbol{s}_{t+1}) = f(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\omega}_t)$.

The problem for applying Eq. 3.5 is that $V(\boldsymbol{s}, t)$ is hard to compute. Using Eq. 3.5 implies that we know an optimal π, which means that the problem is already solved. However, it can be computed backward in time as explained in Section 2.1; if, for some t, you know $V(\boldsymbol{s}, t+1)$ for all $\boldsymbol{s}$, then

$$V(\boldsymbol{s}, t) = \sup_{\boldsymbol{a}_t} \mathbb{E} r_t + V(\boldsymbol{s}_{t+1}, t+k+1) \tag{3.8}$$

with $(r_t, \boldsymbol{s}_{t+1}) = f(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\omega}_t)$.

This is correct only when random variables $\boldsymbol{\omega}_t$ are independent. The extension to $\boldsymbol{\omega}_t$ Markovian is classical, but the general case is hard (because Eq. 3.5 is not meaningful anymore; see Eq. 3.9 for the general case and Eq. 3.10 for the intermediate Markovian case).

$$\boldsymbol{a}_t = \pi(\boldsymbol{s}_t, \boldsymbol{\omega}_0, \ldots, \boldsymbol{\omega}_{t-1}). \tag{3.9}$$

$$\boldsymbol{a}_t = \pi(\boldsymbol{s}_t, \boldsymbol{\omega}_{t-1}). \tag{3.10}$$

Whereas Direct Policy Search has no such problem, Stochastic Dynamic Programming methods require the reward function to be additive (i.e. the global reward is a sum of rewards at each time step and rewards for a given state/action pair do not depend on previous or later states). This is a key problem when working e.g. in the classical cases of minimum water flow constraints in power systems (Unit Commitment usually involves hydroelectricity which is penalized by long-term drought effects, which are difficult "joint" terms in the objective function involving several time steps).

3.3 Stochastic Dual Dynamic Programming

The *Stochastic Dual Dynamic Programming* (SDDP) algorithm was originally proposed in [41]. The principle is to use some gradient or subgradient information in order to quickly approximate the value function by a piecewise linear approximation. It is however difficult to take into account a significant random process, unless we assume that the random process is Markovian with a moderate state space. SDDP also assumes that the value function is convex, so that it can be approximated by piecewise linear functions. On the other hand, SDDP is quite fast for solving stochastic dynamic optimization problems and it is one of the most usual variants in the literature. The SDDP procedure is described in Algorithm 6.

SDDP is successful thanks to the use of *Benders methods* for extracting linear functions as lower bounds on the value function V. The representation of convex value functions by piecewise linear functions is elegant and efficient. [42] shows that a finite (though quickly increasing) number of cuts is enough when each time step involves piecewise linear functions. An important point is that high-dimensional action spaces are not a problem when Eq. 3.8 can be encoded in linear programming; the difficulties are caused by the size of the state space. Thanks to forward/backwards successive approximations, SDDP is also more "anytime" than standard backwards SDP. Also it provides an exact solution:

- if the problem is stagewise independent (the $\boldsymbol{\omega}_i$ are independent) and V is convex as a function of $\boldsymbol{x}$ [43];

- or, seemingly (though we did not find a formal proof in the literature) if V is convex as a function of $\boldsymbol{x}$ and $\boldsymbol{x}$ contains all the information included in $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_{t-1}$ (consistently with [44]).

3.4 Approximate Stochastic Dynamic Programming (ADP)

Eq. 3.8 from Stochastic Dynamic Programming is usually impossible to apply directly on problems defined with continuous state space, as infinitely many states have to be considered. A popular solution consists in evaluating the Bellman value V at each time steps for finitely many states $\boldsymbol{x}_i$, and then to apply some learning algorithm (such as Neural Networks, sums of Gaussians or Support Vector Machines) to the $(\boldsymbol{x}_i, V(\boldsymbol{x}_i, t))$ in order to get an approximation $\hat{V}(., t)$. This approximation is used in lieu of $V(., t)$ for computing $V(\boldsymbol{x}, t-1)$ as in Eq. 3.5. This methodology is termed *Approximate Stochastic Dynamic Programming* or simply *Approximate Dynamic Programming* (ADP). Except through tricks like repeating the process with an exponentially increasing discretization size, ADP is not an "anytime" process.

Stochastic Dual Dynamic Programming, presented above, can be considered as a particular case of ADP; it uses piecewise linear functions as approximations of the Bellman equation $V(.)$. We refer to [45] for more on ADP and its variants.

Algorithm 6 Stochastic Dual Dynamic Programming. In most implementations, the random process state is discarded, as it is complicated and expensive to deal with it (this is the heart of stagewise independence)

Input:
$mdp = \langle \mathcal{X}, \mathcal{A}, T, r \rangle$, a Markov Decision Process
Local variables:
$V_t \quad \forall t \in \{1, ..., T\}$, vectors of utilities for states in $\mathcal{X}$ (the value function)

Initialize the current value function V

while true do
// Step 1
Perform one simulation from $t = 0$ to $t = T$, using the current value function V

The output of this simulation is:
a sequence of random processes states $\boldsymbol{\omega}_t$,
a sequence of states $\boldsymbol{x}_t$
a sequence of subgradients of V w.r.t $\boldsymbol{x}$ (not necessarily all variables in $\boldsymbol{x}$)
at each $(\boldsymbol{\omega}_{t-1}, \boldsymbol{x}_t)$ (i.e. at each $\boldsymbol{x}_t$, if $\boldsymbol{\omega}_{t-1}$ is included in $\boldsymbol{x}_t$)

// Rem: the subgradient provides a lower bound on $V(\boldsymbol{x}_t)$ assuming that $V(\boldsymbol{x}')$

// Step 2
for all $\boldsymbol{x}_t$ **do**
Update V adding a Benders cut:
$V = \max(V, \text{the linear approximation which has been backed up for} \boldsymbol{x}_t \text{at step 1})$
end for
end while

return V

3.5 Anticipativity and Cross-validation

Random processes (RP), in the framework of Sequential Decision Making, are usually modelled by *scenario trees*. A scenario tree is a discrete *Markov chain* (without recombinations in most cases).

Strategies then depend on:

- a "state" (variable which depend on the decisions and which are not modeled by the scenario tree);
- the "node" in the scenario tree.

For example, in a SDP setting, there might be Bellman values depending on both. This ensures that decisions made at a node depend only on the past observations, not on future decisions. This assumption is intended to avoid *anticipativity*.

Mathematically speaking, under the assumptions of the Stochastic Dynamic Programming, the solution is supposed to be optimal on the random process which is exactly equal to the scenario tree.

A classical approach for building scenario trees is [37] where the scenario tree is built from an archive of data. However, nothing prevents Stochastic Dynamic Programming from *overfitting*, i.e. a poor generalization of performance, from the events modeled in the scenario tree to the original random process. For these reasons, *cross-validation* makes sense: build a scenario tree on a randomly selected subset of the data (e.g. 9/10th of the data) and test the policy on the remaining 1/10th. This can be reproduced multiple times; this is so-called random 10-fold cross-validation.

3.6 Other Methods

Many other resolution methods exist for Unit Commitment problems. Among them: *Priority list*, *Linear Programming* and *Mixed Integer Linear Programming*, *Lagrangian Relaxation* methods, *Network Flow Programming*, *Genetic algorithms* and other meta-heuristics. Some of these methods are very classic in the Unit Commitment community. See [13, 14] for a more comprehensive list of resolution methods (and their bibliography) for Unit Commitment problems.

Part III

Contributions

Chapter 4

Direct Value Search: a new algorithm combining critic and actor methods

4.1 Introduction

Direct Policy Search (DPS) (section 2.3) is a widely used tool for reinforcement learning; however, it is usually not suitable for handling high-dimensional constrained action spaces such as those arising in power system control (Unit Commitment problems). In this chapter, we propose Direct Value Search, an hybridization of DPS with *Bellman decomposition techniques* (sections 2.1, 3.2, 3.3 and 3.4). We prove runtime properties, and apply the results to an energy management problem.

4.1.1 Motivation

This chapter presents a new methodology for solving Sequential Decision Making problems (as *Markov Decision Processes* and *Non-Markovian Decision Processes*), combining the advantages of Direct Policy Search and the advantages of Bellman-style optimization. The proposed algorithm is aimed at solving problems with a large constrained action space.

In this chapter, a *Stochastic Decision Processes* corresponding to either *Markov Decision Processes* or *Non-Markovian Decision Processes* as our new method presented in this chapter is able to solve both indifferently.

The complexity of solving Stochastic Decision Process

When solving Stochastic Decision Process, at least two different runtime measures are relevant:

- **Solving runtime (SR):** this is the computation time (or complexity) for building the policy, given a Stochastic Decision Process. In other words, this is the *offline* computational cost. In Bellman decomposition techniques, it corresponds to the computation cost of computing the Bellman values V (e.g. Eq. 2.2).
- **Decision making runtime (DMR):** this is the computation time (or complexity) for making one decision with a given policy. In other words, this is the *online* computational cost. It typically depends on the family of policies we consider. In Bellman decomposition techniques, it corresponds to the computation cost of applying the principle of *Maximum Expected Utility* to compute actions knowing Bellman values V (e.g. Eq. 2.3).

The complexity of Stochastic Decision Process solving often refers to SR. A moderate computation cost for DMR can help reducing the complexity of SR: there are many cases in which solving the Stochastic Decision Process is based on solving many times the decision making problem; for instance, applying Bellman's operator in a *Value Iteration* framework involves many decision makings (see section 2.2). Also, *Policy Iterations* needs a lot of simulations, which in turn involve decision makings.

Problems with huge constrained action spaces

Our focus is on problems for which the DMR is huge, unless special tricks are used. Typically,

- we have to consider tenths of state variables;
- we have to make decisions for tenths of time steps. The number of decision variables and the number of constraints is then multiplied by the number of time steps considered for the optimization. This increases a lot the size of the problem.

Many important unit commitments problems have these features. For instance Unit Commitment problems with 80 hydroelectric stocks, 60 nuclear power plants, making decisions ensuring constraint satisfaction over 2 days (assuming that forecasts are reliable over 2 days) give rise to hundreds of thousands of decision variables for each decision making.

The power systems community has found a solution with a moderate DMR, thanks to a careful use of linear programming for approximating convex Bellman value functions by convex piecewise linear functions. This is classically done in Stochastic Dynamic Programming (SDP) [31] or Stochastic Dual Dynamic Programming [41]. However, it is well known that the SR as a function of the state space dimension is still exponential (so-called curse of dimensionality).

Advantages of Direct Policy Search

In our work, we will use DPS in a new manner so that we have a controlled SR, in spite of constrained action spaces. In addition, our method will have a polynomial DMR, similarly to SDP variants using linear approximations – as discussed in chapter 3.

DPS does not have such features. The principle of this algorithm is to optimize a parametric policy. Thanks to the use of a parametric policy, DPS has a moderate SR in spite of huge state spaces (i.e. it can be defined on continuous state spaces). With a bit of human expertise in the structure of this parametric policy, we can have highly efficient methodologies [36]. Robust numerical optimization (like *Evolution Strategies* for noisy optimization [46]) makes DPS efficient in cases where nothing else works. As already stated, the key advantages of DPS include:

- No need for a linear, quadratic, or analytical model; any problem which can be simulated can be tackled by DPS.
- No need for a decomposable criterion; we can optimize medians, quantiles, of delayed rewards.
- The simulations naturally focus on likely parts of the state space, without the tedious backward induction on all the state space as in backward Bellman-based tools.
- It can work with Non-Markovian Random Process.

The last point is a very strong advantage for power system applications but this advantage is often overlooked. Bellman decomposition requires a Markovian Random Process. However – for instance – water inflows to dam reservoirs in hydropower Unit Commitment problems are certainly neither independent nor Markovian. The only method to rigorously making inflows Markovian consists in extending the state representation with a memory of the previous states of the Random Process. This multiplies the size of the state space, making optimization intractable. DPS, on

the other hand, naturally tackles non Markovian random processes, in the sense that parametric policies are optimized on simulations which do not have to be Markovian.

Drawbacks of DPS, and strength of Bellman-style decomposition

There are also severe drawbacks of using DPS:

- A huge size of the action space implies, for many parametric policies, a huge size of the parameter space, leading to redhibitory SR.
- Constraints on the action space are difficult to handle in Direct Policy Search. In Unit Commitment problems, some actions are unfeasible from some given states; this is often difficult to handle with general purpose parametric policies like Neural Networks.
- There are many problems for which optimizing the short-term reward (incurred over the time steps for which we ensure constraint satisfaction) is a large part of the overall computational cost - precisely due to constraints. For such problems, independently of the state, using value-based methods such as Bellman decompositions is much easier than using DPS, because the optimization problem used in Bellman's operator can take all the constraints into account.

The third point is a key for understanding why DPS, as well as other policy-based reinforcement learning methods, are not widely tested in the power systems industry. In most Unit Commitment problems, the reward associated to an action contains, as main component, a complex instantaneous reward to be optimized under difficult constraints; such a situation is difficult to tackle with traditional DPS.

4.1.2 Direct Value Search Principle

The principle of DVS is to combine the representation of SDP and the optimization of DPS.

Our proposal: Direct Value Search

We propose Direct Value Search (DVS), a form of DPS based on the representation classically used in SDP or Stochastic Dual Dynamic Programming (SDDP). We will show that we get a polynomial bound on the DMR (as a function of the number of action variables and constraints), and an anytime algorithm for SR (i.e. an algorithm that smoothly and iteratively converges towards the optimum).

Stochastic Decision Processes.

We consider sequential decision making problems, in the Stochastic Decision Process framework (i.e. as stated before we consider Markov Decision Processes and Non-Markovian Decision Processes). This means that there is a transition function $(r_t, \boldsymbol{x}_{t+1}) = f(\boldsymbol{x}_t, \boldsymbol{\omega}_t, \boldsymbol{u}_t)$ where t indices are discrete time step indices ($t \in \{0, ..., T\}$), $r_t \in \mathbb{R}$ is a reward and $\boldsymbol{x}_t$ is a state ($\boldsymbol{x}_0$ is given). The controller decides its control $\boldsymbol{u}_t$ using some policy $u : \boldsymbol{u}_t = u(\boldsymbol{x}_t)$. The exogenous $\boldsymbol{\omega}_t$ is a random process - the $\boldsymbol{\omega}_t$ are not necessarily independent. This is why $\boldsymbol{u}_t$ only depends on u and $\boldsymbol{x}_t$ (more on this in Eq. 4.5). So we get the following equations[1]:

$$
\begin{aligned}
(r_t, \boldsymbol{x}_{t+1}) &= f(\boldsymbol{x}_t, \boldsymbol{\omega}_t, \boldsymbol{u}_t) && (4.1)\\
R &= \sum_{t \in \{0,...,T-1\}} r_t && (4.2)\\
\boldsymbol{\omega}_t &= \mathit{Random} && (4.3)\\
\boldsymbol{u}_t &= u(\boldsymbol{x}_t) && (4.4)
\end{aligned}
$$

Eq. 4.1 is the system equation, Eq. 4.2 gives the total reward, Eq. 4.3 is a random process and u (in Eq. 4.4) is the unknown strategy to be decided (T is the final time step).

Random processes (RP).

Several difficulties arise when difficult Random Processes are involved (some points have been discussed in chapter 1):

- A common crucial assumption is the independence or stagewise independence of the $(\boldsymbol{\omega}_t)_{t \in \{0,...,T-1\}}$. This is thoroughly analyzed in [43], in particular from the point of view of the (helpful) impact of this assumption on computational complexity.

- A more difficult case is when the $\boldsymbol{\omega}_t$ are a general Random Process and not an independent noise. Then, the past $\boldsymbol{\omega}_t$ (or the part of it which is known) should be encoded in the policy. Assuming that $\boldsymbol{\omega}_0, \ldots, \boldsymbol{\omega}_{t-1}$ (and only them) can be observed for deciding $\boldsymbol{u}_t$, Eq. 4.4 becomes:

$$\boldsymbol{u}_t = u(\boldsymbol{x}_t, \boldsymbol{\omega}_0, \ldots, \boldsymbol{\omega}_{t-1}). \quad (4.5)$$

[1] Note that, actions are now noted $\boldsymbol{u}$ and the policy at DMR is noted u in order to avoid mistakes with the SR policy noted π (Algorithm 8).

In a Bellman approach, the state space now includes $\boldsymbol{\omega}_0,\ldots,\boldsymbol{\omega}_{t-1}$, which is an increasing sequence with t.

- An intermediate case consists in assuming that the $\boldsymbol{\omega}_t$ are a Markov chain. Then, all the information is stored in the last $\boldsymbol{\omega}_t$:

$$\boldsymbol{u}_t = u(\boldsymbol{x}_t, \boldsymbol{\omega}_{t-1}). \tag{4.6}$$

  Unfortunately, such a case is rare. DPS has advantages for handling non-Markovian Random Process: the DPS algorithm is based on simulations, and it makes sense with any Random Process, Markovian or not. Non Markovian cases lead to more inputs (Eq. 4.5) but DPS does not have to sample an exponentially increasing state space.

- For representing non-Markovian Random Processes, common solutions [43] consist in either adding Random Process's past realisations in state (which leads to huge state spaces) or in considering finite scenario trees [37]. In the latter case, the non-Markovian nature of the Random Process is handled by adding an implicit state variable which is the node index.

- Tackling continuous Random Processes is required in many real-world applications. Yet, SDDP works with discrete Random Processes, hence the finite sample average approximation is usual [47, 48, 49, 50].

Our algorithm, Direct Value Search, uses Eqs. 4.1-4.3 and, as a special case of DPS. Eq. 4.4 and Eq. 4.5 can be applied without any adaptation (as usual with DPS variants). The function u is optimized so that the random variable R is approximately maximum.

Direct Value Search: Direct Policy Search for Optimizing the Valorization

Here we present the representation used in our algorithm, DVS. This representation is parametric; we optimize the parameters by non-linear noisy optimization. The non-linear noisy optimization tool we use is a *Self-Adaptive Evolution Strategy* (SA-ES), presented in Section 4.1.2.

Definition of DVS.

Direct Policy Search consists in keeping Eq. 3.5 (as in SDP methods), but not to compute $V(\boldsymbol{x}, t)$ by Eq. 3.8. Instead, we define V by $V(\boldsymbol{x}, t) = \Pi_{\boldsymbol{\theta}}(\boldsymbol{x}, t)$ and we optimize $\boldsymbol{\theta}$ by DPS.

To get lower computation time, we use a *tactical horizon* k (chosen so that we can assume that the k next values of the Random Process $\boldsymbol{\omega}_t$ can be deterministically approximated) as described in Section 3.1 and in Eq. 3.7. Thus we apply

$$u(\boldsymbol{x}, t) = \arg\max_{\boldsymbol{u}_t} \max_{\boldsymbol{u}_{t+1},\ldots,\boldsymbol{u}_{t+k-1}} \mathbb{E}\left(r_t + \cdots + r_{t+k}\right) + \Pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t, \boldsymbol{x}_{t+k-1}, t) \tag{4.7}$$

where

- $\Pi_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}_{t+k-1}, t)$ is a non-linear function of t and $\boldsymbol{x}$ (so that complex controls can be encoded);
- $\Pi_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}_{t+k-1}, t)$ is convex piecewise linear as a function of $\boldsymbol{x}_{t+k-1}$ so that Eq. 4.7 is a linear programming problem if all the $r_{t'}$ can be encoded in a linear program where the decision variables are the $\boldsymbol{u}_t$'s.

A convenient case is as follows:

$$u(\boldsymbol{x}, t) = \arg\max_{\boldsymbol{u}_t} \max_{\boldsymbol{u}_{t+1},\ldots,\boldsymbol{u}_{t+k-1}} \mathbb{E}\left(r_t + \cdots + r_{t+k}\right) + \Pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)^\top \cdot \boldsymbol{x}_{t+k-1} \tag{4.8}$$

In such a setting we benefit from

- the large-scale possibilities of linear programming in terms of size of the action space (provided that $\boldsymbol{u}_t \mapsto (r_t, \boldsymbol{x}_{t+1})$, for a given $\boldsymbol{x}_t$, can be encoded in a linear problem)
- and the genericity of Direct Policy Search.

From now on, we define a trajectory as a randomized sequence $(target_0, target_k, target_{2k}, \ldots, target_T)$ of states, possibly correlated with the random processes $(\boldsymbol{\omega}_0, \ldots, \boldsymbol{\omega}_{T-1})$. A trajectory $(target_0, \ldots, target_T)$ is reachable if it is possible, for some control functions, to ensure with probability 1 that, for all t, $\boldsymbol{x}_t = target_t$.

Properties of DVS.

More formally, the proposed approach satisfies the following properties:

- **DMR-tractability:** If the reward and new state $(r_t, \boldsymbol{x}_{t+1}) = f(\boldsymbol{x}_t, \boldsymbol{\omega}_t, \boldsymbol{u}_t)$ can be expressed with linear programming and if $\boldsymbol{x} \mapsto \Pi_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}_{t+k})$ is convex piecewise linear, then decision making (Eq. 4.7) can be achieved in polynomial time.

- **Genericity:** If the reward r_t is Lipschitz (with Lipschitz constant $l < \infty$) as a function of the decisions, then any reachable trajectory $(target_0, \ldots, target_T)$ can be forced by some function $\Pi_{\boldsymbol{\theta}}$ such that $\forall \boldsymbol{x}, t$, the function $\boldsymbol{x} \mapsto \Pi_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}_{t+k-1}, t)$ is convex piecewise linear with at most $d+1$ linear cuts, where d is the dimension of the state space.

Proof.

Just set $\Pi_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}_{t+k-1}, t) = 2l \times ||\boldsymbol{x}_{t+k-1} - target_{t+k-1}||$ (with l the Lipschitz constant for r_t) for showing the result with $2d$ cuts. Optimizing the placement of linear cuts so that (i) there is only one maximum $target_{t+k}$ and (ii) the gradient is lower bounded by a big enough constant almost everywhere leads to the same result with $d+1$ cuts.

□

We term our algorithm *Direct Value Search* (DVS) because it uses DPS for optimizing the valorization V by Direct Policy Search.

Evaluate the complexity for SR is more complicated. Thanks to the property above, the SR problem boils down to

- the noisy optimization (noisy because there are Random Processes involved)
- of a polynomial time simulator (in the number of time steps, actions, state variables, and data for defining the problem).

Noisy optimization has complexity roughly squared ($O(1/\epsilon^2)$) as a function of the requested precision ϵ over the expected reward, when the objective function is approximately convex (see [51] for related formal definitions, which indeed do not require strict convexity). This leads to a bound on the expected time for SR

$$O(T \times polynomial(k, \text{MDP size})/\epsilon^2).$$

However, this is the complexity for finding a solution for the chosen family of functions; if the parametric function $\boldsymbol{\theta} \mapsto \Pi_{\boldsymbol{\theta}}$ can not do better than a precision $\delta > 0$, no convergence to the optimal expected reward can be reached.

Self-Adaptive Evolution Strategy

Algorithm 7 presents the Self-Adaptive Evolution Strategy (SA-ES) we use to optimize the parameter $\boldsymbol{\theta}$ of the parametric policy $\Pi_{\boldsymbol{\theta}}$. This Self-Adaptive Evolution Strategy is inspired by [52] and uses [53]'s resampling principles for ensuring convergence in spite of noise.

Algorithm 7 Self-Adaptive Evolution Strategy with reevaluations. $\mathcal{N}$ denotes some independent standard Gaussian random variable, with dimension d as required in equations above.

Parameters: $K > 0$, $\zeta \geq 0$, $\lambda \geq \mu > 0$, a dimension $d \in \mathbb{N}^*$
Input: an initial parent population $\boldsymbol{x}_{1,i} \in \mathbb{R}^d$ and an initial $\sigma_{1,i} = 1$, $i \in \{1, \ldots, \mu\}$

$n \leftarrow 1$
while (stop condition) **do**
  Generate λ individuals i_j, $j \in \{1, \ldots, \lambda\}$, independently using

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

  and

$$i_j = \boldsymbol{x}_{n,mod(j-1,\mu)+1} + \sigma_j \mathcal{N}$$

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

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

  where $\mathbb{E}_m$ denotes a sample average over m resamplings.
  Update: compute $\boldsymbol{x}_{n+1,k}$ and $\sigma_{n+1,k}$ using

$$\sigma_{n+1,k} = \sigma_{j_k} \quad \text{and} \quad \boldsymbol{x}_{n+1,k} = i_{j_k}, \ k \in \{1, \ldots, \mu\}$$

  $n \leftarrow n + 1$
end while

Actually, many other non-linear noisy optimization algorithms could be used for $\Pi_{\boldsymbol{\theta}}$'s optimization (e.g. stochastic Newton's method, other ES like CMA-ES [54] or Fabian's method [51]). For a first attempt, we have chosen SA-ES as it as shown good results for similar tasks and it is easy to parallelize, implement and setup. In further work, we will consider other non-linear noisy optimization algorithms and compare their performance in our DVS test cases.

4.1.3 Detailed algorithm

Step 1 (offline): compute $\pi_{\boldsymbol{\theta}^*}(.)$, i.e. optimize the parameter $\boldsymbol{\theta}$

Given a parametric policy $\pi_{\boldsymbol{\theta}}$, get $\boldsymbol{\theta}^*$ maximising some reward function on simulations. See Alg. 8. Simulations are described in Alg. 9.

Algorithm 8 Optimization of DVS parameters (offline SR) with a DPS approach.

Input:
a parametric policy $\pi_{\boldsymbol{\theta}}(.)$ where $\pi_{\boldsymbol{\theta}}$ is a mapping from $\mathcal{X}$ to $\mathcal{U}$,
a Markovian or Non-Markovian Decision Process MNMDP,
an initial state $\boldsymbol{x}$

Output:
a parameter $\boldsymbol{\theta}^*$ leading to a policy $\pi_{\boldsymbol{\theta}^*}(.)$

Find a parameter $\boldsymbol{\theta}^*$ maximising the expectation of the following fitness function:

$$\boldsymbol{\theta} \mapsto \text{Simulate}(\boldsymbol{x}, \text{MNMDP}, \pi_{\boldsymbol{\theta}})$$

with a given non-linear noisy optimization algorithm (e.g. SA-ES, CMA-ES, ...)

return $\boldsymbol{\theta}^*$

Step 2 (online): use $\pi_{\boldsymbol{\theta}^*}(.)$ to solve the actual problem

The offline optimization (finding $\boldsymbol{\theta}^*$ approximately maximizing $\text{Simulate}(\boldsymbol{x}, \text{MNMDP}, \pi_{\boldsymbol{\theta}})$) can be stopped at any time. Then:

- using $\pi_{\boldsymbol{\theta}^*}$, we get $\pi_{\boldsymbol{\theta}^*}(\boldsymbol{x})$ an approximation of the stock marginal value at time $t + k$;

Algorithm 9 Performing a simulation with DVS (online DMR).
Notations:
$\boldsymbol{n}_f$ is the normalisation factor of α (ie. maximum valorization values),
$\boldsymbol{x}^+$ is the state vector with additional information (time factors described below, ...),
$\boldsymbol{u}_{t...k} := (\boldsymbol{u}_t, \ldots, \boldsymbol{u}_{t+k-1})^\top$ is the tuple of decisions from current stage to tactical,
$\boldsymbol{\omega}_{t...k} := (\boldsymbol{\omega}_t, \ldots, \boldsymbol{\omega}_{t+k-1})^\top$ is the tuple of random realizations from current stage to tactical,
$\boldsymbol{u}_{t...h} := (\boldsymbol{u}_t, \ldots, \boldsymbol{u}_{t+h-1})^\top$ is the tuple of decisions from current stage to operational
$\boldsymbol{\omega}_{t...h} := (\boldsymbol{\omega}_t, \ldots, \boldsymbol{\omega}_{t+h-1})^\top$ is the tuple of random realizations from current stage to operational.

Simulate($\boldsymbol{x}_0$, MNMDP, $\pi_{\boldsymbol{\theta}}$)
Input:
an initial state $\boldsymbol{x}_0$,
a Markovian or Non-Markovian Decision Process MNMDP,
a parametric policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{x}) \mapsto \boldsymbol{\alpha}$ with $\boldsymbol{\alpha} \in [0, 1]$
Output:
a reward R associated to the policy $\pi_{\boldsymbol{\theta}}$

$\boldsymbol{x}_0 \leftarrow$ initial_state
$R \leftarrow 0$
for $t \leftarrow t_0,\ t_0 + h,\ t_0 + 2h,\ ...,\ T$ **do**
  $\boldsymbol{\omega}_{t...k} \leftarrow$ get_forecast(.)

  if $t + k - 1 < T$ **then**
    $\boldsymbol{\alpha} \leftarrow \pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t^+)$
    $\boldsymbol{\alpha}_s \leftarrow \text{rescale}(\boldsymbol{\alpha}, \boldsymbol{n}_f)$
    $\boldsymbol{u}_{t...k} \leftarrow \arg\max_{\boldsymbol{u}} \text{reward}(\boldsymbol{u}_{t...k},\ \boldsymbol{\omega}_{t...k},\ \boldsymbol{x}_t) \ + \ \boldsymbol{\alpha}_s^\top \cdot \boldsymbol{x}_{t+k-1}$
  else
    $\boldsymbol{u}_{t...k} \leftarrow \arg\max_{\boldsymbol{u}} \text{reward}(\boldsymbol{u}_{t...k},\ \boldsymbol{\omega}_{t...k},\ \boldsymbol{x}_t)$
  end if

  $R \leftarrow R + \text{MNMDP\_reward}(\boldsymbol{x}_t,\ \boldsymbol{u}_{t...h},\ \boldsymbol{\omega}_{t...h})$
  $\boldsymbol{x}_{t+h} \leftarrow \text{MNMDP\_transition}(\boldsymbol{x}_t,\ \boldsymbol{u}_{t...h},\ \boldsymbol{\omega}_{t...h})$
end for

return R

- we can use it to solve the actual problem, the same manner as in the *Simulate* procedure (Algorithm 9).

In Algorithm 9 $\arg\max_{\boldsymbol{u}}$ is solved with linear programming solvers and we write $\pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t^+)$ instead of $\pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ as we consider time factors are included in $\boldsymbol{x}_t^+$.

4.2 Experiments

We experiment DVS as described in Eq. 4.7 in the following case:

- $\Pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t, \boldsymbol{x}_{t+k-1}, t) = \pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)^\top \cdot \boldsymbol{x}_{t+k-1}$, i.e. $\Pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t, ., t)$ is linear;
- $\pi_{\boldsymbol{\theta}}(.,.)$ is a neural network, with two layers of weights, a sigmoid activation function, and N neurons on the hidden layer.

We optimize $\boldsymbol{\theta}$ by maximizing $\boldsymbol{\theta} \mapsto \text{Simulate}(\boldsymbol{\theta})$ with a (μ, λ) Self-Adaptive Evolution Strategy (Algorithm 7). We use parameters $K = 10$, $\zeta = 2$, $\lambda = 4d+4$ and $\mu = \lceil \lambda/4 \rceil$ where d is the dimension of $\boldsymbol{\theta}$.

4.2.1 Settings

We work on a problem defined as follows:

- We have a 10-dimensional energy stock (made of 10 batteries).
- We have a random market price (which is a random generator, non-Markovian, non-discretized; it can generate arbitrarily many scenarios).
- 168 time-steps are considered (i.e. 7 days with one hour per time step).
- Operational horizon $h = 5$ time steps of 1 hour (each decision making covers 5 hours – see Fig. 3.3 in chapter 3).
- Tactical horizon $k = 5$ time steps of 1 hour (we use Eq. 4.7 with $k = 5$, i.e. 5 hours – see Fig. 3.3 in chapter 3).
- As we already said, $\Pi_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}_{t+k-1}, t) = \pi_{\boldsymbol{\theta}}(\boldsymbol{x}, t)^\top \cdot \boldsymbol{x}_{t+k-1}$ where $\pi_{\boldsymbol{\theta}}(.,.)$ is a neural network with N neurons in a single hidden layer, and weights vector $\boldsymbol{\theta}$. The number of parameters (i.e. the size of $\boldsymbol{\theta}$) to be optimized by the numerical optimization (presented in section 4.1.2) is $N(|\boldsymbol{x}|+1)+|\boldsymbol{u}|(N+1)$. Thus there are 35 parameters to optimize for $N = 1$ and 60 for $N = 2$.

- Inputs (i.e. $\boldsymbol{x}^+$ above) of the neural network are the ten battery levels plus four handcrafted time-dependent auxiliary inputs:

$$\begin{aligned}
\text{hour_of_day_cos_factor} &= \cos\left(2\pi\ (t+k-1)/24\right) \\
\text{hour_of_day_sin_factor} &= \sin\left(2\pi\ (t+k-1)/24\right) \\
\text{hour_of_week_cos_factor} &= \cos\left(2\pi\ \frac{\lfloor (t+k-1)/24 \rfloor}{7}\right) \\
\text{hour_of_week_sin_factor} &= \sin\left(2\pi\ \frac{\lfloor (t+k-1)/24 \rfloor}{7}\right)
\end{aligned}$$

These extra information are more relevant than the raw time index as the market price oscillate in a daily basis and in a weekly basis (therefore decisions might be correlated with these two informations).

This problem is depicted in Fig. 4.1. The 10 batteries are managed to stock 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 sell for each battery) in order to maximize profits (equivalently minimize costs). For each battery, the maximum input and output capacity is 1000 kW and the maximum storage capacity is 96000 kWh. Batteries have different efficiencies (the percentage of energy actually sold for a given output command) and a smart control has to take it into account: the "output efficiency" of battery i is $0.8 - \max(\frac{\lfloor i/2 \rfloor}{10}, 0.4)$.

4.2.2 Baselines

The baselines used for comparison are:

$$u(\boldsymbol{x}, t) = \arg\max_{\boldsymbol{u}_t} \max_{\boldsymbol{u}_{t+1},\ldots,\boldsymbol{u}_{t+k-1}} \mathbb{E} r_t + \cdots + r_{t+k-1} \tag{4.9}$$

and

$$u(\boldsymbol{x}, t) = \arg\max_{\boldsymbol{u}_t} \max_{\boldsymbol{u}_{t+1},\ldots,\boldsymbol{u}_{t+k-1}} \mathbb{E} r_t + \cdots + r_{t+k-1} + \boldsymbol{\alpha}^\top \cdot \boldsymbol{x}_{t+k-1} \tag{4.10}$$

where function u is defined with a null valorization at tactical horizon in the first case (Eq. 4.9) and with a constant marginal reward in the second case (Eq. 4.10). In Eq. 4.10, $\boldsymbol{\alpha}$ is a constant vector $(a, a, a, \ldots, a)$, with a a constant value optimized numerically.

We then reproduce the results (we reproduce both learning and test), after a rescaling of the outputs of the neural network to reasonable values (so that the valorization of each battery ranges from 0 to 40 euros per kilo watts). These additional results are presented in Table 1 and in Fig. 4.3.

4.2.3 Results

Results are presented in Fig. 4.2. With DVS, we get better results than the two baseline methods and after rescaling, we obtain a faster convergence.

Then, for an equivalent DMR, the improvement compared with null valorization and constant marginal valorization is respectively +80% and +18% for DVS using 1 neuron on the hidden layer of $\pi_{\boldsymbol{\theta}}$ and +96% and +34% for DVS using 2 neuron on the hidden layer of $\pi_{\boldsymbol{\theta}}$. These small neural networks are sufficient for the required policy.

Table 1 and Fig. 4.3 show that DVS's valorizations are relevant and give good results. We can see that DVS wisely uses the state information for valorization (i.e. for computing the stock valorization $\Pi_\theta(x_t, x_{t+k-1}, t)$ in Eq. 4.8); results are clearly better than our baseline stateless valorization methods. This validates the DVS's proof of concept. We will check in further work if DVS can take advantage of its nice properties to efficiently solve very large scale problems (some being unreachable so far).

Quantitatively comparing DVS and SD(D)P or DPS is tricky as assumptions are not the same: S(D)DP requires Markovian Random Process but here we use a general Random Process in our test case and DPS would not comply with the decision constraints unless we make a handcrafted test case specific parametric policy.

Comparing DVS to each of them according to their own assumptions could be considered for further work but DVS's advantages (no simplified model, no assumption on Random Process, no convexity constraints, many stages, big spaces) would be lost.

4.3 Conclusion

We have proposed a variant of Direct Policy Search, as a merge of SDP and DPS. The resulting algorithm, termed DVS, has the following properties:

- DVS can handle large and/or constrained action spaces; there are 240 decisions, with constraints, in our problem and we can expect much more in further experiments.

- DVS is completely anytime (i.e. the approach can provide results after a few seconds even if convergence is incomplete - an important property when the problem is part of an investment optimization where many scenarios must be compared).

- It is highly parallel, because it boils down to a highly parallel noisy optimization problem.
- It can encode non-linear policies, indirectly, without losing the polynomial time DMR, as long as the decision making can be encoded as a linear problem. The model can be arbitrarily complex and can include small time scale effects without increasing the state space, whereas SDP-based methods have a precision loss when the number of time-steps increases.
- DPS does not need convex Bellman values (whereas SDDP does). Stock-dependent efficiencies in pumped-storage systems is not a problem.
- ADP, SDP and SDDP work with Markovian Random Processes, which is not a requirement for DVS. DVS can be applied on a problem where Random Processes are continuous and not Markovian. This is relevant for climate data.
- Stopping criteria can be defined for SDDP [48, 41] or more generally SDP methods. These stopping criteria are, by nature, asymptotic, as well as stopping criteria of noisy optimization algorithms involved in our DVS method.

Further work. We tested Direct Value Search on a simple problem, for which it has given satisfactory results. Our test case is not that much a toy problem: we use a random generator without discretization, which is a challenge for many algorithms. Our approach is, by design, less sensitive to dimension than classical methodologies and does not require any convexity assumption. Therefore Direct Value Search should be tested on very high scale problems (hundreds of stocks and thousands of time steps), showing its ability to handle non-Markovian Random Processes, non-convex Bellman values, high-scale state space.

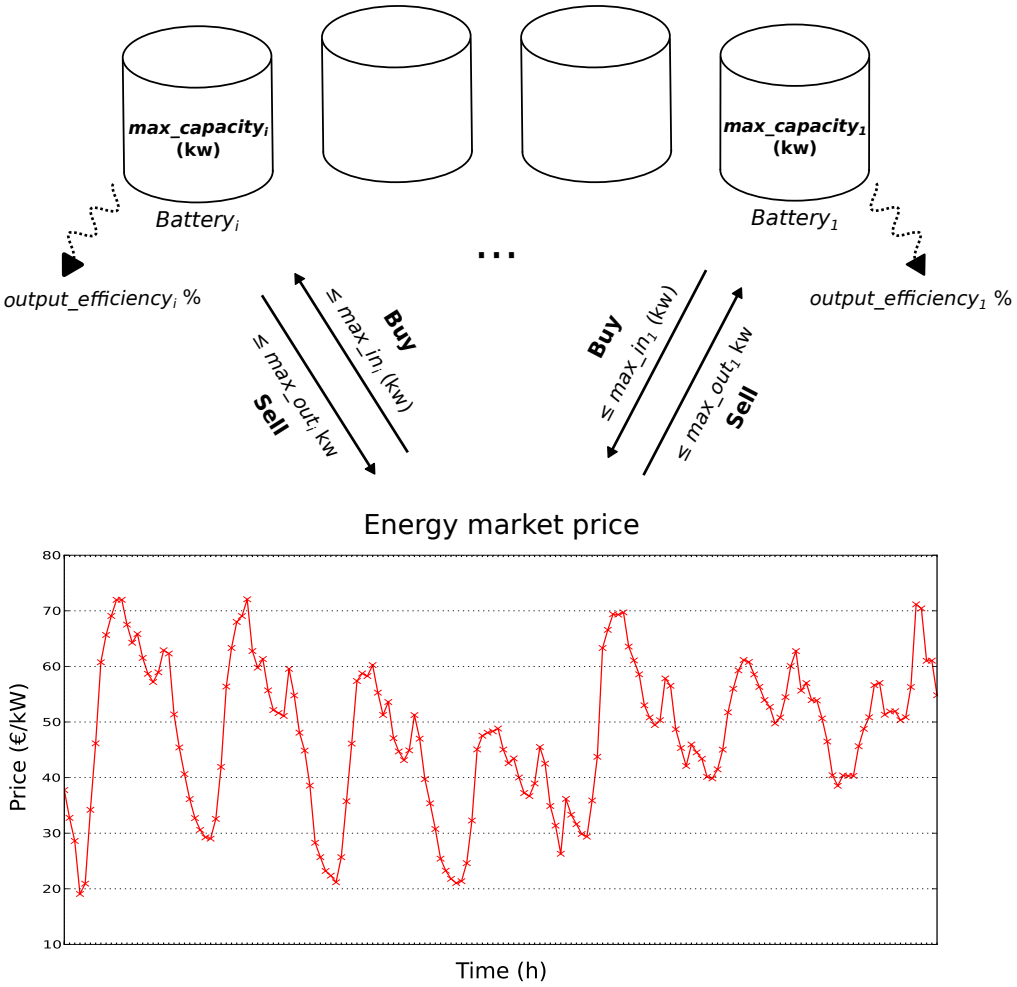

Figure 4.1: Illustration of the considered problem (with a realisation example of the random market price). Here, max_in_i is the maximum quantity of energy (in kilowatt) that can be input at any time in the i^{th} battery; max_out_i is the maximum quantity of energy (in kilowatt) that can be output at any time in the i^{th} battery; max_capacity_i is the maximum capacity (in kilowatt hour) of the i^{th} battery; $\text{output\_efficiency}_i$ is the percentage of energy actually output of the i^{th} battery for a given output command.

Constant valorization

Parameters	Mean reward (M€)
$a = 0$	1.082 ± 0.018
$a = 5$	1.104 ± 0.020
$a = 10$	1.292 ± 0.025
$a = 15$	1.755 ± 0.032
$a = 20$	2.205 ± 0.049
$a = 25$	1.959 ± 0.085
$a = 30$	0.390 ± 0.143
$a = 35$	-2.315 ± 0.216
$a = 40$	-6.095 ± 0.264
$a = 45$	-10.44 ± 0.291
$a = 50$	-15.90 ± 0.299
$a = 55$	-20.81 ± 0.282
$a = 60$	-25.83 ± 0.257
$a = 65$	-30.37 ± 0.228

Direct Value Search

Optimizer	Mean reward (M€)
SAES 1 neuron	2.524 ± 0.004
SAES 2 neurons	2.815 ± 0.004

Table 4.1: Comparison of mean results obtained with DVS and with constant marginal valorization (with valorization $\boldsymbol{\alpha} = (a, \ldots, a)$). The case $a = 0$ corresponds to the null valorization at tactical horizon. These results are the mean value of 2000 evaluations (standard deviation is also given). The mean execution time per evaluation is 1.5 seconds. Computations have been done on an Intel Xeon X5660 CPU (at 2.80GHz) with 49GB RAM. DVS results are given for policies trained with 250000 evaluations (i.e. 250000 calls to the Simulate function).

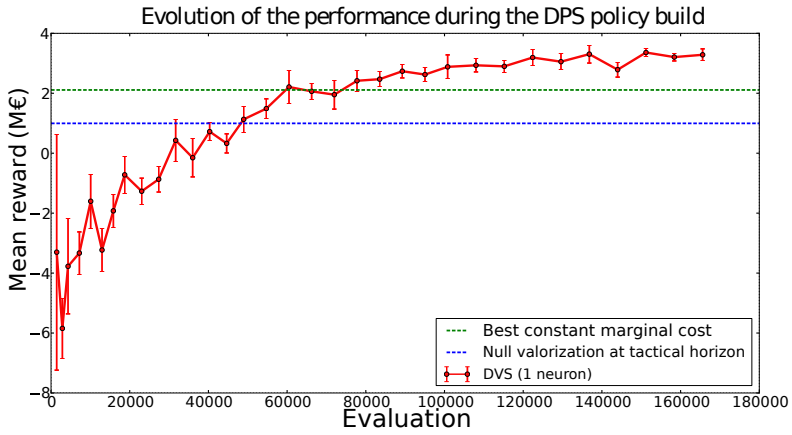

Figure 4.2: DVS's results with 1 neuron, using "5 time-steps aheard" optimization (Eq. 4.7 with $k = 5$), compared to the simple "5 time-steps ahead" optimization without valorization (Eq. 4.9) and to a constant marginal valorization of the battery at an optimal price (euros/MWh) with the same "5 time-steps ahead" context (Eq. 4.10, where $\boldsymbol{\alpha}$ is a constant). Error bars show ± standard deviation. Here, printed results are rewards (i.e. the higher the better).

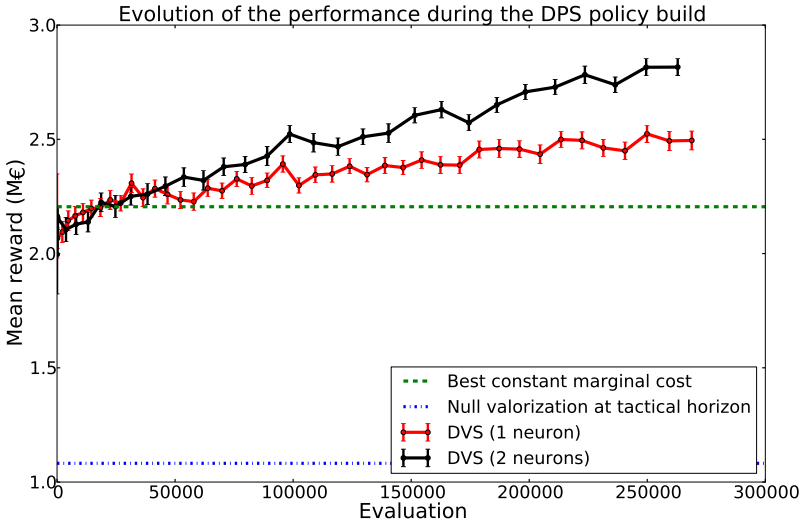

Figure 4.3: Experimental setup as in Fig. 4.2 with rescaled outputs. We also compare 1 neuron and 2 neurons neural networks. The baselines in the comparison are the same as in Fig. 4.2. Comparing this to Fig. 4.2, we see that rescaling the output values to reasonable values makes the convergence faster.

Chapter 5

Noisy Optimization

5.1 Motivation

Noisy optimisation is a key component of our Direct Value Search algorithm. It is, more generally, a crucial component of Direct Policy Search. In such frameworks, evolution strategies are widely used. The mathematical theory of evolution strategies has the following limitations:

- there exist few results in the noisy optimization setup;
- all existing results are based on very simple functions, typically sphere-like functions.

In Section 5.2, we extend the theory of evolution strategies in the direction of noisy optimization. In particular, we show lower bounds on runtimes. We also perform experiments on some classical variance reduction methodologies (Section 5.3); we design some deceptive functions (showing cases in which the pairing methodology does not work), but get empirically positive results. Then, the next chapter will study difficult functions, with no quasi-convexity assumptions.

Some terms used in this chapter and the following one are briefly explained in Appendix A.

5.2 Lower bounds on runtimes

5.2.1 Introduction

Considered framework

There are many convergence rates known in numerical optimization, depending on assumptions (derivative-free [55] or not, comparison-based [56] or not, global [57] or not). Robustness to noise, even early in the origins of evolutionary computation [58], is cited as a strength of these algorithms. In spite of this strong interest for noise in evolutionary computation, complexity in the noisy case is less clear, because details on the assumption have a big impact on the performance [59, 60, 61, 62, 63, 64]; a main distinction being adaptive noise [65] (with small noise variance around the optimum) and additive noise [66] (with lower-bounded variance around the optimum). Also, defining convergence rates is more difficult when either the algorithm or the objective function has a random part, because the distance of $\boldsymbol{x}_n$ to the optimum $\boldsymbol{x}^*$ is not a constant (n being the n^{th} objective function evaluation). Various convergence rates can be defined depending on the precise considered definition of convergence (we will not give too many details on this in this introduction, but our results will state precisely the definitions involved).

For example, the algorithm CLOP [67, 68] reaches a convergence $\|\boldsymbol{x}_n - \boldsymbol{x}^*\| = \tilde{O}(1/\sqrt{n})$ on a wide range of symmetric objective functions, using regression; in a very structured case (when the family of functions is very well approximated by a known model), statistical tools (supervised machine learning) provide such fast rates, even on very flat functions.

Another example of algorithm for noisy optimization is R-EDA (Racing-based Estimation of Distribution Algorithm, [69, 70]). It considers a different definition of convergence (more on this later). R-EDA was proposed as a typical noisy optimization algorithm, easy to analyze and making a good approximation of real-world approaches [71]. The noisy optimization framework is described in Algorithm 10 and R-EDA is defined in Algorithm 11. For $\beta > 0$, the convergence rate is typically $\|\boldsymbol{x}_n - \boldsymbol{x}^*\| = \tilde{O}(1/n^{1/\beta})$ on the family of objective functions

$$F_{\beta,\gamma} = \{f_{\boldsymbol{x}^*,\beta,\gamma}(\boldsymbol{x}); \boldsymbol{x}^* \in D\}\,, \tag{5.1}$$

where $D = [0,1]^d$ and

$$f_{\boldsymbol{x}^*,\beta,\gamma}(\boldsymbol{x}) = \mathcal{B}\left(\gamma\left(\frac{\|\boldsymbol{x} - \boldsymbol{x}^*\|}{\sqrt{d}}\right)^{\beta} + (1-\gamma)\right). \tag{5.2}$$

Here $\mathcal{B}(p)$ refers to a Bernoulli random variable with parameter p (i.e. equal to 1 with probability p and 0 otherwise). Fig. 5.1 show the expected values of $f_{\boldsymbol{x}^*,\beta,\gamma}$ with respect to $\boldsymbol{x}$ for some set of parameters.

We will show in this section that R-EDA is optimal within logarithmic factors for $F_{\beta,\gamma}$ under locality assumptions discussed below and for some values of γ (case with variance linearly decreasing as a function of expected fitness values).

Algorithm 10 Noisy optimization framework. This is not an optimization algorithm; this just explains the framework of noisy optimization with Bernoulli random variables (the fitness function outputs 1 if random ($= \omega_n$) is less than $\mathbb{E}f_{\boldsymbol{x}^*,\beta,\gamma}(\boldsymbol{x}_{\boldsymbol{x}^*,n,\omega,\omega'})$). *Optimize* is an optimization algorithm taking as input a sequence of visited points, their binary noisy fitness values and an internal noise. It outputs a new point to be visited, looking for points $\boldsymbol{x}$ of the domain such that $f_{\boldsymbol{x}^*,\beta,\gamma}(\boldsymbol{x})$ is as small as possible.

Input:
ω the uniform noise of f,
ω' a random seed of the algorithm,
$\boldsymbol{x}^*$ the optimal point,
β and γ two fixed parameters of f.
Output:
$\boldsymbol{x}_{\boldsymbol{x}^*,n,\omega,\omega'}$ the estimation of the optimum.

for all $n = 1, 2, 3, \ldots$ **do**
  $\boldsymbol{x}_{\boldsymbol{x}^*,n,\omega,\omega'} = \text{Optimize}(\boldsymbol{x}_{\boldsymbol{x}^*,1,\omega,\omega'},\ \ldots,\ \boldsymbol{x}_{\boldsymbol{x}^*,n-1,\omega,\omega'},\ y_1,\ \ldots,\ y_{n-1},\ \omega')$
  if $\omega_n \leq \mathbb{E}f_{\boldsymbol{x}^*,\beta,\gamma}(\boldsymbol{x}_{\boldsymbol{x}^*,n,\omega,\omega'})$ **then**
    $y_n = 1$
  else
    $y_n = 0$
  end if
end for
return $\boldsymbol{x}_{\boldsymbol{x}^*,n,\omega,\omega'}$

Symmetry assumptions and information theory

We pointed out above that a $\tilde{O}(1/\sqrt{n})$ rate of convergence is possible for some algorithms (e.g. CLOP) using models which are very close to the real objective function (for example if we know that the fitness function is a Bernoulli as in Eq.

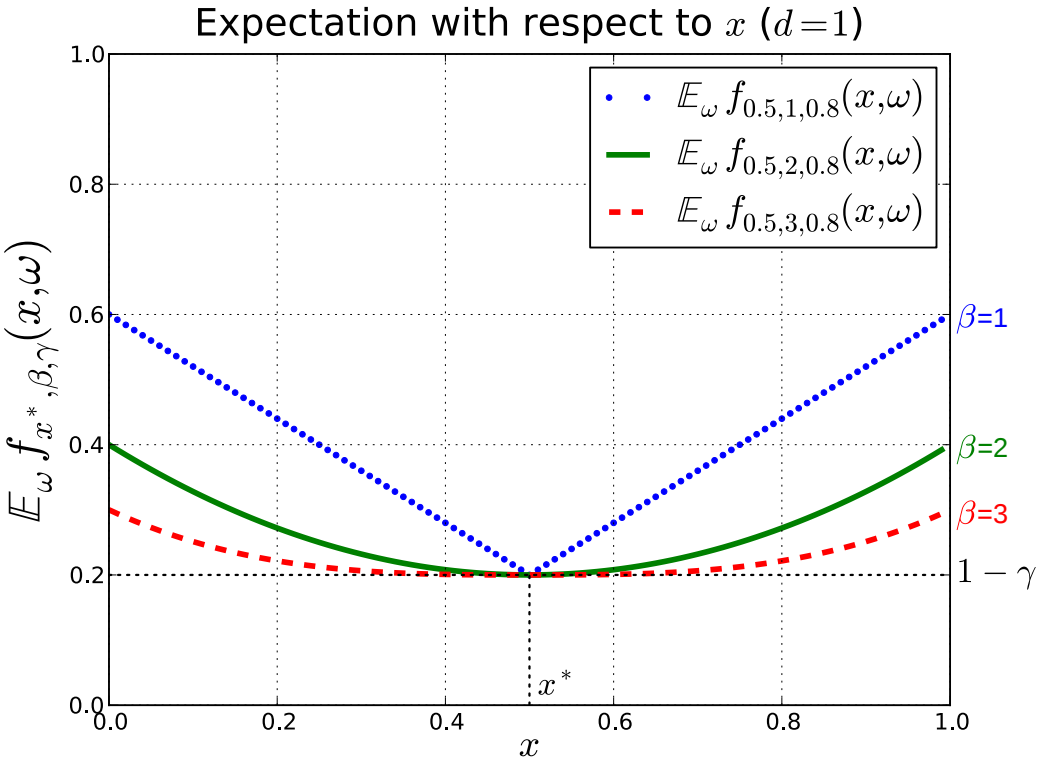

Figure 5.1: Expected values of $f_{\boldsymbol{x}^*,\beta,\gamma}$ with respect to $\boldsymbol{x}$, for $d = 1$ and with the optimal point $\boldsymbol{x}^* = 0.5$, the noise level $\gamma = 0.8$ and β (the "flatness" of $\mathbb{E}f$ around $\boldsymbol{x}^*$) equals to 1, 2 and 3.

Algorithm 11 R-EDA: algorithm for optimizing noisy fitness functions. *Bernstein* denotes a Bernstein race, as defined in Algorithm 12. The initial domain is $[\boldsymbol{x}_0^-, \boldsymbol{x}_0^+] \in \mathbb{R}^D$, δ is the confidence parameter. This algorithm goes back to [69, 70]. Please note that $\boldsymbol{x}_i^-$ and $\boldsymbol{x}_i^+$ are indexed by i, the iteration number, and not by the number of evaluations as in our convergence criteria.

$n \leftarrow 0$
while True **do**
  // Pick the coordinate with highest uncertainty
  $c_n = \arg\max_i (\boldsymbol{x}_n^+)_i - (\boldsymbol{x}_n^-)_i$
  $\delta_n^{\max} = (\boldsymbol{x}_n^+)_{c_n} - (\boldsymbol{x}_n^-)_{c_n}$
  for $i \in [[1, 3]]$ **do**
    // Consider the middle point
    $\boldsymbol{x'}^i_n \leftarrow \frac{1}{2}(\boldsymbol{x}_n^- + \boldsymbol{x}_n^+)$
    // The c_n^{th} coordinate may take 3 $\neq$ values
    $(\boldsymbol{x'}^i_n)_{c_n} \leftarrow (\boldsymbol{x}_n^-)_{c_n} + \frac{i-1}{2}(\boldsymbol{x}_n^+ - \boldsymbol{x}_n^-)_{c_n}$
  end for
  $(good_n, bad_n) = Bernstein(\boldsymbol{x'}^1_n, \boldsymbol{x'}^2_n, \boldsymbol{x'}^3_n, \frac{6\delta}{\pi^2(n+1)^2})$.
  // A good and a bad point
  Let H_n be the halfspace
  $\{\boldsymbol{x} \in \mathbb{R}^D; ||\boldsymbol{x} - good_n|| \leq ||\boldsymbol{x} - bad_n||\}$
  Split the domain: $[\boldsymbol{x}_{n+1}^-, \boldsymbol{x}_{n+1}^+] = H_n \cap [\boldsymbol{x}_n^-, \boldsymbol{x}_n^+]$
  $n \leftarrow n + 1$
end while

5.2). In these algorithms, the sampled point is not necessarily close to the optimum or to the current approximation of the optimum that the algorithm has. This is related to information theory: there are areas in which one gets more information than others, and points minimizing $\boldsymbol{x} \mapsto E_\omega f(\boldsymbol{x}, w)$ are not necessarily the most informative. Typically, when you have a relevant model of the objective function, you will learn more about the optimum by sampling maximum uncertainty points (i.e. points $\boldsymbol{x}$ such that $f(\boldsymbol{x}, \omega)$ has high variance), rather than by sampling points close to the optimum.

Algorithm 12 Bernstein race between 3 points. Eq. 5.3 is Bernstein's inequality to compute the precision for empirical estimates (see e.g. [72, p124]); $\hat{\sigma}_i$ is the empirical estimate of the standard deviation of point $\boldsymbol{x}_i$'s associated random variable $F_t(\boldsymbol{x}_i)$ (it is 0 in the first iteration, which does not alter the algorithm's correctness); $\hat{f}(\boldsymbol{x})$ is the average of the fitness measurements at $\boldsymbol{x}$.

Bernstein$(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \delta')$
$T = 0$
repeat
  $T \leftarrow T + 1$
  Evaluate the fitness of points $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$ once, *i.e.* evaluate the noisy fitness at each of these points.
  Evaluate the precision:

$$\epsilon_{(T)} = 3 \log \left(\frac{3\pi^2 T^2}{6\delta'} \right) / T + \max_i \hat{\sigma}_i \sqrt{2 \log \left(\frac{3\pi^2 T^2}{6\delta'} \right) / T}. \tag{5.3}$$

until Two points (*good* ,*bad*) satisfy $\hat{f}(bad) - \hat{f}(good) \geq 2\epsilon$ — **return** $(good, bad)$

There are therefore two distinct strategies[1]:

- sampling close to the current estimation of the optimum;
- sampling maximum uncertainty areas.

As we have already said, getting knowledge on the objective function far from the current approximation of the optimum does not help for finding $\boldsymbol{x}^*$ if you have no model of the objective function (at least in a setting without tricky local optima). But, if you have a strong prior on the objective function, you can indeed sample only very far from the current estimation of the optimum, at locations where the objective function is less flat and from this sampling, you get knowledge on $\boldsymbol{x}^*$. Again, this leads to algorithms which sample much more far from the current approximation of the optimum than close to it.

[1] Interestingly there were debates on the mailing list dedicated to the BBOB noisy optimization testbeds because its designers assess the quality of optimization algorithms not from their estimation on the location of the optimum, but from the points they sample, which is clearly not fair for algorithms which are precisely based on estimating the optimum from points far from the optimum.

These algorithms have two distinct steps:

- *Exploration*: choosing a point $\boldsymbol{x}_n$ for which they want to sample $f(\boldsymbol{x}_n)$.
- *Recommendation*: providing an estimate $\tilde{\boldsymbol{x}}_n^*$ of $\arg\min \mathbb{E}f$.

Nevertheless, various compromises are possible (for not relying too much on the model) between strategies relying only on maximum uncertainty (strongly trusting a model) and strategies relying only on sampling close to the estimation of the optimum; [73] is based on methods for choosing to which extent the model should be trusted.

So far, we have discussed the distinction between algorithms which sample close to the estimation of the optimum and those which sample at maximum uncertainty areas. But in this work we will only consider the fastest theoretical convergence rates for algorithms which sample close to the estimation of the optimum. Formally speaking, we assume the following *locality assumption* for some $0 < \delta < 1/2$ and some constant $C(d) > 0$ depending on d only,

$$\forall f \in F, \forall i \leq n, \|\boldsymbol{x}_i - \boldsymbol{x}^*\| \leq \frac{C(d)}{i^\alpha}, \tag{5.4}$$

with probability at least $1 - \delta/2$; $\alpha > 0$ large implies that there is a fast convergence. This equation depends on n; in fact, the whole work would make sense with n replaced by ∞, but we can show our results for any n sufficiently large, so we keep this assumption under this form (the main theorem will assume this for all n, but results in it are derived for n sufficiently large).

Actually, convergence rates could be formalized differently. For example, we might consider that the rate is α if

$$\forall f \in F, \forall n, \|\boldsymbol{x}_{k_n} - \boldsymbol{x}^*\| \leq \frac{C(d)}{{k_n}^\alpha}. \tag{5.5}$$

for some increasing sequence k_n. This is a weaker assumption, because only $\boldsymbol{x}_i$ such that $\exists n; i = k_n$ have a fast rate; other points can be sampled anywhere in the domain without modifying the measure α of the convergence rate. For instance, the following algorithm satisfies Eq. 5.5 but not Eq. 5.4, for $0 < \alpha < 1/2$:

- define $k_n = n^2$;
- if i is different from k_n for all n, then do *exploration* ($\boldsymbol{x}_i$ is uniformly drawn on the domain);

- otherwise, do *exploitation* ($\boldsymbol{x}_i$ is the maximum likelihood estimate of $\boldsymbol{x}^*$ given the $\boldsymbol{x}_i$ and their fitness evaluations).

Indeed, this algorithm is quite good for the objective function model that we have chosen (see Eq. 5.2); but it does not satisfy Eq.5.4 as some points are sampled far from the optimum. In fact the algorithm above can even be optimized by choosing $\boldsymbol{x}_i$, for exploration, at locations where it is most likely to help finding the optimum (this is active learning); see e.g. [74, 75]. Also, sometimes, the $\boldsymbol{x}_i$ for exploitation are computed, but not evaluated.

When designing a testbed for noisy optimization, this issue makes sense. Many optimization algorithms for noisy settings distinguish $\boldsymbol{x}_i$'s which are supposed to be good approximations of the optimum and $\boldsymbol{x}_i$'s which are sampled for gathering information about the optimum. If the testbed makes no difference between the two kinds of points, it implicitly assumes that sampling far from the optimum for gathering information is unlikely to be a good method. The rates reachable with no constraints are a well established part of the state of the art [51]; we here focus on rates which can be attained when focusing on Eq. 5.4 as a criterion for convergence. Importantly, this criterion is also relevant when all fitness values sampled are important; e.g. when improving, online, a production unit.

To sum up, the locality assumption (Eq.5.4) used to obtain our main result means that the algorithm has a given rate if and only if all its search points follow this rate; it is not allowed, for instance, to have one point out of two which is close to the optimum, and another far away in order to get some information which, for some reason, would help for the convergence. In other words, this assumption means that we consider rates that can be reached without relying on long distance correlations between fitness values. This is by no mean a negligible technical detail; as we have already said, there are fast algorithms which rely on sampling far from the optimum; these fast algorithms, however, can only be fast when there is a strong structure on the objective functions so that sampling far away can provide significant improvements on the convergence rate. This section is devoted to showing bounds on rates for algorithms which do not use such sampling "far" from the current estimate of the optimum.

The results can therefore be viewed with two different complementary conclusions:

- either as the proof that fast rates (faster than the limits obtained in our work) can only be obtained by sampling also far from the optimum;

- or as the proof that fast rates (faster than the limits obtained in our work) are only possible for algorithm which assume some strong "flatness" of the

objective function around the optimum, and these algorithms will not be so fast if we test them on other objective functions.

Under the locality assumption (Eq. 5.4), we show that for the family $F_{\beta,\gamma}$ of objective functions ($\gamma > 0$), α is necessarily less than or equal to $1/\beta$ - if the function is very flat around the optimum (β large), the convergence of algorithms sampling close to the optimum (Eq. 5.4) is necessarily slow ($\alpha \leq 1/\beta$ in Eq. 5.4). On the other hand, for $\beta = 2$, local algorithms (like R-EDA) have the same order as the rate reached by machine learning methods, and can even be better for $\beta = 1$, reaching $\tilde{O}(1/n)$ when $\beta = 1$ instead of $\tilde{O}(1/\sqrt{n})$ by max-uncertainty sampling.

Models and Lemmas

Algorithm 10 describes our framework. As introduced in section 5.2.1 (Eq. 5.4), we assume, for some $0 < \delta < 1/2$, the locality assumption

$$\forall f \in F, \forall i \leq n, \|\boldsymbol{x}_i - \boldsymbol{x}^*\| \leq \frac{C(d)}{i^\alpha}$$

with probability at least $1 - \delta/2$. This contains two important elements:

- there is an $\tilde{O}(1/n^\alpha)$ convergence to the optimum;
- there is no sampling far from the current estimate of the optimum.

This implies that the algorithm converges and does not sample far from its limit.

As already stated in Eq. 5.1 and 5.2, we also consider the family of functions

$$F = F_{\beta,\gamma} = \{f_{\boldsymbol{x}^*,\beta,\gamma}(\boldsymbol{x}) \ ; \ \boldsymbol{x}^* \in D\},$$

where

$$f_{\boldsymbol{x}^*,\beta,\gamma}(\boldsymbol{x}) = \mathcal{B}\left(\gamma\left(\frac{\|\boldsymbol{x}_n - \boldsymbol{x}^*\|}{\sqrt{d}}\right)^\beta + (1-\gamma)\right),$$

that is to say the random variable ω is uniform in $[0,1]$ and $f(\boldsymbol{x},\omega) = 1$ if and only if $\omega \leq \gamma\left(\frac{\|\boldsymbol{x}_n - \boldsymbol{x}^*\|}{\sqrt{d}}\right)^\beta + (1-\gamma)$ ($f(\boldsymbol{x},\omega) = 0$ otherwise).

Equation 5.2 and the locality assumption (Eq. 5.4) imply

$$\underbrace{\mathbb{E}f(\boldsymbol{x}^*)}_{1-\gamma} \ \leq \ \mathbb{E}f(\boldsymbol{x}_n) \ \leq \ \underbrace{\mathbb{E}f(\boldsymbol{x}^*)}_{1-\gamma} + \frac{\gamma}{d^{\beta/2}}\frac{C(d)^\beta}{n^{\alpha\beta}},$$

with probability at least $1-\delta/2$ and where $\mathbb{E}f(\boldsymbol{x})$ is a short notation for $\mathbb{E}_\omega f(\boldsymbol{x},\omega)$.

The n^{th} function evaluation y_n is, by definition, at $\boldsymbol{x}_{\boldsymbol{x}^*,n,\omega,\omega'}$ which depends on $\boldsymbol{x}^*$ (which specifies the objective function), ω (which is the sequence of random variables ω of the stochasticity of the objective function), ω' (which is the sequence of random choices within the optimization algorithm, which might be stochastic). It also depends on β and γ, but these will be considered as fixed.

For short, we will note $X_{n,\Omega}$, with $\Omega=(\omega,\omega')$ the set of all the $\boldsymbol{x}_{\boldsymbol{x}^*,n,\omega,\omega'}$ for all $\boldsymbol{x}^*$, that is to say $X_{n,\Omega}$ is the set of all points which can be chosen by the optimization algorithm *Optimize* for the n^{th} point to sample if we consider a given noise and internal randomness.

To obtain our main result, we first need the following combinatorial lemma :

Lemma 1 *The cardinality of $X_{n,\Omega}$ is at most 2^N, where N is the cardinality of*

$$\left\{1\leq i\leq n\ ;\ \underbrace{\mathbb{E}f(\boldsymbol{x}^*)}_{1-\gamma}\leq\omega_i\leq\underbrace{\mathbb{E}f(\boldsymbol{x}^*)}_{1-\gamma}+\frac{\gamma}{d^{\beta/2}}\frac{C(d)^\beta}{i^{\alpha\beta}}\right\}.$$

Proof $\boldsymbol{x}_n$ is deterministic as a function of Ω and of the fitness values; so the possible values of $\boldsymbol{x}_{\boldsymbol{x}^*,n,\omega,\omega'}$ only depend on the 2^N possible values of the y_i for i in the set above.

We also need the following

Lemma 2 *Let N be the cardinality of*

$$\left\{1\leq i\leq n\ ;\ \mathbb{E}f(\boldsymbol{x}^*)\leq\omega_i\leq\mathbb{E}f(\boldsymbol{x}^*)+\frac{\gamma}{d^{\beta/2}}\frac{C(d)^\beta}{i^{\alpha\beta}}\right\}.$$

Then, N has expectation at most

$$z=\frac{\gamma}{d^{\beta/2}}C(d)^\beta\sum_{i=1}^{n}i^{-\alpha\beta}\tag{5.6}$$

and variance also at most z.

which is a direct consequence of the definition of N.

Lemma 2 and Chebyshev's inequality [76, 77, 78] ensure the following lemma:

Lemma 3 *Consider $\delta\in[0,1]$. $N\leq z+\sqrt{z}\,(\delta/2)^{-1/2}$ with probability at least $1-\delta/2$.*

Lemmas 1 and 3 together imply that the cardinality of $X_{n,\Omega}$ is at most

$$2^{z+\frac{\sqrt{z}}{\sqrt{\delta/2}}}\tag{5.7}$$

with probability at least $1-\delta/2$.

5.2.2 Main results

Theorem 1 *Assume that F is as proposed in Eq. 5.2 for some $1 > \gamma > 0$ and $\beta > 0$. Assume that Eq. 5.4 (locality assumption) holds for all $n \geq 1, d \geq 1$, and for some $C(d)$, $\delta < 1$, $\alpha > 0$, that is to say*

$$\forall f \in F, \forall i \leq n, \|\boldsymbol{x}_i - \boldsymbol{x}^*\| \leq \frac{C(d)}{i^\alpha},$$

with probability at least $1 - \delta/2$. Then $\alpha \leq 1/\beta$.

Proof Let us show that $\alpha\beta \leq 1$. In order to do so, let us assume, in order to get a contradiction, that $\alpha\beta > 1$; then, knowing convergence of Riemann series for $\alpha\beta > 1$

$$\sum_{i=1}^{n} \frac{1}{i^{\alpha\beta}} \quad < \quad \frac{\alpha\beta}{\alpha\beta - 1}$$

equation 5.6 leads to:

$$z \leq \frac{\gamma C(d)^\beta}{d^{\beta/2}} \frac{\alpha\beta}{\alpha\beta - 1} \text{ if } \alpha\beta > 1 \tag{5.8}$$

Consider any optimization algorithm (stochastic or not). Eq. 5.8 implies the finiteness of z, and therefore by Eq. 5.7 the finiteness of $X_{n,\Omega}$, bounded above by a constant C independent of n, with probability at least $1 - \delta/2$.

We will here use sets of points with lower bounded distance to each other; such sets are classical in statistics [79], and are now also used for building lower bounds based on information theory [80, 81].

Consider R a set of cardinality C' such that

$$\frac{C'}{C} > \frac{1 - \frac{\delta}{2}}{1 - \delta} \tag{5.9}$$

and such that two distinct elements of R are at distance greater than 2ϵ, with $\epsilon = C(d)/n^\alpha$, from each other; such a set certainly exists if n is large enough. A nice property of this set is that if the optimum $\boldsymbol{x}^*$ is uniformly drawn in R, then it can only be found with probability $1 - \delta/2$ and with precision $C(d)/n^\alpha$ if $X_{n,\Omega}$ contains one point close to r for a proportion at least $1 - \delta/2$ of points $r \in R$.

Consider $f_{\boldsymbol{x}^*} = f_{\boldsymbol{x}^*,\beta,\gamma}$ with $\boldsymbol{x}^*$ uniformly distributed in R. Then:

$$\begin{aligned}
& P(\|\boldsymbol{x}_n - \boldsymbol{x}^*\| \leq \epsilon) \\
\leq\ & \mathbb{E}_\Omega P_{\boldsymbol{x}^*}(\boldsymbol{x}^* \in \mathrm{Enl}(X_{n,\Omega}, \epsilon)) \\
\leq\ & P(\#X_{n,\Omega} \leq C) P_{\boldsymbol{x}^*}(\boldsymbol{x}^* \in \mathrm{Enl}(X_{n,\Omega}, \epsilon) | \#X_{n,\Omega} \leq C) \\
& + P(\#X_{n,\Omega} > C) \\
\leq\ & (1 - \frac{\delta}{2}) \frac{C}{C'} + \frac{\delta}{2} \\
<\ & 1 - \frac{\delta}{2}
\end{aligned}$$

where $\mathrm{Enl}(U, \epsilon)$ is the ϵ-enlargement of U defined as:

$$\mathrm{Enl}(U, \epsilon) = \{\boldsymbol{x}; \exists \boldsymbol{x}' \in U, \|\boldsymbol{x} - \boldsymbol{x}'\| \leq \epsilon\}\,.$$

This contradicts Eq. 5.4.

This concludes the proof of $\alpha\beta \leq 1$.

5.2.3 Conclusion

Our results are based on the *locality assumption* (Eq. 5.4). They show tight results in some cases. The locality assumption is somewhat natural, as most evolution strategies (not all, but almost all) verify this assumption; for example, [82], one of the main evolutionary optimization convergence results, shows a linear convergence, with no sampling far away; [83], showing faster rates with surrogate models, also verifies this assumption; and polynomial rates in noisy optimization as in [68, 84] have the same property as well as many experimentally known rates [85]. Nonetheless, other assumptions leading to similar results (e.g. assumptions ensuring that points far from the optimum cannot help too much) are worth being investigated for clarifying the overall picture.

Basically, our results are about optimization tricks as follows: an optimization algorithm uses the *far sampling trick* if there is a clear distinction between the approximation of the optimum that they propose and the search points that they use for exploring the fitness function.

Our results can be seen either:

- As proofs that fast rates (faster than those in the tables below) are possible only if you assume that points far away from the optimum do bring information, by statistical model estimation, which are relevant for improving the rate (so that the "far sampling trick" can work).

- As proofs that algorithms which, like most evolutionary algorithms (but not all), do not sample far away from the optimum, can not be optimal when the function is "flat" enough around the optimum (there are algorithms and families of functions for which better rates are possible - which does not mean that algorithms which do not want to use the far sampling trick are necessarily bad algorithms).

The hot discussions on the BBOB mailing list, around the fact that the testbeds should distinguish the search points used for approximating the optimum and the search points used for gathering information by sampling far away, suggest that our work comes at the right moment for noisy optimization formal analysis.

Table 5.1 summarizes the state of the art; new result from our work are in bold, and we emphasize cases in which a gap is known. We see that our results show the tightness in the case $\gamma = 1$ (small noise; the variance goes to zero around the optimum), and reduce the gap in the case $\gamma < 1$ (large noise). Our results also show that for fast rates, sampling far from the optimum is necessary; e.g. if $\beta = 4$, $\alpha = 1/2$ is possible only with sampling far from the optimum. Though, this is only possible if there are long-range dependencies on the fitness function, so that such points far from the optimum can be used.

Further work

The locality assumption might be realistic or not, depending on the application. From many discussions around that, we believe that there is room for works like this one, in which a locality assumption prevents the use of information far from the optimum, reliable only when strong assumptions on the model are available. It is also a model which shows that some rates imply sampling far from the optimum. Fast optimization algorithms might, as CLOP, be a compromise between sampling close to the optimum and sampling on areas of maximum uncertainty. Further investigations on intermediate models might be a good idea.

There is still a gap between the upper and the lower bound, for algorithms having the locality assumption, in the case $\gamma < 1$ (large noise), which is an immediate further work.

We consider noisy optimization in the case of local convergence; clearly, the global convergence case can also be interesting [88].

We did not compute exactly constants C and C'. Maybe it is possible to obtain more information on the constant in the convergence using detailed computations of C and C'.

"Flatness" β	Proved rate for R-EDA in [70] ("flatness" on an envelope of the fitness function; the fitness function does not have to be flat around $\boldsymbol{x}^*$)	Lower bound in [70]	R-EDA experimental rate in [68] (on functions with invariances)	Our work (lower bound under locality assumption)	Rate for learnable cases
Framework $\gamma = 1$ (small noise)					
1	$\alpha \geq 1$	$\alpha \leq 1$	$\alpha = 1$	$\alpha \leq 1$	$\alpha = 1/2$
2	$\alpha \geq 1/2$	$\alpha \leq 1$	$\alpha = 1/2$	$\alpha \leq \mathbf{1/2}$	$\alpha = 1/2$
4	$\alpha \geq 1/4$	$\alpha \leq 1$	$\alpha = 1/4$	$\alpha \leq \mathbf{1/4}$	$\alpha = 1/2$
Framework $\gamma < 1$ (large noise)					
1	$\alpha \geq 1/2$	$\alpha \leq 1$	$\alpha = 1/2$	$\alpha \leq 1$	$\alpha = 1/2$
2	$\alpha \geq 1/4$	$\alpha \leq 1$	$\alpha = 1/4$	$\alpha \leq \mathbf{1/2}$	$\alpha = 1/2$
4	$\alpha \geq 1/8$	$\alpha \leq 1$	$\alpha = 1/8$	$\alpha \leq \mathbf{1/4}$	$\alpha = 1/2$

Table 5.1: This table summarizes the state of the art in terms of α for which $||\tilde{\boldsymbol{x}}_n - \boldsymbol{x}^*|| = \tilde{O}(1/n^\alpha)$ is possible. Rates for E-EDA include functions which are not differentiable, and are just upper bounded by flat functions around $\boldsymbol{x}^*$ with β coefficient; see [68] for more details. Experimental rates for R-EDA are for functions with strong invariances/symmetries; see [68] for details. The last column presents results with $\alpha = 1/2$, corresponding to cases in which using statistical model estimation is possible: the limit case of infinite differentiability in [86, 87, 51], is also reached by quadratic logistic regression under parametric assumptions on the objective function [68]; assumptions are not directly comparable to those of the other columns. Lower bounds on the complexity (upper bounds on α) from our work are under the additional assumption of local sampling.

5.3 Variance reduction

In this section, we consider noisy optimization and some traditional variance reduction techniques aimed at improving the convergence rate, namely (i) common random numbers, which is relevant for population-based noisy optimization, and (ii) stratified sampling, which is relevant for most noisy optimization problems. We present artificial models of noise for which common random numbers are very efficient, and artificial models of noise for which common random numbers are detrimental. We then experiment on a desperately expensive unit commitment problem. Stratified sampling is never detrimental. In practice, common random numbers nonetheless provided most of the improvement.

5.3.1 Introduction

We consider a function $f(\boldsymbol{x}, \omega)$, with $\boldsymbol{x}$ in a d-dimensional domain and ω a random variable with values in $D \subset \mathbb{R}$. We assume that the optimization algorithm has only access to random realizations of $f(\boldsymbol{x}, \omega)$. The goal of the optimization algorithm is to approximate $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x} \in \mathcal{X}} \mathbb{E}_\omega f(\boldsymbol{x}, \omega)$.

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 $\boldsymbol{x}$ and receive a realization of $f(\boldsymbol{x}, \omega_n)$. The ω_n, $n \in \{1, 2, \ldots\}$, are independent copies of ω. The algorithm can not influence ω_n. Contrarily to this standard setting, we here assume that the algorithm can request $f(\boldsymbol{x}, \omega_n)$ where ω_n is: (i) either an independent copy of ω (independent of all previously used values); (ii) or a previously used realization ω_m for some $m < n$ (m is chosen by the optimization algorithm). Due to this possibility, *common random numbers* can be applied, i.e. the same ω_n can be used several times, as explained later. We also assume that we have *strata*. A stratum is a subset of D. Strata have empty intersections and their union is D (they realize a partition of D). When an independent copy of ω is requested, the algorithm can decide to provide it conditionally to a chosen stratum. Thanks to strata, we can apply *stratified sampling*.

Statistics of variance reduction

Monte Carlo methods aim at the estimation of the expected value of a random variable owing to a randomly drawn sample. Typically, in our context, $f(\boldsymbol{x}, \omega)$ can be

estimated as a result of $f(\boldsymbol{x}, \omega_1), f(\boldsymbol{x}, \omega_2), \ldots, f(\boldsymbol{x}, \omega_n)$, where the ω_i are independent copies of ω, $i \in \{1, \ldots, n\}$. Laws of large numbers prove, under various assumptions, the convergence of Monte Carlo estimates such as

$$\hat{\mathbb{E}} f(\boldsymbol{x}, \omega) = \frac{1}{n} \sum_{i=1}^{n} f(\boldsymbol{x}, \omega_i) \to \mathbb{E}_\omega f(\boldsymbol{x}, \omega). \tag{5.10}$$

There are classical techniques for improving the convergence:

Antithetic variates (symmetries): ensure some regularity of the sampling by using symmetries. E.g. if the random variable ω is invariant by symmetry w.r.t 0, then, instead of Eq. 5.10, we use Eq. 5.11, which reduces the variance. More sophisticated antithetic variables are possible (combining several symmetries).

$$\hat{\mathbb{E}} f(\boldsymbol{x}, \omega) = \frac{1}{n} \sum_{i=1}^{n/2} \left(f(\boldsymbol{x}, \omega_i) + f(\boldsymbol{x}, -\omega_i) \right). \tag{5.11}$$

Importance sampling: instead of sampling ω with density dP, we sample ω' with density dP'. We choose ω' such that the density dP' of ω' is higher in parts of the domain which are critical for the estimation. However, this change of distribution will introduce a bias. Therefore, when computing the average, we change the weights of individuals by the ratio of probability densities as shown in Eq. 5.12.

$$\hat{\mathbb{E}} f(\boldsymbol{x}, \omega) = \frac{1}{n} \sum_{i=1}^{n} \frac{dP(\omega_i)}{dP'(\omega_i)} \left(f(\boldsymbol{x}, \omega_i) \right). \tag{5.12}$$

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 in bad cases and much better in good situations [89, 90, 91]. There are interesting (but difficult and rather "white box") tricks for making them applicable for time-dependent random processes with many time steps [92].

Finite sample which approximate a random process: [93] proposes to generate a finite sample which approximate a random process for some metric. This method has advantages when applied in the framework of Bellman algorithms, but it is hardly applicable when aiming at the convergence to the solution for the underlying random process.

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

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

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 ω_i is randomly drawn conditionally to a stratum (Eq. 5.13). We consider that the domain of ω 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. ω_i is randomly drawn conditionally to $\omega_i\in S_{s(i)}$.

$$\hat{\mathbb{E}}f(\boldsymbol{x},\omega)=\sum_{i\in\{1,\ldots,n\}}\frac{P(\omega\in s(i))f(\boldsymbol{x},\omega_i)}{\text{Card }\{j\in\{1,\ldots,n\};\omega_j\in s(i)\}}. \tag{5.13}$$

Common random numbers (CRN): also named *correlated sampling*, *paired comparison* or *paired sampling*, refers to the case where we want to know $\mathbb{E}f(\boldsymbol{x},\omega)$ for several $\boldsymbol{x}$, and use the same samples $\omega_1,\ldots,\omega_n$ for the different possible values of $\boldsymbol{x}$.

In this work, we focus on *stratified sampling* and *common random numbers*, 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 (i) building strata and (ii) 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, [94, 95] present an iterative algorithm which stratifies a highly skewed population into a take-all stratum and a number of take-some strata. [96] 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 $g(\omega)$ might approximate $\omega \mapsto f(\boldsymbol{x}^*, \omega)$, and then strata can be defined as a partition of the $g(\omega)$. It is also convenient for visualization, as in many cases the user is interested in viewing statistics for ω leading to extreme values of $\mathbb{E}_\omega f(\boldsymbol{x}^*, \omega)$. More generally, two criteria dictate the choice of strata:

- a small variance inside each stratum, i.e. $Var_{\omega|S} f(\boldsymbol{x}^*, \omega)$ 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 $P(\omega \in S)$;
- *optimal*, i.e. the number of samples in stratum S is proportional to a product of $P(\omega \in S)$ and an approximation of the standard deviation $\sqrt{Var_{\omega|S} f(\boldsymbol{x}^*, \omega)}$. In this case, reweighting is necessary, as in Eq. 5.13.

Stratified noisy optimization: Compared to classical stratified Monte Carlo, an additional difficulty when working in stratified noisy optimization is that $\boldsymbol{x}^*$ is unknown, so we can not easily sample $f(\boldsymbol{x}^*, \omega)$. Also, the strata should be used for many different $\boldsymbol{x}$; if some of them are very different, nothing guarantees that the variance $Var_{\omega|S} f(\boldsymbol{x}, \omega)$ is approximately the same for each $\boldsymbol{x}$ and for $\boldsymbol{x}^*$. 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. [97] uses the word "stratified" for defining their sampling applied to noisy optimization; however it is *Latin Hypercube Sampling* (LHS) rather than stratified sampling.

Common random numbers

Common Random Numbers (CRN), is a simple but powerful technique for variance reduction in noisy optimization problems. Consider $\boldsymbol{x}_1, \boldsymbol{x}_2 \in R^d$, where d is the

dimension of search domain and ω_i denotes the i^{th} independent copy of ω:

$$\begin{aligned} & Var \sum_{i\in\{1,\dots,n\}} (f(\boldsymbol{x}_1,\omega_i) - f(\boldsymbol{x}_2,{\omega'}_i)) = nVar(f(\boldsymbol{x}_1,\omega_i) - f(\boldsymbol{x}_2,{\omega'}_i)) \\ = \;& nVar\ f(\boldsymbol{x}_1,\omega_i) + nVar\ f(\boldsymbol{x}_2,{\omega'}_i) - \ 2n \times Covariance(f(\boldsymbol{x}_1,\omega_i), f(\boldsymbol{x}_2,{\omega'}_i)) \\ = \;& Var \sum_{i\in\{1,\dots,n\}} f(\boldsymbol{x}_1,\omega_i) + Var \sum_{i\in\{1,\dots,n\}} f(\boldsymbol{x}_2,{\omega'}_i) \\ & - 2n \times Covariance(f(\boldsymbol{x}_1,\omega_i), f(\boldsymbol{x}_2,{\omega'}_i)). \end{aligned}$$

If $Covariance(f(\boldsymbol{x}_1,\omega_i), f(\boldsymbol{x}_2,{\omega'}_i)) > 0$, i.e. there is a positive correlation between $f(\boldsymbol{x}_1,\omega_i)$ and $f(\boldsymbol{x}_2,{\omega'}_i)$, the estimation errors are smaller. CRN is based on $\omega_i = {\omega'}_i$, which is usually a simple and efficient solution for correlating $f(\boldsymbol{x}_1,\omega_i)$ and $f(\boldsymbol{x}_2,{\omega'}_i)$; there are examples in which, however, this correlation does not exist. In Section 5.3.3, we will present examples in which CRN does not work. 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 [98]. The cost of labelling is a classical reason for this. This is different from simulating against paired random realizations, 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* [99, 100] and *Differential Evolution* [101]. [102] proposes a paired comparison-based *Interactive Differential Evolution* method with faster rates. In *Direct Policy Search*, paired noisy optimization has been proposed in [103, 104, 105]. Our work follows such approaches and combines them with stratified sampling. This is developed in the next section.

In *Stochastic Dynamic Programming* (section 3.2) and its dual counterpart Dual SDP (section 3.3), 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 runs. 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 [106] or one (new, independent) random realization per iteration [107]. 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 this work; 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.

Paired noisy optimization for dynamic problems. Paired statistical tests (e.g. Pegasus [108]) 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" [103, 104]. *Wilcoxon signed rank sum test* pays more attention to small improvements across all scenarios rather than large changes over the return of an individual one, so that it can reduce the "overfitting" caused by a few extreme (good or bad) scenarios. [104] also shows that using an adaptive number of trials for each policy can speed-up learning in such a CRN framework. The most related existing work is [105]. 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.

Different forms of pairing. For each request $\boldsymbol{x}_n$ to the objective function oracle, the algorithm also provides a set $Seed_n$ of random seeds; $Seed_n = \{seed_{n,1}, \ldots, seed_{n,m_n}\}$. 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). However, to the best of our knowledge, there is no published work about using a common pool of random seeds for all search points, and then randomly drawing a subset of this pool for each search point. In Section 5.3.3, 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 below an algorithm for switching smoothly from full pairing to no pairing at all.

5.3.2 Algorithms

We have seen that pairing can be efficient or detrimental depending on the problem. We will here propose intermediate algorithms, somewhere in between the paired case ($g(n) = r(n)$ below) and the totally unpaired case ($g(n)$ large, see below as well). Let us consider algorithms with:

- λ individuals to be evaluated at each iteration;
- each of these λ individuals is evaluated $r(n)$ times (n is the iteration index) with $r(n)$ distinct random seeds randomly drawn in the family $P_{g(n)}$.

$P_{g(n)}$ is a set of cardinal $g(n)$ (with $g(n) \geq r(n)$), which is either:

- nested case: the set $\{(\omega'_1, \omega''_1), (\omega'_2, \omega''_2), \ldots, (\omega'_{g(n)}, \omega''_{g(n)})\}$, with (ω'_i, ω''_i) independently distributed, identically to $\omega = (\omega', \omega'')$;
- or independent case: an independently drawn random set of cardinal $g(n)$ according to the distribution of $\omega = (\omega', \omega'')$.

5.3.3 Experiments

Artificial experiments

We consider a $(\mu/\mu, \lambda)$-Self-Adaptive Evolution Strategy (see section 4.1.2), with $\lambda = 8d^2$, $\mu = min(2d, round(\lambda/4))$. All experiments are performed with 10000 function evaluations and are reproduced 9999 times.

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 would not change any comparison and has no impact on the algorithm run. Setting $\alpha = 1$ in Eq. 5.14 (below) provides an example in which pairing totally cancels the noise. 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;
- we assume that 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 has a dominant orientation;
- each policy is evaluated on 50 different simulated wind events.

With CRN: If the wind orientation was on average more East than it would be on expectation, then this "East orientation bias" is the same for all evaluated policies. As a consequence, the good individuals are more East-oriented. The next iterate is therefore biased toward East-oriented.

Without CRN: (i) 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. (ii) Therefore, some individuals will be selected with a East orientation bias, but others with a West orientation bias or other biases. (iii) As a conclusion, the next iterate will incur an average of many uncorrelated random biases, which is therefore less biased.

We will design an artificial testbed which smoothly (parametrically depending on α) switches from ideal case for pairing (pairing cancels the noise) to a worst case setting for pairing (as illustrated above). We will compare stratified sampling and common random numbers on this artificial testbed. Later, we will consider a realistic application.

Artificial testbed for paired noisy optimization. With $\omega = (\omega', \omega'')$, let us define

$$f(\boldsymbol{x}, \omega) = ||\boldsymbol{x}||^2 + \alpha\omega' + 20(1-\alpha)\omega'' \cdot \boldsymbol{x}. \tag{5.14}$$

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

- **Continuous case:** ω' is a unidimensional standard Gaussian random variable and ω'' is a d-dimensional standard Gaussian random variable.
- **Discrete case:** ω' is a Bernoulli random variable with parameter $\frac{1}{2}$ and ω'' 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 ω'', which leads 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: a *Sample Average Approximation* is enough for fast convergence, as in the noise-free case, $r(n) = g(n)$ leads to cancelling noise, even with $r(n) = 1$.

- The case $\alpha = 0$ is very hard for pairing; the case $r(n) = g(n)$ means that the noise has the same bias for all points.

- For the noisy framework, the stratified sampling directly reduces the dimension of the noisy case: the two first components have no more noise in the stratified case.

Experimental results. We study $\mathbb{E}\log(||\boldsymbol{x}||^2)/log(n_e)$ (the lower the better), where $\boldsymbol{x}$ 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 5.2. Standard deviations are ± 0.0015 for the worst cases and are not presented. Essentially, the results are: (i) When α is close to 1, β small (more pairing) is better. (ii) When α is close to 0, β large (nearly no pairing) is better. In the discrete case, it is easy to define pairing as we can use strata that correspond to distinct values of the two first components of ω''. Using the four strata corresponding to the 2 possible values of each of the two first components of ω'', we get the results presented in Table 5.3. 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.

Real world experiments

For the real world experiments, the same framework as presented in chapter 4 is considered. This framework is described in section 4.2.1. We use the same settings here, except that we use an *operational horizon* $h = 5$ time steps (i.e. each time we make a decision, it covers 5 time steps) and a *tactical horizon* $k = 10$ 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). We use Direct Value Search (chapter 4) as the resolution method in these experiments. Like in chapter 4, a (μ, λ)-evolution strategy is used to optimize the parametric policy which approximates of the valorization function V. Moreover, like in chapter 4, this parametric policy is a neural network with one hidden layer. The (μ, λ)-evolution strategy is described in section 4.1.2. The following setup is used: $d = 60$ (number of weights in the neural network);

$\lambda = 4(d+1) = 244$; $\mu = \lambda/4$; $r(n) = \lceil 10\sqrt{n+1} \rceil$. We define paired optimization (common random numbers) and stratified sampling in such a case:

- We apply the evolutionary algorithm for optimizing the parameters (i.e. the weights) of the neural network controller.
- Each (noisy) evaluation within the evolutionary algorithms is a Monte Carlo average reward for a vector of parameters; a Monte Carlo evaluation is a call to $f(x, \omega)$. Here, f is the "Simulate" function defined in Algorithm 9. It repeatedly applies the parametric policy and the transition function from an initial state to a final state. The returned value is the cumulated reward (or cost).
- 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 $r(n) = g(n)$, i.e. the same random outcomes $\omega_1, \ldots, \omega_{r(n)}$ are used for all individuals of a generation (of the evolutionary algorithm). The random outcomes $\omega_1, \ldots, \omega_{r(n)}$ are independently drawn for each new generation.

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 (i) the 25% best cases (ii) the 25% worst cases (iii) the second quartile (iv) 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. $f(x, \omega)$. The four strata are the four quantiles on the annual average of an important scalar component of the noise.

Experimental results in Fig. 5.2 show that pairing provides huge improvement in the realistic case. Stratification has a minor impact.

5.3.4 Conclusions

We tested, in Direct Policy Search, paired optimization 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

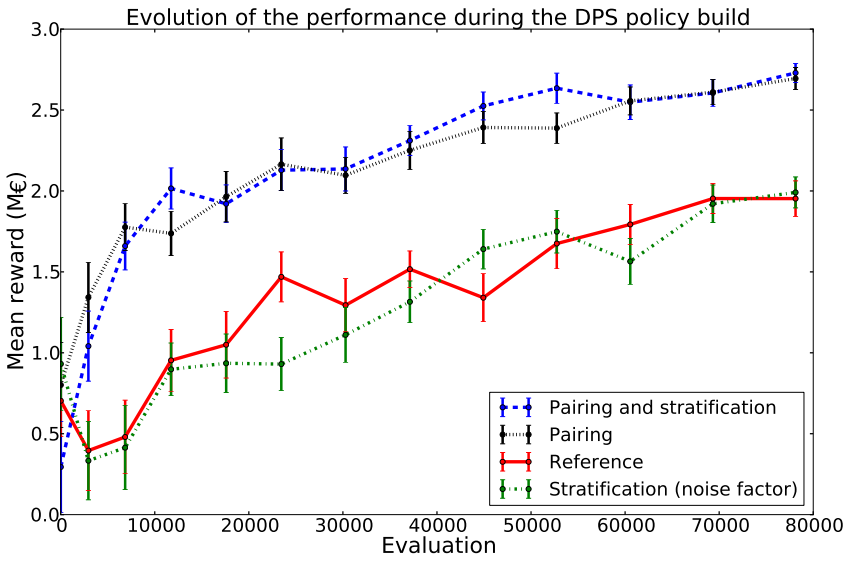

Figure 5.2: X-axis: evaluation index. Y-axis: reward (the higher the better). We see that pairing is very efficient whereas stratification provides no clear improvement.

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 (Eq. 5.14). We provided illustrative examples of such a detrimental effect (Section 5.3.3). 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 by our application (which does not have natural strata).

Further work. Other variance reduction techniques are possible. Methodologies protecting variance reduction techniques from their possible detrimental effects are a nice challenge (e.g. as efficient as CRN when $\alpha = 1$ and as efficient as no pairing when $\alpha = 0$).

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)							
$\alpha = 0$	-0.07435	-0.06654	-0.07670	-0.08581	-0.09219	**-0.09603**	-0.09344
$\alpha = 0.8$	-0.34475	-0.34661	-0.35921	-0.36253	-0.36565	-0.36709	**-0.36917**
$\alpha = 1$	**-0.75048**	-0.52772	-0.50544	-0.49794	-0.49109	-0.49339	-0.49182
dimension 3 (bold for best tested algorithm)							
$\alpha = 0$	-0.06258	-0.06373	-0.07978	-0.09489	-0.10463	-0.10931	**-0.10977**
$\alpha = 1$	**-0.47681**	-0.43320	-0.41439	-0.41004	-0.40880	-0.40202	-0.39641
dimension 5 (bold for best tested algorithm)							
$\alpha = 0$	0.02965	0.03964	0.04409	0.04394	0.04680	0.04826	0.04823
$\alpha = 0.8$	-0.15077	-0.15977	-0.16369	-0.16687	-0.16770	-0.16793	**-0.16920**
$\alpha = 1$	**-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)
$\alpha = 0$	0.097						**-0.033**
$\alpha = 0.8$	0.038						**-0.053**
$\alpha = 1.0$	**-0.057**						-0.054

Table 5.2: 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. Intermediate values of β (intermediate levels of pairing) were never significantly better than others and not clearly more robust to changes in α.

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

Table 5.3: Table of results (the lower, the better; see text for details) 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.

Chapter 6

Convergence in non-quasi-convex problems

In this chapter, we study the linear convergence of a simple pattern search method on non quasi-convex functions defined on continuous domains. The assumptions retained include an assumption on the sampling performed by the evolutionary algorithm (supposed to cover efficiently the neighborhood of the current search point), the conditioning of the objective function (so that the probability of improvement is not too low at each time step, given a correct step size), and the unicity of the optimum.

6.1 Introduction

Continuous evolutionary algorithms are well known for robust convergence. However, most proven results are for simple objective functions, e.g. sphere functions [82]. Results also include compositions with monotone functions (so that not only convex functions are covered), but the considered objective functions are nonetheless still almost always quasi-convex (i.e. sublevel sets are convex), as well as most derivative free optimization algorithms [55], whereas nearly all testbeds are based on more difficult functions [109, 110]. Extensions to non quasi-convex functions are still rare [111] and limited to convergence (i.e.: asymptotically we will find the optimum). We here extend such results to linear convergence (i.e. the precision after n iterations is $O(\exp(-\Omega(n)))$. There are works devoted to unimodal objective functions, without convexity assumptions [112], but such works are in the discrete domain and do not say anything for the linear convergence on continuous domains. All in all, only one of the six objective functions of Fig. 6.1 is covered by existing results, in terms of

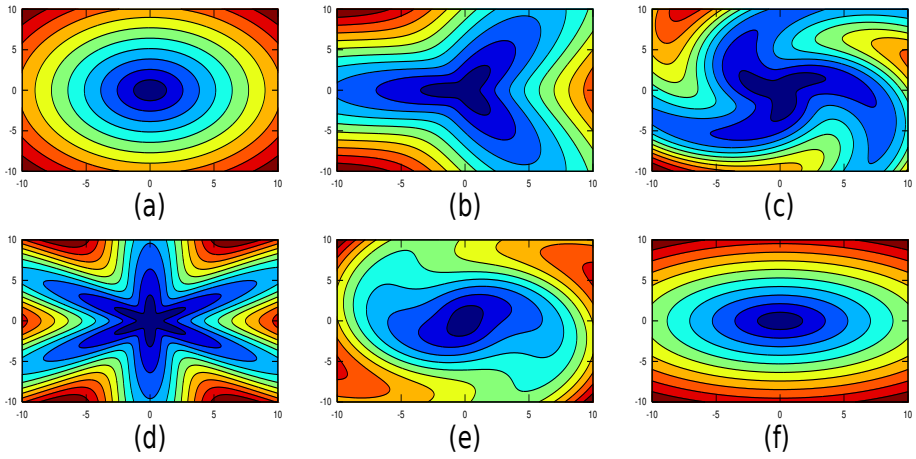

Figure 6.1: Six graphical representations of "easy" objective functions; only the first one (sphere) is covered by existing linear convergence results. Even the sixth one (ellipsoids) is not included in published linear convergence results. We extend to all functions verifying Eqs. 6.1-6.6 (see Section 6.2), including all functions presented here. We present assumptions under which our results hold in Section 6.2, the main result in Section 6.3, and we will show in details that the sixth case above (quadratic functions) is covered by the result in Section 6.3.3 (but case 1 is a special case of case 6, and cases 2, 3, 4, 5 can be tackled similarly). [111] provides other examples, with different but very related assumptions; their examples are also covered by our theorem.

linear convergence.

In this chapter, we prove linear convergence of a simple pattern search method with derandomized sampling on non quasi-convex families of functions. Section 6.2 presents the framework, and the assumptions under which our results hold. Section 6.3 is the mathematical analysis, under this set of assumptions. Section 6.3.3 presents the application to positive definite quadratic forms: it shows that the family of quadratic forms with conditioning bounded by some constant verifies our set of assumptions, and therefore that our evolution strategy with derandomized sampling has linear convergence rate on such objective functions. Incidentally, this section emphasizes the critical underlying assumptions for proving the result, suggesting extensions to other families of fitness functions. Section 6.4 concludes and discusses limitations and further work.

6.2 A Simple Pattern Search Method

We consider an evolutionary algorithm as in Alg. 13. As the sampling is derandomized, we might indeed call this algorithm a pattern search method. We assume the following framework.

6.2.1 The Objective Function

We assume that the function f has a unique minimum. Without loss of generality, we assume that the objective function verifies $f(0) = 0$ and that this is the minimum. The considered algorithms are invariant by translation or composition with monotone functions, so this does not reduce the generality of the analysis.

6.2.2 Conditioning

We assume that

$$K'||\boldsymbol{x}|| \leq f(\boldsymbol{x}) \leq K''||\boldsymbol{x}|| \tag{6.1}$$

for all $\boldsymbol{x}$ in $\mathbb{R}^d$ and for some constants $K' > 0$ and $K'' > 0$. We point out that, as we consider algorithms which are invariant under transformations of the objective function by composition with monotonic functions, this assumption is not so strong as a constraint and quadratic positive definite forms with bounded condition number are in fact covered (their square root verifies Eq. 6.1).

6.2.3 Good Sampling

Here we use the derandomized sampling assumptions (Eqs. 6.2-6.6), which are crucial in our work. This sampling is deterministic, as in pattern search methods [55]. We

Algorithm 13 The Simple Evolution Strategy. In case there is no unicity for choosing $\boldsymbol{x}'$, any tie breaking solution is good. (c) refers to the counting operation, which will be important in the proof. $[[1, k]]$ stands for the integer set $\{1, \ldots, k\}$.

Initialize $\boldsymbol{x} \in \mathbb{R}^d$
Parameters $k \in \mathbb{N}^*, \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_k \in \mathbb{R}^d, \sigma \in \mathbb{R}_+^*, k_1 \in \mathbb{N}^*, k_2 \in \mathbb{N}^*$
for $t = 1, 2, 3, \ldots$ **do**

  // **just for archiving**
  $\boldsymbol{X}_t \leftarrow \boldsymbol{x}$

  // **mutations**
  For $i \in [[1, k]]$, $\boldsymbol{x}_i \leftarrow \boldsymbol{x} + \sigma \boldsymbol{\delta}_i$

  // **useful auxiliary variables**
  $n \leftarrow$ number of $\boldsymbol{x}_i$ such that $f(\boldsymbol{x}_i) < f(\boldsymbol{x})$ (c)
  $\boldsymbol{x}' \leftarrow \boldsymbol{x}_i$ with $i \in [[1, k]]$ such that $f(\boldsymbol{x}_i)$ is minimum

  // **step-size adaptation**
  if $n \leq k_1$ **then**
    $\sigma \leftarrow \sigma / 2$
  end if
  if $n \geq k_2$ **then**
    $\sigma \leftarrow 2\sigma$
  end if

  // **win: accepted mutation**
  if $k_1 < n < k_2$ **then**
    $\boldsymbol{x} \leftarrow \boldsymbol{x}'$
  end if
end for

assume that for some $0 < b < b' \leq 2b' \leq c' \leq c, 0 < \eta < 1$ and $\forall \boldsymbol{x} \in \mathbb{R}^d$,

$$\begin{aligned} \sigma \text{ too large:} \quad & \sigma \geq b^{-1}||\boldsymbol{x}|| \\ & \Rightarrow \#\{i \in [[1,k]]; f(\boldsymbol{x}+\sigma\boldsymbol{\delta}_i) < f(\boldsymbol{x})\} \leq k_1 \end{aligned} \tag{6.2}$$

$$\begin{aligned} \sigma \text{ small enough:} \quad & \sigma \leq b'^{-1}||\boldsymbol{x}|| \\ & \Rightarrow \#\{i \in [[1,k]]; f(\boldsymbol{x}+\sigma\boldsymbol{\delta}_i) < f(\boldsymbol{x})\} > k_1 \end{aligned} \tag{6.3}$$

$$\begin{aligned} \sigma \text{ large enough:} \quad & \sigma \geq c'^{-1}||\boldsymbol{x}|| \\ & \Rightarrow \#\{i \in [[1,k]]; f(\boldsymbol{x}+\sigma\boldsymbol{\delta}_i) < f(\boldsymbol{x})\} < k_2 \end{aligned} \tag{6.4}$$

$$\begin{aligned} \sigma \text{ too small:} \quad & \sigma \leq c^{-1}||\boldsymbol{x}|| \\ & \Rightarrow \#\{i \in [[1,k]]; f(\boldsymbol{x}+\sigma\boldsymbol{\delta}_i) < f(\boldsymbol{x})\} \geq k_2 \end{aligned} \tag{6.5}$$

$$\begin{aligned} \text{Perfect } \sigma: \quad & b'^{-1}||\boldsymbol{x}|| \leq \sigma \leq c'^{-1}||\boldsymbol{x}|| \\ & \Rightarrow \exists i \in [[1,k]]; f(\boldsymbol{x}+\sigma\boldsymbol{\delta}_i) \leq \eta f(\boldsymbol{x}) \end{aligned} \tag{6.6}$$

6.2.4 Discussion on Assumptions

Assumptions 6.2, 6.3, 6.4, 6.5, 6.6 basically assume that the sampling is regular enough for the shape of the level sets. For example, the finite VC-dimension of ellipsoids ensure that, when the conditioning is bounded, quadratic functions verify the assumptions above (and therefore the theorem below) with arbitrarily high probability if $\boldsymbol{\delta}_1,\ldots,\boldsymbol{\delta}_k$ are randomly drawn and if k is large enough. Importantly, the critical assumption in the derandomization is that all iterations have the same $\boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_k$. This will be developed in Section 6.3.3.

Assumptions 6.6 and 6.1 use the fitness values; but they just have to hold for one of the fitness values obtained by replacing f with $g \circ f$ with g a monotone function.

6.3 Mathematical Analysis

6.3.1 Main Theorem

Assume Eqs. 6.1-6.6. There exists a constant K, depending on $\eta, K', K'', \max_i ||\boldsymbol{\delta}_i||$ only such that for index t sufficiently large

$$\ln(||\boldsymbol{X}_t||)/t \leq K < 0 \tag{6.7}$$

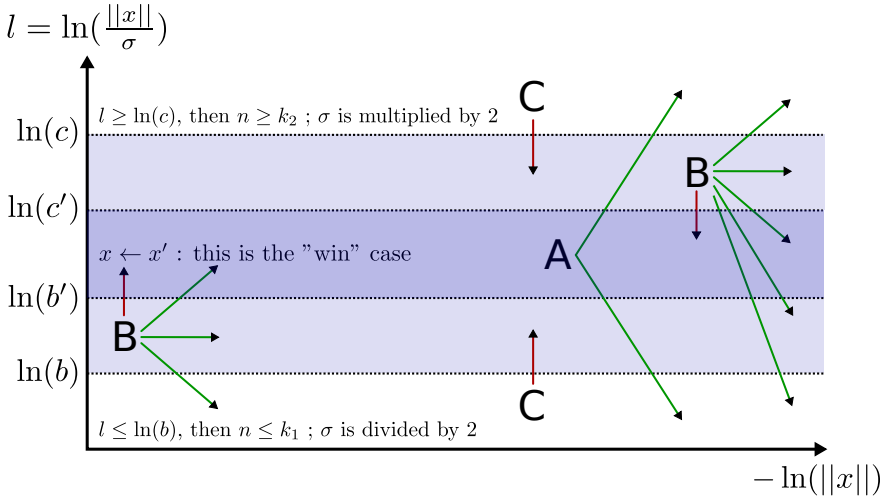

Figure 6.2: The linear convergence proof in a nutshell. X-axis: $-\ln(||\boldsymbol{x}||)$. Y-axis: $l = \ln(\frac{||\boldsymbol{x}||}{\sigma})$. At each iteration, either **case A holds:** then the iteration is for sure an improvement by a factor at least η, or **case B holds:** the iteration can be an improvement or not; if not, the point is moved towards case A by $\ln(2)$ upwards or downwards, or **case C holds:** then $\boldsymbol{x}$ is moved upwards (if it is at the bottom) or downwards (if it is at the top). This ensures that after finitely many time steps we go back to case A unless there is a "win" by case B in the mean time. The crucial point for the proof is that each "win" is an improvement by at least a controlled factor η, so that the slope of "win" arrows is bounded, so that there is linear convergence and not only an infinite sequence of "small" improvements.

(with $\ln(0) = -\infty$*) where the sequence of* $\boldsymbol{X}_t$ *is defined as in Alg. 13.*

6.3.2 Proof

First, we briefly explain and illustrate the proof, before the formal proof below. The proof is sketched in Fig. 6.2. At each iteration t, we are at some point in the figure; the x-axis is $-\ln(||\boldsymbol{x}||)$ (equivalent to $-\ln(f(\boldsymbol{x}))$, by Eq. 6.1), the y-axis is $l = \ln\left(\frac{||\boldsymbol{x}||}{\sigma}\right)$. The step-size adaptation ensures that if we are at the bottom ($l \leq \ln(b)$), we go upwards; if we are at the top ($l \geq \ln(c)$), we go downwards. Between $l = \ln(b)$ and $l = \ln(c)$, everything can happen; but if there is no "win" case in the mean time, we will arrive between $l = \ln(b')$ and $l = \ln(c')$, where a win is ensured. As steps are fast, this can not take too much time (if there is no "win", l increases by $\ln(2)$ or decreases by $\ln(2)$ in direction of the "forced win" range $[\ln(b'), \ln(c')]$). This will be

formalized below. $c' \geq 2b'$ ensures that the algorithm can not jump from $l < \ln(b')$ to $l > \ln(c')$ or from $l > \ln(c')$ to $l < \ln(b')$. Therefore there is necessarily a "win" in the mean time. Eq. 6.6 ensures that wins provide a significant improvement.

We now write the proof formally. Consider an iteration of the algorithm, with n the number of mutations i with $f(\boldsymbol{x} + \sigma\boldsymbol{\delta}_i) < f(\boldsymbol{x})$ (as defined in Alg. 13, Eq. (c)).

Define $l = \ln\left(\frac{||\boldsymbol{x}||}{\sigma}\right)$. Eqs. 6.2-6.6 can be rephrased as follows:

$$l \leq \ln(b) \Rightarrow \#\{i \in [[1,k]]; f(\boldsymbol{x} + \sigma\boldsymbol{\delta}_i) < f(\boldsymbol{x})\} \leq k_1 \quad (6.8)$$

$$l \geq \ln(b') \Rightarrow \#\{i \in [[1,k]]; f(\boldsymbol{x} + \sigma\boldsymbol{\delta}_i) < f(\boldsymbol{x})\} > k_1 \quad (6.9)$$

$$l \leq \ln(c') \Rightarrow \#\{i \in [[1,k]]; f(\boldsymbol{x} + \sigma\boldsymbol{\delta}_i) < f(\boldsymbol{x})\} < k_2 \quad (6.10)$$

$$l \geq \ln(c) \Rightarrow \#\{i \in [[1,k]]; f(\boldsymbol{x} + \sigma\boldsymbol{\delta}_i) < f(\boldsymbol{x})\} \geq k_2 \quad (6.11)$$

$$\ln(b') \leq l \leq \ln(c') \Rightarrow \exists i \in [[1,k]]; f(\boldsymbol{x} + \sigma\boldsymbol{\delta}_i) \leq \eta f(\boldsymbol{x}) \quad (6.12)$$

Define $\boldsymbol{x}'$ as in Alg. 13. We get the following behavior:

- Forced increase: if $l \leq \ln(b)$, then $n \leq k_1$; σ is divided by 2, and l is increased by $\ln(2)$ (Eq. 6.8). This is a case C in Fig. 6.2.
- Forced decrease: if $l \geq \ln(c)$, then $n \geq k_2$; σ is multiplied by 2, and l is decreased by $\ln(2)$ (Eq. 6.11). This is a case C in Fig. 6.2.
- Forced win: if $\ln(b') \leq l \leq \ln(c')$, then $\boldsymbol{x} \leftarrow \boldsymbol{x}'$; this is the "sure win" case (Eq. 6.12); l can be increased (at most by $\max_i ||\boldsymbol{\delta}_i||$) or decreased (by $\Delta = \ln(||\boldsymbol{x}||/||\boldsymbol{x}'||)$). This is a case A in Fig. 6.2.

Importantly, these 3 cases do not cover all possible cases; $\ln(c') < l < \ln(c)$ and $\ln(b) < l < \ln(b')$ are not covered in items above. These two remaining cases are termed case B in Fig. 6.2.

Step 1: Showing that there are infinitely many wins.

The two first lines above (case $l \leq \ln(b)$ and case $l \geq \ln(c)$) ensure that if l is too low or too high, it eventually comes back to the range $[\ln(b'), \ln(c')]$ (where a win necessarily occurs), unless there is a win in the mean time (in the range $[\ln(b), \ln(c)]$ where wins are not sure but are possible). Importantly, l can increase or decrease by $\ln(2)$ at most; so the algorithm can not jump from less than $\ln(b')$ to more than $\ln(c')$. This ensures that infinitely often we have a win $\boldsymbol{x} \leftarrow \boldsymbol{x}'$. But we want linear convergence. Therefore we must consider how many steps there are before we come back to a "win", and how large are improvements in case of "win".

Step 2: showing that "wins" are big enough.

In all cases of "win", i.e. $k_1 < n < k_2$, with $\Delta = \ln(||\boldsymbol{x}||/||\boldsymbol{x}'||)$, we know that $f(\boldsymbol{x}') \leq \eta f(\boldsymbol{x})$ and $f(\boldsymbol{x}') \leq K''||\boldsymbol{x}'|| \leq \frac{K''}{K'}\frac{||\boldsymbol{x}'||}{||\boldsymbol{x}||} f(\boldsymbol{x})$ so that $\ln(f(\boldsymbol{x}))$ is decreased by at least

$$\max(\ln(1/\eta), \ln(K'/K'') + \Delta). \tag{6.13}$$

After a "win", with $l' = \ln\left(\frac{||\boldsymbol{x}'||}{\sigma}\right)$,

- if $l' \leq \ln(b')$, then the number of iterations before the next win is at most $z = 1 + \ln(\frac{c}{b})\Delta/\ln(2)$, because $l' \geq \ln(b) - \Delta \geq \ln(b') - \ln(b'/b) - \Delta \geq \ln(b') - \ln(c/b) - \Delta$ and forced increase are by steps of at least $\ln(2)$.
- if $l' \geq \ln(c')$, then the number of iterations before the next win is at most $z = 1 + \ln(\frac{c}{b}) \max_i \ln(||\boldsymbol{\delta}_i||)/\ln(2)$, because $l' \leq \ln(c) - max_i \ln(\boldsymbol{\delta}_i) \leq \ln(c') - \ln(c'/c) - \max_i \ln(\boldsymbol{\delta}_i) \leq \ln(c') - \ln(c/b) - \max_i \ln(\boldsymbol{\delta}_i)$ and forced decreases are by steps of at least $\ln(2)$.
- less than in both cases above, otherwise.

In both cases, Eq. 6.13 divided by z is lower bounded by some positive constant

$$\begin{aligned} & \mathit{ProgressRate} \\ = \; & \text{Eq. 6.13 divided by } z \\ = \; & \frac{\max(\ln(1/\eta), \ln(K'/K'') + \Delta_i)}{\min(1 + \ln(c/b)\Delta_i/\ln(2), 1 + \ln(c/b)\max_j \ln(||\boldsymbol{\delta}_j||)/\ln(2))}. \end{aligned} \tag{6.14}$$

Step 3: summing iterations.

Eq. 6.14 is the progress rate between two wins, after normalization by the number of steps between these two wins. Hence if $t > n_0$,

$$\begin{aligned} \ln(f(\boldsymbol{X}_t)) \; \leq \; & \ln(f(\boldsymbol{X}_1)) - (t - n_0) \times \\ & \sum_i \frac{\max(\ln(1/\eta), \ln(K'/K'') + \Delta_i)}{\min(1 + \Delta_i/\ln(2), 1 + \max_j \ln(||\boldsymbol{\delta}_j||)/\ln(2))} \end{aligned} \tag{6.15}$$

where the summation is for i index of an iteration t with a "win", and n_0 is the number of initial iterations before a "win" (i.e. n_0 depends on the initial conditions but it is finite).

Eq. 6.15 yields the expected result. □

This result would be void if there was no algorithm and no space of functions for which assumptions 6.1-6.6 hold. Therefore, next Section is devoted to showing that for the important case of families of quadratic functions with bounded conditioning, assumptions 6.1-6.6 hold, and therefore the theorem above holds.

6.3.3 Application to Quadratic Functions

This section shows an example of application of the theorem above. The main strength of our results is that it covers many families of functions; yet, Eqs. 6.1-6.6 are not so readable. We show in this section that a simple family of fitness functions verify all the assumptions.

We consider the application to positive definite quadratic forms with bounded conditioning, i.e. we consider $f \in F$ with F the set of quadratic positive definite objective functions f such that

$$\frac{maxEigenValue(Hessian(f))}{minEigenValue(Hessian(f))} < c_{max} < \infty. \tag{6.16}$$

Notably, thanks to the use of VC-dimension, the approach is indeed quite generic and can be applied to all families of functions obtained by rotation/translation from fitness functions in Fig. 6.1.

Instead of working on Q directly, with $\boldsymbol{x} \mapsto Q(\boldsymbol{x} - \boldsymbol{x}^*)$ a quadratic form with Q positive definite with optimum in 0, we work on $\boldsymbol{x} \mapsto \sqrt{Q(\boldsymbol{x} - \boldsymbol{x}^*)}$, so that Eq. 6.1 is verified; as considered algorithms are invariants by composition with monotone functions, this does not change the result.

We assume that $\boldsymbol{\delta}_1, \boldsymbol{\delta}_2, \ldots, \boldsymbol{\delta}_k$ are independently uniformly randomly drawn in the unit ball $B(0,1)$. From now on, we note $p = p_{\boldsymbol{x},\sigma,f}$ the probability that $\boldsymbol{x} + \sigma\boldsymbol{\delta}_i$ is in $E = f^{-1}([0, f(\boldsymbol{x})[)$, and $\hat{p} = \hat{p}_{\boldsymbol{x},\sigma,f}$ the frequency $\frac{1}{k}\sum_{i=1}^{k} \mathbf{1}_{\boldsymbol{x}+\sigma\boldsymbol{\delta}_i \in E}$. We will often drop the indices for short.

The assumptions in Section 6.2 essentially mean that frequencies are close to expectations for $\boldsymbol{x}+\sigma\boldsymbol{\delta}_i \in f^{-1}([0, f(\boldsymbol{x})[)$ and $\boldsymbol{x}+\sigma\boldsymbol{\delta}_i \in f^{-1}([0, \eta f(\boldsymbol{x})[)$, independently of $\boldsymbol{x}$, σ, f. This is typically a case in which VC-dimension [113] can help.

The purpose of this section is to show Eqs. 6.1-6.6, for a given family of functions, namely the family F defined above; by proving Eqs. 6.1-6.6, we show the following.

Corollary:

Assume that the $\boldsymbol{\delta}_i$ are uniformly randomly drawn in the unit ball $B(0,1)$. Assume that F is the set of quadratic functions with minimum in 0 ($f(0) = 0$) which verify

Eq. 6.16 for some $c_{max} < \infty$. Then, almost surely on the sequence $\boldsymbol{\delta}_1, \boldsymbol{\delta}_2, \ldots, \boldsymbol{\delta}_k$, for k large enough and some parameters k_1 and k_2 of Alg. 13, then Eqs. 6.1-6.6 hold, and therefore for some $K < 0$, for all $t > 0$,

$$\ln(||\boldsymbol{X}_t||)/t \leq K \tag{6.17}$$

with $\ln(0) = -\infty$ and where the sequence of $\boldsymbol{X}_t$ is defined as in Alg. 13.

Proof of the corollary

We use the main theorem above for proving Eq. 6.17, so we just have to prove that Eqs. 6.1-6.6 hold.

Step 1: using VC-dimension for approximating expectations by frequencies.

Thanks to the finiteness of the VC-dimension of quadratic forms (see e.g. [72]), we know that for all $\epsilon > 0$, almost surely in $\boldsymbol{\delta}_1, \boldsymbol{\delta}_2, \ldots, \boldsymbol{\delta}_k$, for all $\delta > 0$ and k sufficiently large, with probability at least $1 - \delta$,

$$\sup_{\boldsymbol{x},f,\sigma>0} |\hat{p}_{\boldsymbol{x},\sigma,f} - p_{\boldsymbol{x},\sigma,f}| \leq \epsilon/2 \tag{6.18}$$

where $\boldsymbol{x}$ ranges over the domain, f ranges over F.

For short, we will often drop the indices, so that Eq. 6.18 becomes Eq. 6.19:

$$\sup_{\boldsymbol{x},f,\sigma>0} |\hat{p} - p| \leq \epsilon/2 \tag{6.19}$$

The important point here is that this result is a uniform result (uniform on $f \in F$); this is not just a simple law of large numbers, it is a uniform law of large numbers, so that it is not a mistake if there is a supremum on $\boldsymbol{x}, \sigma, f$. Almost surely, the supremum is bounded; it is not only bounded almost surely for each $\boldsymbol{x}, \sigma, f$ separately, and this is the key concept in this proof.

Step 2: showing that σ small leads to high acceptance rate and σ high leads to small acceptance rate.

Thanks to the bounded conditioning (Eq. 6.16), there exists $\epsilon > 0$ s.t.

$$s' < \frac{1}{2}s \tag{6.20}$$

$$\text{with } s = \sup\left\{\frac{\sigma}{||\boldsymbol{x}||}; \sigma, \boldsymbol{x}, f \text{ s.t. } p \geq \frac{\epsilon}{2}\right\}$$
$$\text{and } s' = \inf\left\{\frac{\sigma}{||\boldsymbol{x}||}; \sigma, \boldsymbol{x}, f \text{ s.t. } p < \frac{1}{2} - \frac{\epsilon}{2}\right\}$$

because $s' \to 0$ and $s \to \infty$ as $\epsilon \to 0$.

Eq. 6.18 implies

$$\frac{1}{2}\hat{s} \geq \frac{1}{2}s \tag{6.21}$$

and

$$s' \geq \hat{s}' \tag{6.22}$$

$$\text{with } \hat{s} = \sup\left\{\frac{\sigma}{||\boldsymbol{x}||}; \sigma, \boldsymbol{x}, f \text{ s.t. } \hat{p} \geq \epsilon\right\}$$
$$\text{and } \hat{s}' = \inf\left\{\frac{\sigma}{||\boldsymbol{x}||}; \sigma, \boldsymbol{x}, f \text{ s.t. } \hat{p} < \frac{1}{2} - \epsilon\right\}$$

So, Eqs. 6.21, 6.22 and 6.19, with k large enough, imply

$$\frac{1}{2}\hat{s} > \hat{s}' \tag{6.23}$$

Eq. 6.23 provide k_1, k_2, c' and b' as follows for Eqs. 6.4 and 6.3:

$$\frac{1}{b'} = \hat{s}\ , \qquad \frac{1}{c'} = \hat{s}',$$
$$k_1 = \lfloor \epsilon k \rfloor \quad \text{and} \quad k_2 = \left\lceil (\frac{1}{2} - \epsilon)k \right\rceil$$

with $\epsilon < \frac{1}{10}$ (due to step 1). Eqs. above imply $c' \geq 2b'$.

Step 3: showing that k large enough and σ well chosen leads to at least one mutation with significant improvement.

Similarly, k large enough yields

$$b^{-1} = \sup\left\{\frac{\sigma}{||\boldsymbol{x}||}; \sigma, \boldsymbol{x}, f \text{ s.t. } \hat{p} > k_1/k\right\},$$
$$c^{-1} = \inf\left\{\frac{\sigma}{||\boldsymbol{x}||}; \sigma, \boldsymbol{x}, f \text{ s.t. } \hat{p} < k_2/k\right\},$$

which provide Eqs. 6.5 and 6.2 with $b < c$ thanks to $\epsilon < \frac{1}{10}$ (ϵ was chosen with $\epsilon < \frac{1}{10}$ in step 1). Eqs. 6.2-6.5 then imply $b < b'$ and $c' < c$.

We now have to ensure Eq. 6.6. Equations Eq. 6.1-6.5 are proven above for k sufficiently large; from now on, we note $q = q_{\boldsymbol{x},\sigma,f}$ the probability that $\boldsymbol{x} + \sigma\boldsymbol{\delta}_i$ is in $E' = f^{-1}([0, \eta f(\boldsymbol{x})[)$, and $\hat{q} = \hat{q}_{\boldsymbol{x},\sigma,f}$ the frequency $\frac{1}{k}\sum_{i=1}^{k} \mathbf{1}_{\boldsymbol{x}+\sigma\boldsymbol{\delta}_i \in E'}$. For showing Eq. 6.6, let us assume

$$b^{-1} \leq \frac{\sigma}{||\boldsymbol{x}||} \leq c^{-1};$$

this implies $q > \epsilon_0$ for some $\epsilon_0 > 0$; so for k sufficiently large for ensuring $\sup_{\sigma,\boldsymbol{x},f} |q_{\boldsymbol{x},\sigma,f} - \hat{q}_{\boldsymbol{x},\sigma,f}| \leq \epsilon_0/2$, by VC-dimension, we get $q' \geq \epsilon_0/2 > 0$, which implies that at least one $\boldsymbol{\delta}_i$ verifies $\boldsymbol{x} + \boldsymbol{\delta}_i \in E'$. This is exactly Eq. 6.6.

Step 4: concluding.

We have shown Eqs. 6.1-6.6 for square roots of positive definite quadratic normal forms with bounded conditioning. Therefore, the main theorem can be applied and leads to Eq. 6.17. □

6.4 Discussion and Conclusion

This work provides, to the best of our knowledge, the first proof of linear convergence of evolutionary algorithms (here, the Simple Evolution Strategy in Alg. 13) in continuous domains on non quasi-convex functions. Indeed, even the application to quadratic positive definite forms is new. This proof is for derandomized samplings only, which means that the mutations $\boldsymbol{\delta}_i$, before multiplication by the step-size which obviously varies, are constant. A main missing point for an application is the evaluation of the convergence rate as a function of condition numbers (see extensions below) and the extension to randomized algorithms preferred by many practitioners.

In Section 6.4.1 we discuss extensions of this work that we plane to develop in the near future, and in Section 6.4.2 deeper (harder to get rid of) limitations.

6.4.1 Extensions

Two properties are used for applying our main theorem to quadratic functions with a bound on condition numbers:

- VC-dimension of level sets. VC-dimension is a classical easy tool for showing that a family of functions verify a property such as Eq. 6.19 for arbitrarily small $\epsilon > 0$, provided that k is large enough.

- Eq. 6.20, also crucial in the proof, is directly a consequence of bounded conditioning (assumption formalized in Eq. 6.16).

With these two assumptions, we can show Eqs. 6.1-6.6, and then the theorem can be applied. This is enough for objective functions with level sets having simple graphical representations with rotations/translations.

However, we do not need assumptions so strong as finite VC-dimension for showing Eqs. 6.1-6.6. Glivenko-Cantelli results are enough; and for this, finiteness of the bracketing covering numbers, for example, is enough [79]; this is the most natural extension of this work. In particular, there are results showing the finiteness of bracketing covering numbers for families of Hölder spaces of functions; this is a nice path for applying results from our work to wide families of functions.

Assumptions in [114] are slightly different from the assumptions in this chapter; their main assumption are

- the frontier of any level set $f^{-1}(r)$ has a bounded curvature.
- for some $C_{min} \in \mathbb{R}$ and $C_{max} \in \mathbb{R}$, with $\boldsymbol{x}^*$ the (assumed unique) optimum of the objective function f and $f(\boldsymbol{x}^*) = 0$, for any $r \in \mathbb{R}$, we have

$$B(\boldsymbol{x}^*, C_{min} r) \subset f^{-1}(r) \subset B(\boldsymbol{x}^*, C_{max} r).$$

The second assumption is equivalent to our conditioning assumption, but the first one is not directly equivalent to our derandomized sampling assumptions. Refining the assumptions might be possible by combining their assumptions and our assumptions.

Condition numbers are classical for estimating the difficulty of local convergence; a nice condition number for difficult optimization should generalize some classical condition number from the literature, and include non-differentiable functions as well. [111] did a first step in that direction; in particular, isotropic algorithms do not solve functions with infinite condition number (for the definition of [111]), whereas covariance-based algorithms [58, 115] do. Eq. 6.7 we guess that it is possible to derive a new such number with direct links to convergence rates of evolution strategies.

6.4.2 Limitations

In this chapter, we work on an evolutionary algorithm for which mutations $\boldsymbol{\delta}_i$'s are randomly drawn once and for all (the same mutation vectors $\boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_k$ for all iterations of the algorithms). This makes the proof much easier. We believe that the proof can be extended to the case in which the mutations are randomly drawn at each iteration, as in most usual cases; yet, the adaptation is not straightforward;

we must study the frequency (over iterations) at which assumptions 6.2-6.6 hold, and the consequences of bad cases on Eq. 6.15. For this work, we just assume that the $\boldsymbol{\delta}_i$'s are randomly drawn once and for all iterations; equivalently, they could be quasi-randomized.

Cumulative adaptation [116] is not considered in our analysis; this is a considerably harder step for generalizing our results, because then the simple separation between 5 cases (see Fig. 6.2) is the idea that clearly divides the proof between step-size adaptation and progress rate.

This work covers quadratic functions, but the rates are not independent of the conditioning, so complementary results are necessary for algorithms evolving a covariance matrix, such as [58, 117, 115]. Maybe ergodic Markov chains are a better tool for showing such results [82].

We work under assumptions which imply a very large k. More precisely, using VC-dimension or bracketing numbers, it is possible to get explicit bounds on k, but these numbers will be far above the usual values for k. Obtaining results for limited values of k is a classical challenge in machine learning, and for the moment only huge values of k are applicable when using VC-dimension assumptions. Seemingly, weaker assumptions are enough, such as Glivenko-Cantelli properties [79]. For our work, VC-dimension is easier to use and sufficient for our purpose.

Part IV

General Conclusion

Chapter 7

Conclusion

In this thesis, we aimed at creating new decision aids tools for energy management. We focused on Unit Commitment problems which are important problems with huge political, economical, environmental and sociological repercussions. Our initial motivation was to help electricity managers to face the new challenges described in section 1.2, that is to say, find better exploitation and investment strategies, with larger, more detailed and more realistic models, using multiple scenarios. Our contribution to making solvable large scale Unit Commitment problems with realistic models can be summarized as follows.

7.1 A general trend in artificial intelligence: reducing the model error

In our opinion, strong assumptions on the model is one of the biggest weakness of most classical decision aids tools for energy management. For instance, most *value-based* methods assume that the random processes involved can be reduced to a small Markovian Random Process and *Stochastic Dual Dynamic Programming* assumes that the value function is convex. Thus, a simplified model of the system to be solved is often used in order to fit these assumptions. But as explained in section 1.5, a good solution for a simplified model can be a rather bad solution for the actual one. For instance, [29] points out that a computer-aided decision can provide good results on a simplified model (the one used as objective function in the computer-aided decision process), but be detrimental on a more realistic one.

We do not deny that mathematical programming methods are essential to deal with large action spaces. Indeed, relying purely on Reinforcement Learning methods

cannot work when the action space is of a large dimension, e.g. more than 10000 action variables and constraints. Such a setting is usual in Unit Commitment problems and therefore mathematical programming is widely used. But this often implies substantial simplifications within the model. Using a method like Stochastic Dual Dynamic Programming is possibly more adequate. However, this implies simplifying the model too. Thus we aimed at exploring a different approach, inspired by the Reinforcement Learning community, and tried to understand how Direct Policy Search – which has very few modelling constraints – could be applied to power systems. This means that we are mostly interested in two questions:

- Finding parametric policies, so that they can be applied on large scale power systems. For this, we find inspiration in classical Stochastic Dynamic Programming representations, i.e. splitting the objective function into instantaneous cost and long-term reward.
- Studying optimization algorithms compliant with Direct Policy Search. Evolution strategies are a classical choice for Direct Policy Search, thus we studied:
  - their runtime in noisy settings;
  - their improvement by pairing methods;
  - their convergence proof in non-quasi-convex settings.

Also, from a methodological point of view, we believe that methods should be tested on the best available model, not on the approximation used inside the optimizer.

7.2 Summary of Contributions

We pointed out the different forms of errors, which explain suboptimal choices in power systems simulation and optimization. We advocated the use of tools with a stronger focus on model errors, rather than optimization error.

Direct Value Search. The first part of our work (chapter 4) deals with the multistage aspect of power systems. We proposed *Direct Value Search* (DVS), a tool combining ideas from Bellman decomposition and from *Direct Policy Search*. DVS combines:

- a polynomial time decision making, compliant with high-dimensional constrained action spaces;

- a simulation-based objective function, compliant with non-Markovian random processes and arbitrary models;
- an iterative global optimization procedure, providing an anytime approach.

This tool is convenient, anytime, stable, much more model-free than value-based methods and compliant with non-Markovian random processes.

Noisy optimization. The methodology above, DVS, as well as many DPS methods, involves a noisy black-box non-linear optimization component. In chapter 5, we contributed to the state of the art in non-linear optimization, in two directions:

- we proved rates for noisy optimization under some locality assumption.
- we improved empirical convergence rates with variance reduction techniques.

Convergence in non-quasi-convex problems. Finally, in chapter 6, we have provided – to the best of our knowledge – the first proof of linear convergence of evolutionary algorithms (a simple Evolution Strategy) in continuous domains on non quasi-convex objective functions.

7.3 Direct Value Search Benefits

In chapter 4, we presented Direct Value Search (DVS), a merge of Stochastic Dynamic Programming (SDP) and Direct Policy Search (DPS). It has many strengths:

- with DVS, forecasts naturally included in optimization;
- DVS can encode non-linear policies, indirectly, without losing the polynomial time DMR, as long as the decision making can be encoded as a linear problem. The model can be arbitrarily complex and can include small time scale effects without increasing the state space, whereas SDP-based methods have a precision loss when the number of time steps increases.
- DVS allows detailed simulations (e.g. with very small time scale, for volatility);
- DPS does not need convex Bellman values (whereas SDDP does). Stock-dependent efficiencies in pumped-storage systems is not a problem.
- DVS is an anytime algorithm (it can be stopped anytime, users immediately get a usable approximate solution);

- ADP, SDP and SDDP need Markovian random processes, which is not an issue for DVS. DVS can be applied on a problem where random processes are continuous and not Markovian. This is relevant for climate data.
- DVS can highly and rather easily be parallelized;
- Stopping criteria can be defined for SDDP [48, 41] or more generally SDP methods. These stopping criteria are, by nature, asymptotic, as well as stopping criteria of noisy optimization algorithms involved in our DVS method.
- DVS can handle large state spaces (as DPS);
- DVS can handle large action spaces (as SDP);

Thus, Direct Value Search can work on the "real" problem, without "simplification". To the best of our knowledges, there are no equivalent algorithm capable of solving large problems with such realistic models for an acceptable solving time.

7.4 Further work

Here we would like to mention several directions of future research which look the most promising. Some of these ideas were already discussed in the corresponding chapters and sessions. The further work includes:

Mathematical analysis of DVS. DVS has many advantages in practice. We started a work around the formalization of its advantages, in particular in case of non-Markovian random processes. Assuming that Π_θ is a sufficiently good approximator, we show that DVS can encode optimal policies, though the decision making is polynomial. Interestingly, we do not need the transition function used in the policy (Eq. 4.8) to be the same as the real transition - provided that the *simulations* are performed with the real transition.

More intensive testing of DVS. We tested Direct Value Search on a rather simple problem, for which the approach has given satisfactory results. The problem we have used in our baseline is not that much a toy problem: we applied a random generator without discretization, which is a challenge for many algorithms. Direct Value Search is, by design, less sensitive to dimension than classical methodologies and does not need any convexity assumption. Therefore our method should be tested on very high scale problems (hundreds of stocks and thousands of time

steps), showing its ability to handle non-Markovian random processes, non-convex Bellman values, high-scale state space. Among the other directions to explore, we are working on theoretical proofs for Direct Value Search.

Optimization at the level of investments. The investment problem have been described in section 1.3.3. It attempts to answer the following question: when, where and how much should we invest on new capacities (plants, transmission lines, ...) in order to optimize profits, keep the system in a reasonable condition and adapt it to the future changes of its environment on the long run ?

Long term investment studies are extremely important. They are involving investments in hundreds of billions of euros and they have to be made with huge non-stochastic uncertainties. Optimization at the level of investments is a challenging problem we have not considered so far, but it recently became one of our main goal.

Non-stochastic uncertainties. Such an extension in our algorithms is crucial for a better handling of the investment problem, discussed before and described in 1.3.3. Among relevant non-stochastic uncertainties, there are long-term technological breakthroughs and political and geopolitical uncertainties. This extensions of our work is critical for one of our current works, the POST[1] project, a high scale[2] and long term[3] investment optimization platform with huge (non-stochastic) uncertainties about the future (technologies, subsidies, laws, ...).

[1] *Plateforme d'Optimisation des Supergrids Transcontinentaux* `http://www.presse.ademe.fr/2013/06/les-reseaux-electriques-intelligents-innovent.html`

[2] Transcontinental scale, e.g. Europe and North Africa.

[3] horizon 2030 to 2050.

Bibliography

[1] France. Ministère de l'économie, des finances et de l'industrie and Fontaine, N., *Livre blanc sur les énergies*. Ministère de l'économie, des finances et de l'industrie, 2003. [Online]. Available: http://books.google.fr/books?id=dWwEHQAACAAJ

[2] E. Assoumou, "Modélisation markal pour la planification énergétique long terme dans le contexte français," Ph.D. dissertation, 2006. [Online]. Available: http://pastel.archives-ouvertes.fr/pastel-00002752

[3] Organisation for Economic Co-operation and Development (OECD), "Oecd factbook 2014." [Online]. Available: http://www.oecd-ilibrary.org/economics/oecd-factbook-2014_factbook-2014-en

[4] International Energy Agency (IEA), "World energy outlook 2012." [Online]. Available: http://www.oecd-ilibrary.org/energy/world-energy-outlook-2012_weo-2012-en

[5] J. Conca, "How deadly is your kilowatt? we rank the killer energy sources," http://www.forbes.com/sites/jamesconca/2012/06/10/energys-deathprint-a-price-always-paid/, 2012, accessed: 2010-09-03.

[6] J. Hansen, P. Kharecha, M. Sato, V. Masson-Delmotte, F. Ackerman, D. J. Beerling, P. J. Hearty, O. Hoegh-Guldberg, S.-L. Hsu, C. Parmesan *et al.*, "Assessing "dangerous climate change": required reduction of carbon emissions to protect young people, future generations and nature," *PloS one*, vol. 8, no. 12, 2013. [Online]. Available: http://pubs.giss.nasa.gov/abs/ha08510t.html

[7] B. D. Henri Wallard and P. Cornick, "After fukushima: Global opinion on energy policy," http://www.ipsos.com/public-affairs/after-fukushima-global-opinion-energy-policy/, 2012, accessed: 2010-09-03.

[8] M. Cooper and D. Sussman, "Nuclear power loses support in new poll," http://www.nytimes.com/2011/03/23/us/23poll.html, 2011, accessed: 2010-09-03.

[9] A. McCrone *et al.*, "Global trends in renewable energy investment 2013," http://fs-unep-centre.org/publications/global-trends-renewable-energy-investment-2013, 2013, accessed: 2010-09-03.

[10] A. Zervos *et al.*, "Renewables 2014 global status report," http://www.ren21.net/REN21Activities/GlobalStatusReport.aspx, 2014, accessed: 2010-09-03.

[11] A. Wood, B. Wollenberg, and G. Sheblé, *Power Generation, Operation and Control*. Wiley, 2013. [Online]. Available: http://books.google.fr/books?id=Iev9AAAAQBAJ

[12] S. Kazarlis, A. Bakirtzis, and V. Petridis, "A genetic algorithm solution to the unit commitment problem," *Power Systems, IEEE Transactions on*, vol. 11, no. 1, pp. 83–92, Feb 1996.

[13] N. Padhy, "Unit commitment-a bibliographical survey," *Power Systems, IEEE Transactions on*, vol. 19, no. 2, pp. 1196–1205, May 2004.

[14] G. Sheble and G. Fahd, "Unit commitment literature synopsis," *Power Systems, IEEE Transactions on*, vol. 9, no. 1, pp. 128–135, Feb 1994.

[15] D. Siface, M. Vespucci, and A. Gelmini, "Solution of the mixed integer large scale unit commitment problem by means of a continuous stochastic linear programming model," *Energy Systems*, vol. 5, no. 2, pp. 269–284, 2014. [Online]. Available: http://dx.doi.org/10.1007/s12667-013-0107-z

[16] A. Wood and B. Wollenberg, *Power generation, operation, and control*. Wiley, 1984. [Online]. Available: http://books.google.fr/books?id=dwRtAAAAIAAJ

[17] RTE, "Developpement durable," http://www.rte-france.com/fr/developpement-durable/les-enjeux-1/les-metiers-de-rte-et-les-enjeux-de-developpement-durable/les-metiers-les-enjeux, 2011, accessed: 2010-09-09.

[18] I. T. Sidonie Blanchard, "Les réseaux de transport et distribution d'électricité," http://www.developpement-durable.gouv.fr/IMG/pdf/16_-_Les_reseaux_de_transport_et_de_distribution_electricite.pdf, 2013, accessed: 2010-09-09.

[19] B. Wright, "A review of unit commitment," 2013.

[20] P. Pinson, "Renewable energy forecasts ought to be probabilistic," 2013, wIP-FOR seminar, EDF.

[21] P. Bornard, "Conduite d'un système de production-transport," *Techniques de l'ingénieur Réseaux électriques de transport et de répartition*, no. D4080, 2000. [Online]. Available: http://www.techniques-ingenieur.fr/base-documentaire/energies-th4/reseaux-electriques-de-transport-et-de-repartition-42263210/conduite-d-un-systeme-de-production-transport-d4080/

[22] International Energy Agency (IEA), "Variability of wind power and other renewables. management options and strategies," 2005.

[23] D. Bertsimas, E. Litvinov, X. A. Sun, J. Zhao, and T. Zheng, "Adaptive robust optimization for the security constrained unit commitment problem," *Power Systems, IEEE Transactions on*, vol. 28, no. 1, pp. 52–63, 2013.

[24] C. Autorita per l'Energia Elettrica e il Gas, "Report on the events of september 28th, 2003 culminating in the separation of the italian power system from the other UCTE networks," www.autorita.energia.it/docs/04/061-04all.pdf, 2004, accessed: 2010-09-09.

[25] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality (Wiley Series in Probability and Statistics)*. Wiley-Interscience, 2007.

[26] D. Bertsekas, *Dynamic Programming and Optimal Control, vols I and II*. Athena Scientific, 1995.

[27] R. Sutton and A. Barto, *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press., 1998.

[28] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, April 1994. [Online]. Available: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0471619779

[29] A. Dallagi and T. Simovic, "Optimisation des actifs hydrauliques d'edf : besoins métiers, méthodes actuelles et perspectives," 2012, pGMO.

[30] A. Couetoux, "Monte carlo tree search for continuous and stochastic sequential decision making problems," Ph.D. dissertation, 2013, thèse de doctorat dirigée par Teytaud, Olivier Informatique Paris 11 2013. [Online]. Available: http://www.theses.fr/2013PA112192

[31] R. E. Bellman, *Dynamic Programming*. Princeton, New Jersey, USA: Princeton University Press, 1957.

[32] R. Howard, *Dynamic Programming and Markov Processes*. Cambridge, Massachusetts: MIT Press, 1960. [Online]. Available: http://books.google.fr/books?id=fXJEAAAAIAAJ

[33] J. Nunen, *Contracting Markov Decision Processes*. Amsterdam : Mathematisch Centrum, 1976. [Online]. Available: http://repository.tue.nl/29937

[34] M. L. Puterman and M. C. Shin, "Modified policy iteration algorithms for discounted Markov decision problems," *Management Science*, vol. 24, no. 11, pp. 1127–1137, 1978.

[35] "Tight performance bounds on greedy policies based on imperfect value functions," Northeastern University, Tech. Rep. NU–CCS-93-14, Nov 1993. [Online]. Available: http://leemon.com/papers/1993wb2.pdf

[36] Y. Bengio, "Using a financial training criterion rather than a prediction criterion," CIRANO, CIRANO Working Papers 98s-21, 1998. [Online]. Available: http://ideas.repec.org/p/cir/cirwor/98s-21.html

[37] H. Heitsch and W. Römisch, "Scenario tree reduction for multistage stochastic programs," *Computational Management Science*, vol. 6, no. 2, pp. 117–133, 2009. [Online]. Available: http://EconPapers.repec.org/RePEc:spr:comgts:v:6:y:2009:i:2:p:117-133

[38] W.-B. Powell, *Approximate Dynamic Programming*. Wiley, 2007.

[39] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, p. 273–297, 1995. [Online]. Available: http://dx.doi.org/10.1007/BF00994018

[40] D. P. Bertsekas, "Dynamic programming and suboptimal control: A survey from ADP to MPC," *Eur. J. Control*, vol. 11, no. 4-5, pp. 310–334, 2005.

[41] M. V. F. Pereira and L. M. V. G. Pinto, "Multi-stage stochastic optimization applied to energy planning," *Math. Program.*, vol. 52, no. 2, pp. 359–375, Oct. 1991. [Online]. Available: http://dx.doi.org/10.1007/BF01582895

[42] A. Ruszczynski. North-Holland Publishing Company, Amsterdam, 2003, vol. 10, ch. Decomposition methods.

[43] A. Shapiro, "Analysis of stochastic dual dynamic programming method," *European Journal of Operational Research*, vol. 209, no. 1, pp. 63–72, 2011.

[44] K. Astrom, "Optimal control of Markov decision processes with incomplete state estimation," *Journal of Mathematical Analysis and Applications*, vol. 10, pp. 174–205, 1965.

[45] D. Bertsekas and J. Tsitsiklis, *Neuro-dynamic Programming.* Athena Scientific, 1996.

[46] V. Heidrich-Meisner and C. Igel, "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.* New York, NY, USA: ACM, 2009, pp. 401–408.

[47] Z. L. Chen and W. B. Powell, "Convergent cutting-plane and partial-sampling algorithm for multistage stochastic linear programs with recourse," *J. Optim. Theory Appl.*, vol. 102, no. 3, pp. 497–524, 1999.

[48] C. J. Donohue and J. R. Birge, "The abridged nested decomposition method for multistage stochastic linear programs with relatively complete recourse," *Algorithmic Operations Research*, vol. 1, no. 1, 2006.

[49] K. Linowsky and A. B. Philpott, "On the convergence of sampling-based decomposition algorithms for multistage stochastic programs," *J. Optim. Theory Appl.*, vol. 125, no. 2, pp. 349–366, 2005.

[50] A. Philpott and Z. Guan, "On the convergence of stochastic dual dynamic programming and related methods," *Operations Research Letters*, vol. 36, no. 4, pp. 450 – 455, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167637708000308

[51] V. Fabian, "Stochastic approximation of minima with improved asymptotic speed," *Ann. Math. Statist.*, vol. 38, no. 1, pp. 191–200, 1967.

[52] H.-G. Beyer, *The Theory of Evolutions Strategies*. Heidelberg: Springer, 2001.

[53] S. Astete-Morales, J. Liu, and O. Teytaud, "log-log convergence for noisy optimization," in *Proceedings of EA 2013*, ser. LLNCS. Springer, 2013, p. accepted.

[54] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)," *Evolutionary Computation*, vol. 11, no. 1, pp. 1–18, 2003.

[55] A. Conn, K. Scheinberg, and L. Toint, "Recent progress in unconstrained nonlinear optimization without derivatives," 1997. [Online]. Available: citeseer.ist.psu.edu/conn97recent.html

[56] B. Doerr and C. Winzen, "Towards a complexity theory of randomized search heuristics: Ranking-based black-box complexity," in *CSR*, ser. Lecture Notes in Computer Science, A. S. Kulikov and N. K. Vereshchagin, Eds., vol. 6651. Springer, 2011, pp. 15–28.

[57] S. Grünewälder, J.-Y. Audibert, M. Opper, and J. Shawe-Taylor, "Regret Bounds for Gaussian Process Bandit Problems," in *JMLR Workshop and Conference Proceedings : AISTATS 2010*, vol. 9, Chia Laguna Resort, Sardinia, Italie, 2010, pp. 273–280. [Online]. Available: http://hal-enpc.archives-ouvertes.fr/hal-00654517

[58] H.-P. Schwefel, *Numerical Optimization of Computer Models*. New-York: John Wiley & Sons, 1981, 1995 – 2^{nd} edition.

[59] H.-G. Beyer, "Mutate large, but inherit small ! On the analysis of mutations in $(1, \lambda)$-ES with noisy fitness data," 1998, pp. 109–118.

[60] J. Fitzpatrick and J. Grefenstette, "Genetic algorithms in noisy environments, in machine learning: Special issue on genetic algorithms, p. langley, ed. dordrecht: Kluwer academic publishers, vol. 3, pp. 101 120," 1988.

[61] D. V. Arnold and H.-G. Beyer, "Local performance of the (1+1)-ES in a noisy environment," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 30–41, 2002.

[62] D. V. Arnold and H. georg Beyer, "Evolution strategies with cumulative step length adaptation on the noisy parabolic ridge," Tech. Rep., 2006.

[63] U. Hammel and T. Bäck, "Evolution strategies on noisy functions: How to improve convergence properties," in *Parallel Problem Solving From Nature*, ser. LNCS, Y. Davidor, H.-P. Schwefel, and R. Männer, Eds., vol. 866. Jerusalem: springer, 9–14Oct. 1994, pp. 159–168.

[64] J. M. Fitzpatrick and J. J. Grefenstette, "Genetic algorithms in noisy environments," *Machine Learning*, vol. 3, pp. 101–120, 1988.

[65] M. Jebalia and A. Auger, "On multiplicative noise models for stochastic search," in *Parallel Problem Solving From Nature*, dortmund Allemagne, 2008. [Online]. Available: http://hal.inria.fr/inria-00287725/en/

[66] O. Teytaud and A. Auger, "On the adaptation of the noise level for stochastic optimization," in *IEEE Congress on Evolutionary Computation*, Singapour, 2007. [Online]. Available: http://hal.inria.fr/inria-00173224/en/

[67] R. Coulom, "Clop: Confident local optimization for noisy black-box parameter tuning," in *ACG*, ser. Lecture Notes in Computer Science, H. J. van den Herik and A. Plaat, Eds., vol. 7168. Springer, 2011, pp. 146–157.

[68] R. Coulom, P. Rolet, N. Sokolovska, and O. Teytaud, "Handling expensive optimization with large noise," in *FOGA*, H.-G. Beyer and W. B. Langdon, Eds. ACM, 2011, pp. 61–68.

[69] P. Rolet and O. Teytaud, "Bandit-based estimation of distribution algorithms for noisy optimization: Rigorous runtime analysis," in *Proceedings of Lion4 (accepted); presented in TRSH 2009 in Birmingham*, 2009.

[70] ——, "Adaptive Noisy Optimization," in *EvoStar 2010*, Istambul, Turquie, Feb. 2010. [Online]. Available: http://hal.inria.fr/inria-00459017

[71] V. Heidrich-Meisner and C. Igel, "Uncertainty handling cma-es for reinforcement learning," in *GECCO*, F. Rothlauf, Ed. ACM, 2009, pp. 1211–1218.

[72] L. Devroye, L. Györfi, and G. Lugosi, *A probabilistic Theory of Pattern Recognition*. Springer, 1997.

[73] R. Coulom, P. Rolet, N. Sokolovska, and O. Teytaud, "Handling expensive optimization with large noise," in *FOGA*, H.-G. Beyer and W. B. Langdon, Eds. ACM, 2011, pp. 61–68.

[74] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *J. of Global Optimization*, vol. 13, no. 4, pp. 455–492, 1998.

[75] J. Villemonteix, E. Vazquez, and E. Walter, "An informational approach to the global optimization of expensive-to-evaluate functions," *Journal of Global Optimization*, p. 26 pages, 09 2008. [Online]. Available: dx.doi.org/10.1007/s10898-008-9354-2http://hal-supelec.archives-ouvertes.fr/hal-00354262/en/

[76] L. Bienaymé, "Considérations à l'appui de la découverte de laplace," *Comptes Rendus de l'Académie des Sciences*, vol. 37, pp. 309–324, 1853.

[77] P. Chebyshev, "Sur les valeurs limites des integrales," *Math Pure Appl*, vol. 19, pp. 157–160, 1874.

[78] A. Markov, "On certain applications of algebraic continued fractions," Ph.D. dissertation, St Petersburg, 2002.

[79] A. V. D. Vaart and J. Wellner, *Weak Convergence and Empirical Processes*. Springer series in statistics, 1996.

[80] O. Teytaud and S. Gelly, "General lower bounds for evolutionary algorithms," in 10^{th} *International Conference on Parallel Problem Solving from Nature (PPSN 2006)*, 2006.

[81] H. Fournier and O. Teytaud, "Lower bounds for comparison based evolution strategies using vc-dimension and sign patterns," *Algorithmica*, vol. 59, no. 3, pp. 387–408, 2011.

[82] A. Auger, "Convergence results for (1,λ)-SA-ES using the theory of φ-irreducible Markov chains," *Theoretical Computer Science*, vol. 334, no. 1-3, pp. 35–69, 2005.

[83] A. Auger, M. Schoenauer, and O. Teytaud, "Local and global order 3/2 convergence of a surrogate evolutionnary algorithm," in *Gecco*, 2005, p. 8 p.

[84] D. V. Arnold and H.-G. Beyer, "Efficiency and mutation strength adaptation of the (mu/mui,lambda)-es in a noisy environment," in *Parallel Problem Solving from Nature*, ser. LNCS, M. S. et al., Ed., vol. 1917. springer, 2000, pp. 39–48.

[85] H.-G. Beyer, *The Theory of Evolution Strategies*, ser. Natural Computing Series. Springer, Heideberg, 2001.

[86] H. Chen, "Lower rate of convergence for locating a maximum of a function," *Ann. Statist.*, vol. 16, no. 3, pp. 1330–1334, 1988.

[87] J. Kiefer and J. Wolfowitz, "Stochastic estimation of the maximum of a regression function," *Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 1952.

[88] E. Vazquez, J. Villemonteix, M. Sidorkiewicz, and E. Walter, "Global optimization based on noisy evaluations: an empirical study of two statistical approaches," *Journal of Global Optimization*, p. 17 pages, 2008. [Online]. Available: dx.doi.org/10.1007/s10898-008-9313-yhttp://hal-supelec.archives-ouvertes.fr/hal-00354656/en/

[89] R. Cranley and T. Patterson, "Randomization of number theoretic methods for multiple integration," *SIAM J. Numer. Anal.*, vol. 13, no. 6, p. 904–914, 1976.

[90] M. Mascagni and H. Chi, "On the scrambled halton sequence," *Monte-Carlo Methods Appl.*, vol. 10, no. 3, pp. 435–442, 2004.

[91] X. Wang and F. Hickernell, "Randomized halton sequences," *Math. Comput. Modelling*, vol. 32, pp. 887–899, 2000.

[92] W. J. Morokoff, "Generating quasi-random paths for stochastic processes," vol. 40, no. 4, pp. 765–788, Dec. 1998. [Online]. Available: http://epubs.siam.org/sam-bin/dbq/article/31795

[93] J. Dupacova, N. Gröwe-Kuska, and W. Römisch, "Scenario reduction in stochastic programming: An approach using probability metrics," ser. Stochastic Programming E-Print Series, J. L. Higle, W. Römisch, and S. Sen, Eds. Institut für Mathematik, 2000, no. 20, published; Springer; Berlin [u.a.]; Mathematical Programming; 95; 2003; 3.

[94] P. Lavallée and M. Hidiroglou, "On the stratification of skewed populations," *Survey Methodology*, vol. 14, no. 1, pp. 33–43, 1988. [Online]. Available: http://www.amstat.org/sections/srms/Proceedings/papers/1987_142.pdf

[95] V. Sethi, "A note on optimum stratification of populations for estimating the population means," *Australian Journal of Statistics*, vol. 5, no. 1, pp. 20–33, 1963.

[96] M. Kozak, "Optimal stratification using random search method in agricultural surveys," *Statistics in Transition*, vol. 6, no. 5, pp. 797–806, 2004. [Online]. Available: http://www.researchgate.net/publication/229051808_Optimal_stratification_using_random_search_method_in_agricultural_surveys/file/d912f5062bc010dd58.pdf

[97] J. Linderoth, A. Shapiro, and S. Wright, "The empirical behavior of sampling methods for stochastic programming," *Annals OR*, vol. 142, no. 1, pp. 215–241, 2006. [Online]. Available: http://dblp.uni-trier.de/db/journals/anor/anor142.html#LinderothSW06

[98] S.-C. Huang, R. Coulom, and S.-S. Lin, "Monte-carlo simulation balancing in practice," in *Computers and Games*, 2010, pp. 81–92.

[99] C. Hamzaçebi and F. Kutay, "Continuous functions minimization by dynamic random search technique," *Applied Mathematical Modelling*, vol. 31, no. 10, pp. 2189–2198, 2007.

[100] Z. B. Zabinsky, "Random search algorithms," *Wiley Encyclopedia of Operations Research and Management Science*, 2009.

[101] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[102] H. Takagi and D. Pallez, "Paired comparison-based interactive differential evolution," in *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on.* IEEE, 2009, pp. 475–480.

[103] M. Strens and A. Moore, "Direct policy search using paired statistical tests," in *Proceedings of the 18th International Conference on Machine Learning.* Morgan Kaufmann, San Francisco, CA, 2001, pp. 545–552.

[104] M. Strens, A. Moore, C. Brodley, and A. Danyluk, "Policy search using paired comparisons," in *Journal of Machine Learning Research*, 2002, pp. 921–950.

[105] N. L. Kleinman, J. C. Spall, and D. Q. Naiman, "Simulation-based optimization with stochastic approximation using common random numbers," *Management Science*, vol. 45, no. 11, pp. 1570–1578, 1999. [Online]. Available: http://pubsonline.informs.org/doi/abs/10.1287/mnsc.45.11.1570

[106] V. de Matos, A. Philpott, and E. Finardi, "Improving the performance of stochastic dual dynamic programming," *Applications – OR and Management Sciences (Scheduling)*, 2012. [Online]. Available: http://www.optimization-online.org/DB_FILE/2012/07/3529.pdf

[107] A. Shapiro, W. Tekaya, J. P. da Costa, and M. P. Soares, "Risk neutral and risk averse stochastic dual dynamic programming method," *European Journal of Operational Research*, vol. 224, no. 2, pp. 375–391, 2013.

[108] M. Dowell and P. Jarratt, "The "pegasus" method for computing the root of an equation," *BIT Numerical Mathematics*, vol. 12, no. 4, pp. 503–508, 1972. [Online]. Available: http://dx.doi.org/10.1007/BF01932959

[109] N. I. M. Gould, D. Orban, and P. L. Toint, "Cuter and sifdec: A constrained and unconstrained testing environment, revisited," *ACM Trans. Math. Softw.*, vol. 29, no. 4, pp. 373–394, 2003.

[110] T.Back, F.Hoffmeister, and H.Schewefel, "A survey of evolution strategies," Dpt. of Computer Science XI, University of Dortmund, D-4600 , Dortmund 50, Germany, Tech. Rep., 1991.

[111] O. Teytaud, "Conditionning, halting criteria and choosing lambda," in *EA07*, Tours France, 2007. [Online]. Available: http://hal.inria.fr/inria-00173237/en/

[112] S. Droste, T. Jansen, and I. Wegener, "On the optimization of unimodal functions with the (1+1) evolutionary algorithm," in *Parallel Problem Solving from Nature - PPSN V*, ser. Lecture Notes in Computer Science, A. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, Eds. Springer Berlin / Heidelberg, 1998, vol. 1498, pp. 13–22, 10.1007/BFb0056845. [Online]. Available: http://dx.doi.org/10.1007/BFb0056845

[113] V. Vapnik and A. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," in *Theory of probability and its applications, 16:264-280*, 1971.

[114] R. Bergasse, "Stratégies d'évolution dérandomisées," Ecole Normale Supérieure de Lyon, Tech. Rep., 2007.

[115] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[116] A. Ostermeier, A. Gawelczyk, and N. Hansen, "Step-size adaptation based on non-local use of selection information," in *Parallel Problem Solving from Nature - PPSN III*. Springer, 1994, pp. 189–198.

[117] H.-G. Beyer and B. Sendhoff, "Covariance matrix adaptation revisited - the CMSA evolution strategy," in *Proceedings of PPSN*, G. Rudolph, T. Jansen, S. M. Lucas, C. Poloni, and N. Beume, Eds., 2008, pp. 123–132.

[118] H.-G. Beyer and W. B. Langdon, Eds., *Foundations of Genetic Algorithms, 11th International Workshop, FOGA 2011, Schwarzenberg, Austria, January 5-8, 2011, Proceedings*. ACM, 2011.

Appendix A

Appendix

Here are some useful definitions taken form `https://complexityzoo.uwaterloo.ca/Zoo_Glossary#ospeedup`:

adaptive Each question you ask (say to an oracle) can depend on the answers to the previous questions.

decision problem A problem for which the desired answer is a single bit (1 or 0, yes or no). For simplicity, theorists often restrict themselves to talking about decision problems.

function problem A problem where the desired output is not necessarily a single bit, but could belong to a set with more than 2 elements. Contrast with decision problem.

nondeterministic machine A hypothetical machine that, when faced with a choice, is able to make all possible choices at once - i.e. to branch off into different 'paths.' In the end, the results from all the paths must be combined somehow into a single answer. One can obtain dozens of different models of computation, depending on the exact way this is stipulated to happen. For example, an NP machine answers 'yes' if any of its paths answer 'yes.' By contrast, a PP machine answers 'yes' if the majority of its paths answer 'yes.'

o **("little-oh")** For a function $f(n)$ to be $o(g(n))$ means that $f(n)$ is $O(g(n))$ and is not $\Omega(g(n))$ (i.e. $f(n)$ grows more slowly than $g(n)$).

O **("big-oh")** For a function $f(n)$ to be $O(g(n))$ means that for some nonnegative constant k, $f(n)$ is less than $kg(n)$ for all sufficiently large n.

Ω **(Omega)** For a function $f(n)$ to be $\Omega(g(n))$ means that for some nonnegative constant k, $f(n)$ is greater than $kg(n)$ for all sufficiently large n.

Θ **(Theta)** For a function $f(n)$ to be $\Theta(g(n))$ means that $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ (i.e. they grow at the same rate).

polynomial To mathematicians, a polynomial in n is a sum of multiples of nonnegative integer powers of n: for example, $3n^2 - 8n + 4$. To computer scientists, on the other hand, polynomial often means upper-bounded by a polynomial: so $n + \log(n)$, for example, is "polynomial." Also an adverb ("polynomially"). A function that grows polynomially is considered to be 'reasonable,' unlike, say, one that grows exponentially.

quasipolynomial $O(2^{\log^c n})$, for some constant c.

subexponential Growing slower (as a function of n) than any exponential function. Depending on the context, this can either mean $2^{o(n)}$ (so that the Number Field Sieve factoring algorithm, which runs in about $2^{n^{1/3}}$ time, is "subexponential"); or $2^{o(n^\epsilon)}$ for every $\epsilon > 0$.

superpolynomial Growing faster (as a function of n) than any polynomial in n. This is not the same as exponential: for example, $n^{\log n}$ is superpolynomial, but not exponential.

lower bound A result showing that a function grows at least at a certain asymptotic rate. Thus, a lower bound on the complexity of a problem implies that any algorithm for the problem requires at least a certain amount of resources. Lower bounds are much harder to come by than upper bounds.

upper bound A result showing that a function grows at most at a certain asymptotic rate. For example, any algorithm for a problem yields an upper bound on the complexity of the problem.

tight bound An upper bound that matches the lower bound, or vice versa. I.e. the best possible bound for a function.

with high probability (w.h.p.) Usually this means with probability at least 2/3 (or any constant greater than 1/2). If an algorithm is correct with 2/3 probability, one can make the probability of correctness as high as one wants by just repeating several times and taking a majority vote. Note: Sometimes people say "high probability" when they mean "non-negligible probability."