Ca' Foscari
University
of Venice

Master's Degree programme

in Economics and Finance

Final Thesis

Adaptive evolutionary algorithms for portfolio selection problems: state of the art and experimental analysis

Supervisor
Ch. Prof. Giacomo di Tollo

Assistant supervisor
Ch. Prof. Marco Corazza

Graduand
Gianni Filograsso
Matriculation Number 878206

Academic Year
2020/2021

Abstract

This thesis aims at solving complex portfolio selection problems by introducing an adaptive strategy for parameter control in Evolutionary Algorithms (EAs), with the aim of achieving accurate and robust solutions. In our work, we review a broad set of parameter tuning and parameter control strategies, then we implement an adaptive policy, based on the parameter control technique proposed by Maturana et al. (2010), on a variety of non-convex risk measures, that display many local optima, for which traditional minimization strategies like gradient descent methods are not suitable. The idea behind this method is to solve problems by managing the well-known *EvE* balance in the context of evolutionary computation, which is widely acknowledged as a key issue in terms of search performance. This approach allows the EA to use an appropriate parameter setting at different stages of the search process, typically by generating large improvements of the solution quality at the beginning and finally by fine-tuning the solution. We apply this method to large scale optimization problems; in particular, we start by considering relatively basic programming problems with easy constraints, then we take into account a set of *NP-hard* integer programming problems, which display well-known computational issues.

Contents

Introduction

Over the past decades, Evolutionary Algorithms (EAs) have received considerable attention in academic research as powerful tools for solving complex optimization and search problems. Among many problem classifications, a useful distinction is the one discussed in Eiben and Smith (2015), in which they consider essentially the difference between combinatorial and numerical optimization problems (in the former case the *search space* -namely, the set of all possible solutions- S is continuous, in the latter it is discrete). EAs have been extensively exploited for a broad variety of problems, in many different fields: among continuous problems, we mention the portfolio selection problem (PSP) (Chang et al. (2000)) and feature selection for machine learning (Goldberg (1988)), whereas in the group of combinatorial problems we mention the Boolean satisfiability problem (SAT) (Lardeux et al. (2006)) and the vehicle routing problem (VRP) (Baker and Ayechew (2003)).

Among them, in our work we consider a variant of the portfolio selection problem (PSP) developed by Markowitz (1959): the basic idea of portfolio selection involves the selection of promising assets in terms of reward and risk and the allocation of capital to each of them. In particular, he formulates the PSP as a bi-criteria optimization problem in a mean-variance framework, under the hypothesis that the distribution of asset returns are fully characterized by their means, variances and covariances. Actually, there are two main issues with this approach (see e.g. Corazza et al. (2013) and Lwin et al. (2017)), which we recall briefly here:

- Asset returns are asymmetric and have excess kurtosis and the mean-variance framework cannot take fully describe the investor preferences. New risk measures are required to satisfy proper formal properties and to take into account non-normal empirical distributions; many of these risk measures are unfortunately nonconvex, nondifferentiable and nonlinear. Solving these optimization problems require ad-hoc reformulations, which cannot anyway accomodate integer constraints.

- The optimization problem with realistic and practical constraints is formulated as a mixed-integer programming problem (e.g. due to the presence of cardinality constraints); this problem has been proven to be *NP-hard* by Moral-Escudero et al. (2006). As we argue below, NP-hard problems provide a strong motivation for the application of heuristic approaches;

Altogether, in our work we implement a modified version of the PSP, which addresses jointly the two critical questions raised above. The issues arising from the application of integer constraints and complex risk measures provide strong support for the development of an efficient EA, by which it is possible to manage flexibly a variety of risk measures and problem instances. In particular, as we point out below, a major issue in this field is the design of the algorithm, by which it is possible to adjust its behaviour in a static or dynamic manner, depending on the nature of the problem to solve. The key idea behind EAs is to deal efficiently with complex problems at a reasonable computational cost: in general, this is exactly where evolutionary strategies typically fit in, i.e. they can deal with problems for which a compromise between the accuracy of the solution and computational cost must be found. In these

cases, a fast and exact solver does not actually exist: heuristics based on evolutionary computation, if appropriately tuned, can lead to a good or even near-optimal solution in a reasonable amount of time, for any given problem size. The flexibility and the efficiency of EAs on a wide range of problems is even more clear when *NP-hard* and *NP-complete* problems are taken into account: for these classes of problems, known algorithms currently require superpolynomial time (with respect to n input size) to solve the problem. A practical way of tackling these classes of problems is to resort to heuristics to generate good solutions. The quest for high-quality results at reasonable cost has subsequently laid the foundation of a field of study on evolutionary strategies (De Jong (1975), Goldberg (1988), and Holland (1992)). Essentially, an EA manages a population of individuals, which is gradually altered by variation operators, with the aim of converging as close as possible to an optimal solution with respect to a fitness function.

Thereafter, most efforts to improve the performance of EAs have focused on the *parameter setting problem* (Eiben et al. (1999)). In a nutshell, the choice of an optimal parameter setting that results in the best performance across different problem instances can be achieved in a 'static' fashion (Huang et al. (2020)), otherwise the parameter values may change over the run of an EA (Karafotias et al. (2015)), in a 'dynamic' fashion. Being dynamic and adaptive, they argue, EAs likely benefit from using different parameter values at different stages of the search process in terms of performance. Let us consider this aspect in further detail. di Tollo et al. (2015) observe that two key notions describe the behaviour of an EA: on the one hand, *exploitation* typically denotes the tendency of evolutionary algorithms to generate concentrated individuals in a specific area of the search space S, on the other hand *exploration* describes the tendency of the algorithms to spread new individuals in a larger area of S. In the field of parameter control, finding an optimal balance between exploration and exploitation (EvE dilemma) in order to achieve better performance is a matter of particular interest, which has drawn a special attention of many studies and is currently considered a key issue for the algorithm performance (Huang et al. (2020)).

Karafotias et al. (2015) find that there is an extensive literature on parameter control, focusing particularly on adaptive operator selection (AOS), which amounts to the selection of the optimal operator at the following iteration of the search process, given a set of variation operators. Some authors (Maturana et al. (2010) and di Tollo et al. (2015)) propose to adjust the EvE balance dynamically for combinatorial optimization problems, in order to improve search efficiency. They aim at achieving an optimal EvE balance through a *controller* that performs AOS, which is in charge of identifying a suitable operator at each iteration of the search, based on a compromise amongst quality and diversity criteria. There is an extensive literature on metaheuristics for portfolio selection problems; for instance, Corazza et al. (2013) test the basic PSO algorithm on a complex mixed-integer programming problem, whereas Kaucic (2019) tests the benefits of a variety of multiobjective PSO algorithms with risk parity control. Chang et al. (2000) and Chang et al. (2009) test a variety of metaheuristics, including basic genetic algorithms with a standard variation operator, while Lin and Liu (2008) evaluate a simple multiobjective genetic algorithm on a set of programming problems with minimum transaction lots. Finally, Lwin et al. (2017), Anagnostopoulos and Mamanis (2011) compare the performance of a set of popular multiobjective evolutionary algorithms, including a popular multiobjective approach based on genetic algorithms (NSGA-II). Most of the contributions based on genetic algorithms resort to a single variation operator, despite its crucial role in the exploration of the search space. Consequently, in our work we analyze further the benefits of managing the values of various parameters of an EA: for a start, we search and we evaluate the available literature in the field of *parameter setting*: the values of all the parameters of an EA impact greatly the efficiency of an algorithm (see e.g. Eiben et al. 1999), so we examine thoroughly two major classes of parameter setting, i.e. *parameter tuning*, by which the parameter values are selected before

the run and *parameter control*, by which the values are changed during the run. We present a definition for both of them and then we discuss the state-of-the-art algorithms. Though there is no one-size-fits-all strategy in EAs, we opt for an approach based on parameter control which has shown a good performance on a variety of combinatorial optimization problems: in our work we integrate a genetic algorithm with a dynamic strategy based on Adaptive Operator Selection (Maturana et al. (2010)) to select, at each step of the search process, an operator out of a pre-defined operator set. In particular, we investigate the properties of a set of variation operators and then we adopt the framework proposed in di Tollo et al. (2015), i.e. we implement a controller with the aim of finding a good EvE compromise during the search process, by choosing either an exploration operator or an exploitation one, on the basis of an external criterion defined by the user; then, we embed high-level strategies which guide the search dynamically, by changing the EvE balance over time. Finally, we employ the controller to solve a variety of portfolio selection problems, in order to assess the optimal portfolios and to evaluate the out-of-sample performance over real data.

The rest of the thesis is organized as follows. In chapter 1 we present a literature review of parameter tuning and parameter control methods. In chapter 2 we recall the basics of portfolio selection and we discuss some desirable properties of risk measures. Furthermore, we discuss the design of our mixed-integer programming problem and an unconstrained reformulation based on the exact penalty method (Bazaraa et al. (2013)), which is largely based on the discussion in Corazza et al. (2013). In chapter 3 we present a literature review of three classes of GA crossover operators, which lays the foundations for the discussion in chapter 4; in the last chapter, we show first some experiments in which we test the performance of simple genetic algorithms with different variation operators. Then, we comment the adaptive operator selection approach, which serves as a tool for performing new tests with a dynamic policy. Finally, we run a set of experiments based on a mixed-integer programming problem, in which we evaluate both in-sample and out-of-sample performance of the adaptive strategy.

Chapter 1

A literature review of parameter tuning and parameter control

In this chapter, we propose a literature review of strategies that address the parameter setting problem for evolutionary algorithms (EAs). The general class of evolutionary algorithms includes all the heuristic methods based essentially on the role of variation operators (recombination and mutation) and selection operators (Eiben and Smit (2011)), whose aim is to manipulate automatically a population of solutions. Given the general framework of EAs, which encompasses a broad variety of strategies, the user has to provide an application-specific design, by specifying a suitable set of *parameters*, i.e. 'all the details that make the EA concrete and executable' (Eiben and Smith (2015)). We introduce the topic with an informal discussion involving parameter setting in section 1.1, providing also some key concepts and the necessary terminology, while in sections 1.2 and 1.3 we take a deep dive into the topics of parameter tuning and parameter control. In both sections, we first provide a formal definition and then we examine some performance metrics; finally, we discuss the state-of-the-art methods in the field.

1.1 Overview

In this section we propose a very general overview regarding the algorithm setup: in particular, we deal with the notion of EA parameter in a sweeping framework. We want here to tackle the parameter setting problem, which has experienced an intense development recently, attracting the attention of many scholars. The first technical contribution in this field is definitely De Jong (1975)'s doctoral thesis, which recommends a set of 'proper' values for both the probabilities of single point crossover and (bit) mutation probabilities: the underlying idea is that, given these parameter values, one should expect a desirable performance. He basically determines a set of 'correct' values for some test functions as follows:

- population size: 50;
- probability of crossover: 0.6;
- probability of mutation: 0.001;
- generation gap: 100%;
- selection strategy: *elitist*.

However, an alternative approach, which is the one mainly discussed in this chapter following the taxonomy of Eiben et al. (1999), highlights a simple evidence: specific problems require specific EA setups, that is, an 'optimal parameter setting' has a

very narrow meaning, so empirical/experimental procedures, which aim at being as general as possible, typically fail to address a wide range of optimization problems. Note indeed that for new problems a specific setup may turn out to be poor (see for instance Huang et al. (2020), Eiben and Smit (2011)): a crucial contribution to this topic is the so-called *No Free Lunch Theorem (NFL)* Wolpert and Macready (1997) which states in a nutshell that there is not an universal algorithm which suits every optimization problem well. The underlying intuition is that the parameter setting problem is not a one-time quest, instead it must be tailored properly when new problems are met. As a consequence, the parameter setting is a task that has to be considered over and over again, because an algorithm, given NFL, on average cannot outperform other algorithms on any conceivable problem. Eiben and Smith (2015) restate the theorem by noting that it is referred to nonrevisiting (i.e. a point in the search space is generated only once) and black box algorithms, namely those that do not include instance specific knowledge.

Furthermore, interaction between parameters is another critical issue when dealing with an optimal setting: indeed, parameters are not independent; tuning 'by hand' or 'by analogy' typically cannot deal with interaction between parameters, which are somewhat complex and not easily explicable as well. Moreover, note that these empirical methods are for sure unfit to manage properly a wide array of algorithm setups. Suppose to use a 'brute force' algorithm to determine the fittest combination of parameters; this method has a $O(n!)$ time complexity, i.e. testing four different values for, say, eight parameters requires $4^8 = 65536$ different and independent runs. A rigorous quest for performance must take into account at least 'quality' (see below) and speed. Eiben and Smith (2015) summarizes this concept as follows: one may identify quality by means of a fitness function, which is usual and straightforward. Speed is instead typically identified by CPU time or wall-clock time; then at least three combinations of them can be used to determine the algorithm performance. As a consequence, the user could:

1. Set a maximum running time and define the performance as the best fitness for that given amount of time;

2. Set a minimum fitness level and evaluate the time required to reach that amount;

3. Exploit strategy (1) and strategy (2): a run is successful if both targets are reached, in terms of time and quality.

To keep this overview very general, consider now the problem of finding 'desirable' values for an EA, i.e. determining a 'good' setting. Eiben and Smith (2015) highlight that the design of an EA is an optimization problem itself, so a more rigorous treatment of 'parameter' must be given. As a consequence, they propose a specific convention, in order to distinguish those parameters whose domain is ordered and usually a subset of $\mathbb{R}$, like the mutation probability, which they call *numeric parameters*, and all those parameters which are characterized by an unordered domain, i.e. a set with no distance metric (for instance, a set of crossover operators {onepoint, uniform, averaging}), called *symbolic parameters*. Another convention, which is of practical importance, is used here. We treat an EA as an algorithm with his own operators, i.e. symbolic parameters. If only one of these is different, then we are handling another algorithm; instead, if only one value of the problem is modified, we have a new *problem instance*. Moreover, following Eiben and Smith (2015), two EAs differing only in their numeric parameters (e.g.population size) are considered as variants of the same EA, so that an EA is a partially specified algorithm: in other words, an EA is fully defined by its symbolic parameters, while its exact specification (a variant) is characterized by numeric parameters.

We also highlight here that some authors (for instance Hoos (2012)) propose a in-depth classification of parameters, though we will mostly refer to the one of Eiben and Smit (2011), discussed before:

1. *Categorical parameters* have a finite set of discrete and unordered values, whose goal is to select from a collection of mechanisms;

2. *Boolean parameters* are used to disable or enable certain heuristics, so they typically assume integer values;

3. *Ordinal parameters* are parameters that assume only discrete and ordered values;

4. *Conditional parameters* are those parameters whose state is conditional to the value of other parameters; for instance, they arise when mechanisms are activated using some parameters, whose behaviour in turn is conditional to the behaviour of other parameters.

Therefore, we can now move on by recalling the distinction between two ways of setting parameters, namely *parameter tuning* and *parameter control*. Another way of classifying parameter setting methods, which is mainly a matter of terminology, is the one discussed often in machine learning literature; one may consider *offline tuning* and *online tuning* as Alpaydin (2019) does:

- Offline tuning (parameter tuning) is a method for determining good parameter values before running the algorithm; then the optimal setting found in the process is reused to solve a given problem and is left unchanged for the whole run;

- Online tuning (parameter control) is a method which normally starts with a pre-defined set of parameter values (which can be basically tuned 'by analogy' or 'by hand') and then some values are allowed to vary according to a certain goal defined by the user during the search process.

In general, there is no one-size-fits-all parameter management strategy that yields good results for every problem: consequently, a few issues are raised by the definitions provided above and will be discussed thoroughly in the upcoming sections. That holds true in general, whichever the problem at hand and the selected model. Here we note that parameter tuning is normally very time-consuming but at the same time it proves to be very general; on the other hand, parameter control, though more problem specific, provides a greater degree of flexibility in the search process, so that, given the inherently dynamic nature of EA, one may set different parameter values according to his own will, which is based on the algorithm performance: more explorative features are preferred to exploitative approaches (based either on symbolic or numeric parameters) in the early generations (Angeline (1995), Bäck (1993), Davis (1989), Hinterding et al. (1997)).

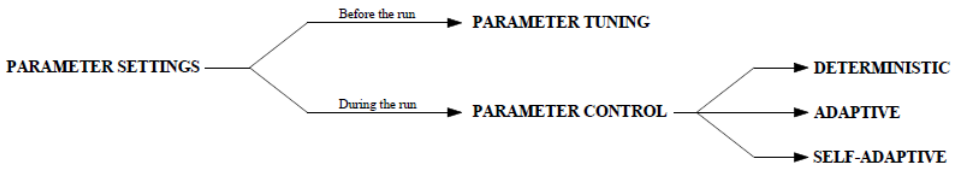


Figure 1.1: Classification of parameter setting techniques, adapted from Eiben et al. (1999)

Finally, one further distinction (see Figure 1.1) involving parameter control has to be considered here: those methods (i.e. parameter control) that involve changing parameter values are generally are usually classified in three different categories. We propose here the taxonomy introduced first by Eiben et al. (1999):

- *Deterministic parameter control*: this method typically alters certain parameter values according to a deterministic rule, without employing any feedback from the search process. Nonetheless, in order to design a rational strategy, rules are often time-based and will be typically linked to the number of generations; in this way, the user implicitly defines an 'expectation' about the desired behaviour of an operator (for instance, it is supposed to search new areas as time passes and so on).

- *Adaptive parameter control*: this method is based on a feedback procedure from the search process, namely from the historical data involving the behaviour of the operators. This information is stored and sent to a credit assignment module, which establishes the direction of the algorithm. Finally, a selection module is expected to pick a value according to a certain rule, which should take into account both the credits and to a certain extent, also some randomness to favor a more explorative attitude.

- *Self-adaptive parameter control*: the underlying idea here is to force the parameters to undertake recombination and mutation, so that the well-known rule involving the survival of the fittest can be applied to them too, so that better individual are likely to survive and produce in turn better offspring; this approach, if properly tuned, may actually lead to self-propagation of 'good' solutions, without any intervention from the user.

1.2 Parameter tuning: a literature review

In this section we provide a broad review of the literature involving the topic of parameter tuning; though we will not provide a very detailed analysis, we aim at offering an updated review. Currently, the parameter tuning literature is pretty 'stable': starting from the end of the nineties, a wide array of new techniques has been presented, which are now intensively applied. Differently, the parameter control literature looks still in full development, with still more room for improvement. We find that Huang et al. (2020) and Eiben and Smit (2011) provide at the moment a very thorough survey of automated parameter tuning methods. Though we will refer directly to the methods proposed in literature, we follow their taxonomy and their logical steps here. As a consequence, we propose a similar classification of tuning methods, focusing in particular on the distinction amongst *generate and evalute methods*, *iterative generate and evaluate methods* and *high level generate and evaluate methods*.

1.2.1 A formal definition of the parameter tuning problem

We move forward now by taking into account a more formal and rigorous statement of the configuration problem. Though many versions have been provide over time, we state a definition (e.g. Birattari et al. (2002), Hoos (2012)) which has been already employed over time in literature.

Given:

- A parametrized algorithm A with $p_1, \ldots, p_n$ parameters affecting its behaviour;

- A space C of configurations, where each configuration $c \in C$ specifies values for the parameters of A, whose behaviour on a given problem instance is therefore unambiguous (i.e. specified);

- A set of problem instances $I = i_1, \ldots, i_m$;

- A performance metric m that measures the performance of A on the instance set I $\forall$ c configurations.

The problem is aimed to find a configuration $c^\star \in C$ which shows an optimal performance of A on I, for a user-defined performance metric m.

We define with $A(c)$ the target algorithm A under a certain configuration c, completely described by a set of values $\theta_{i1}, \ldots, \theta_{ij}$, which is called *domain*, which contains a set of values for a given configuration c_i.

1.2.2 Evaluating the tuning algorithm

So far, we have only sketched a formal definition of the tuning problem, but we have neglected a deeper discussion involving each part of it; in particular there are some open questions of practical relevance. In order to tackle them, we consider now a larger taxonomy and also how to manage practically the evaluation of an algorithm, in terms of performance and quality. Note that a further distinction between symbolic and numeric parameters is implicitly contained in the above mentioned definition in section 1.1.

As far as algorithm quality is concerned, we refer the reader to section 1.1 for a broad discussion of the topic. Here we raise, with respect to parameter tuning problems, some more specific points. For instance, Eiben and Smit (2011), Eiben and Smith (2015) and Huang et al. (2020) cope with both performance and robustness. First of all, the EAs are stochastic algorithms, so the performance measure is consequently random and typically computationally intensive, given the inherently stochastic nature of EAs and the randomness in problem instances selection. This amount of stochasticity has to be managed and analyzed carefully: the estimation of the expected performance is usually inferred by Monte Carlo techniques, by means of which one can generally keep variability under control. Furthermore, when evaluating an algorithm, one should also break down its quality into a series of drivers, typically performance and robustness.

Beginning from the former, one may aggregate the measures mentioned in section 1.1 to get a performance metrics, giving a grasp of the global performance.

- *MBF* (mean best fitness)
- *AES* (average number of evaluations to solution)
- *SR* (success rate, i.e. the ratio of runs with required quality to the total number of runs)

Hence, many paths are available: one may also take a different standpoint and choose to maximize a combination of these performance metrics (i.e. *SP* Success Performance in Eiben and Smit (2011)), for instance by performing linear aggregation of the above mentioned quantities. A key point here is to acknowledge that a 'good' or a 'poor' tuning process could be highly sensitive to the performance measurement. One further distinction should take then into account the notion of robustness: intuitively, it can be seen as a measure of 'flexibility', namely it amounts to the variance of the algorithm performance with respect to some dimension. We can summarize this informal idea as follows:

- *Robustness with respect to problem spefication*: suppose to tune an EA (say, A) on a function f. Then the algorithm (or, to be more precise, a target algorithm, as it is tested on a specific configuration c: the notion of robustness is related to pre-specified EA instances) is evaluated with respect to the parameter vector and to f, namely it is a *specialist* algorithm (a more rigorous treatment is available in Smit and Eiben (2010a)), because it is a target algorithm which shows good performance with only one problem instance. Often, the EA designer wants the algorithm to be as general as possible, albeit, under the NFL theorem this is a dead-end quest. The target algorithms tuned on a test suite of $f_1, \ldots, f_n$ functions are called *generalist* if they show good performance as the problem instance changes. More in general, the core difference is that the performance metrics are determined according to a different approach:

  - A *specialist* target algorithm is tuned on one problem instance, so a very rough evaluation of it could be based on the mean fitness calculated on a series of runs on that specific instance. For sure, it is much more likely to find a good parameter setting for a specialist target algorithm, i.e. a problem-specific solution;
  - A *generalist* target algorithm is based on a set of instances, so the performance has to be evaluated in some way (i.e. via normalization) with respect to a whole set of instances (say, the average fitness of each one).

- *Robustness with respect to parameter values*: any variation to parameter values affects the specific algorithm instance $A(c)$, whose performance is based in terms of 'tuneability' and 'tolerance' (Eiben and Smit (2011)): an EA characterized by a volatile quality as the parameter vector varies, is 'tuneable' because it can considerably improve by choosing proper values. Note that here we measure quality not in terms of fitness, but with respect to parameters, namely we evaluate the quality of parameter vectors for a certain problem;

- *Robustness with respect to random number generator*: the Success Rate measure mentioned before is typically useful to evaluate the robustness with respect to the 'stochasticity' of the algorithm: one can obtain information about it by considering the ratio of runs with the same setup but with changing random seed, so one can develop a success rate, which takes into account results above a user-defined threshold T and those below, namely the difference between the worst and best runs: if big, then the EA instance is 'unstable', otherwise it is called successful.

1.2.3 Tuning methods

In the last two decades, much effort has been devoted to the topic of parameter tuning, which seems pretty obvious in light of the challenges discussed in section 1.1. At the moment, a broad range of methods is available: hence, starting from the conceptual framework presented in Eiben and Smit (2011), which proposes a detailed taxonomy, many surveys (see, for example, Dobslaw (2010), Hoos (2012), Eiben and Smit (2012), Montero et al. (2014) and Huang et al. (2020)) appeared since then; the research activity in this field seems much more mature now.

A serious challenge here regards the choice of a proper taxonomy: while Dobslaw (2010) stresses the difference between model and model-free methods, Eiben and Smit (2012) focus their attention on a taxonomy based on search effort measurement. Tuning algorithms classification hinges on the allocation of search effort spending, according to a desidered direction. In a nutshell:

- Methods which use a small number of parameter vectors, i.e. 'optimizing the spending' (Nannen and Eiben (2007));

- Methods using a small number of tests per parameter vector (Balaprakash et al. (2007));

- Methods that use a small number of parameter vectors and tests (Yuan and Gallagher (2007));

- Methods using a small number of function evaluations, i.e. optimizing the number of fitness evaluation per EA run. Currently, this last subclass is purely 'theoretical' (proposed in Eiben and Smit (2012)) as, to our knowledge, no methods have been developed for this goal.

One further classification is the one of Montero et al. (2014), which is based on a 'historical' classification, spanning from De Jong (1975)'s seminal work to more recent developments. They basically detect five classes, as follows:

- Hand-made tuning (e.g. De Jong (1975));
- Tuning by analogy, a family in which De Jong (1975)'s PhD thesis is included;
- Experimental design-based tuning, namely those studies based on experimental design to set the parameters values (e.g. Birattari et al. (2002), Balaprakash et al. (2007));
- Search based tuning, i.e. meta-EAs, which includes all those evolutionary algorithms employed to optimize in turn other EAs, like Revac (Nannen and Eiben (2007)) and ParamILS (Hutter et al. (2007));
- Hybrid tuning includes Calibra (Adenso-Díaz and Laguna (2006)) method, which combines local search techniques and and experimental design tuning.

Here we focus our attention on the most recent taxonomy presented in Huang et al. (2020), which is based on the idea that every tuning method is based on a *generate and evaluate* principle (generate a configuration space C and then evaluate according to performance metrics m); accordingly, the currently available tuning methods are classified into three categories:

- *Simple generate-evaluate methods* (e.g. Birattari et al. (2002), Birattari (2003)) include all the approaches made up of two simple steps, namely a generate step, by which the candidate configurations are generated (for instance, by using experimental design strategies). Finally, the candidates are evaluated according to a performance metrics;
- *Iterative generate-evaluate methods* (e.g. Hutter et al. (2007), Nannen and Eiben (2007), Balaprakash et al. (2007)), by which the generate-evaluate steps are iterated, so that new small groups of candidates are generated at each step and not all at once. The subgroups are then evaluated to find a 'best-so-far' candidate to be compared in successive steps;
- *High-level generate-evaluate methods* (e.g. Yuan et al. (2013)), by which a set of elite candidates is generated with traditional tuners (like F-Race), then the best configuration is selected among elite candidates through evaluation.

1.2.3.1 Simple generate-evaluate methods

Here Huang et al. (2020) include all the methods that provide the most straightforward process of parameter setting. It is basically made up of two core steps:

1. Generate a population of candidate configurations;
2. Evaluate configurations to find a 'good' setting.

Brute force and F-Race are included in this section. Here we borrow from machine learning the distinction between the 'training phase' and the 'testing phase'; in the first one the parameter setting is determined according to a performance measure, for a given set of traning instances. In the latter the configuration is tested on new problem instances. The configuration chosen, which is the result of the calibration process carried out in the training phase, is expected to perform well across different (new) problem instances.

1. *Brute-Force*

   The Brute Force belongs to the category of *simple generate-evaluate methods* because, despite being a 'naive' way of tuning algorithms, it estimates the optimal configuration by testing each candidate on a large set of training instances and on a large number of runs; the candidate setting with best expected performance is chosen as the optimal configuration. The set of parameter configurations are

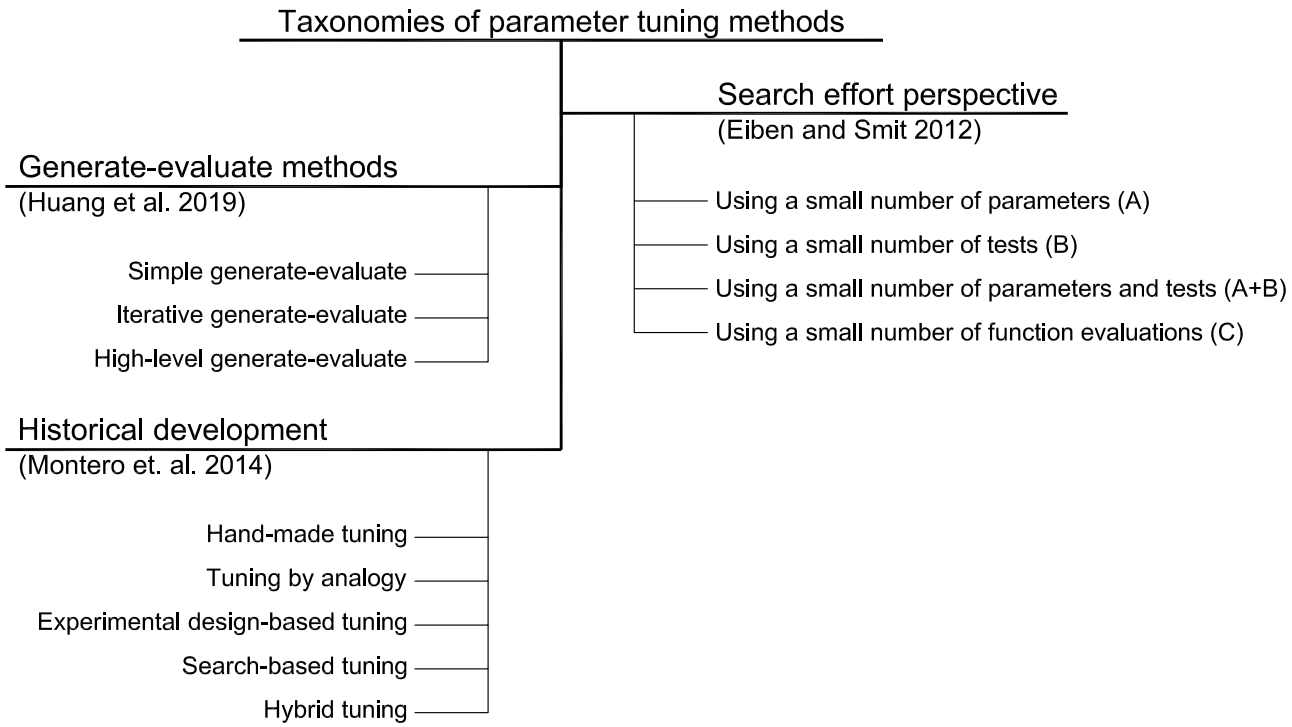


Figure 1.2

usually generated by *full factorial design* (FFD), so that it cointains all combinations of values or levels, using *design of experiments* (DOE) jargon, for a set of discrete parameters. Note that brute force allocates computational resources equally to each candidate, including the poor performing ones. Though being easy to apply, the computational power is not spent with parsimony and, as a consequence, this method is typically recommended only for small-scale problems. Furthermore, as noted by Birattari et al. (2002), there is not a proper criterion to decide how many runs should be performed to manage the stochasticity of the target algorithm.

2. *F-Race*

   The racing method has been proposed first by Birattari et al. (2002) and then implemented in Birattari (2003), based on racing methods[1]. The idea is to provide a better allocation of computational resources among candidate configurations, by reducing the resources allocated to poor configurations. F-Race evaluates a set of candidate configurations for each problem instance, eliminating those settings which show poor performance.

   F-Race exploits the *Friedman test*, a non-parametric statistical test based on ranking; hence, hypotheses on the distribution of observations are not required.

[1]The essential idea of racing methods is to look for good configurations, starting from an initial set of candidates and discarding the worst performing configurations as soon as statistically sufficient evidence is gathered against them.

It is used to nail differences in performance among candidate parameter configurations. A blocking design is then implemented, i.e. the units are organized in blocks and a blocking factor is specified (i.e. a source of variation), which in this case is the problem instance. Birattari et al. (2002) highlight the fact that EAs are intrinsically stochastic and both instances and configurations may be very different with one another. The blocking process helps to manage properly a disturbing source of variation, by focusing the attention on different configurations within each instance, removing the impact caused by differences among instances, i.e. normalizing the performance metrics observed on different instances. The null hypothesis that all rankings within each block are equally likely is tested with the following test statistic:

$$T = \frac{(n-1)\sum_{j=1}^{n}(R_j - ((m(n+1))/2))^2}{\sum_{l=1}^{m}\sum_{j=1}^{n}(R_{ij}^2 - (m(n+1)^2)/4} \tag{1.1}$$

where $T \sim \chi^2(n-1)$, m steps of the race, n parameter configurations (candidates), R_{ij} is the rank of parameter configuration c_j within block l and $R_j = \sum_{l=1}^{m} R_{lj}$, with an user defined confidence level α. We can summarize the test hypothesis as follows:

$$\begin{cases} H_0: \text{ 'All rankings within each block are equally likely'} \\ H_1: \text{ 'At least one candidate shows a better performance than one another'} \end{cases}$$

F-Race can be sketched as follows: first, as always, a set C of configurations is generated. After each evaluation round of the candidate configurations, the Friedman test is performed to check if at least one candidate is significantly different from others in terms of performance. If so (i.e. null hypothesis is rejected), then pair-wise comparisons between the best ranked and each other configuration are performed, and all the candidates showing worse performance are removed from the set of candidate configurations C and will not appear in the following evaluation steps.

Algorithm 1: F-Race (simplified version)

Input : C: Set of candidate configurations
α: Confidence level
max_iter: Maximum number of iterations

```
i = 0, S_θ = C, C_θ = ∅
while i < max_iter do
    dataset=RandomSampled()
    for c ∈ S_θ and |S_θ| > 1 do
        C_θ = EvalSolution(max_iter, c, dataset)
    end
    i = i + 1
end
Remove inferior (in terms of resulting quality) candidate configurations from
  S_θ by Friedman test for a given α confidence level.
```

1.2.3.2 Iterative generate-evaluate methods

The methods falling in this category are basically all those tuners which execute iteratively the *'generate & evaluate'* steps (Huang et al. (2020)). The underlying idea is to exploit the information gained in the previous iterations to explore more effectively the best candidates solutions. Though many other subclasses could be detected, here we limit ourselves to the broader class of iterative tuners, focusing in particular on those methods that use some heuristic rules to generate new candidates.

1. *Relevance Estimation and Value Calibration*

   The *Relevance Estimation and Value Calibration (Revac)* method has been introduced by Nannen and Eiben (2007) and it is often included in the class of *Meta-EAs*, due to the fact that metaheuristics are well-suited to find an approximate solution to complex problems like parameter tuning.

   An intuitive understanding of this method typically starts from a graphical representation of the problem (see table 1.1).

   Table 1.1: A Revac table X_t updated at step t

	$\mathcal{D}(\theta^1)$	$\cdots$	$\mathcal{D}(\theta^i)$	$\cdots$	$\mathcal{D}(\theta^k)$
$\mathbf{c_1}$	$\{\theta_1^1$	$\cdots$	θ_1^i	$\cdots$	$\theta_1^k\}$
$\vdots$		$\ddots$			
$\mathbf{c_j}$	$\{\theta_j^1$	$\cdots$	θ_j^i	$\cdots$	$\theta_j^k\}$
$\vdots$				$\ddots$	
$\mathbf{c_m}$	$\{\theta_m^1$	$\cdots$	θ_m^i	$\cdots$	$\theta_m^k\}$

   If one looks at the table horizontally, then a series of m parameter vectors (i.e. configurations $C = \{\mathbf{c_1}, \ldots, \mathbf{c_m}\}$) are shown, while vertically the representation of each parameter value is shown in each column (for instance, in the first column the parameter θ_1 is shown for different configurations and so on).

   Consider now the columns in table 1.1: $\mathcal{D}(\theta_i)$ represents a marginal density function defined on the parameter θ_i. The k columns define in turn a joint distribution C: this interpretation turns out to be very powerful in the framework proposed by Nannen and Eiben (2007). When each parameter is initialized, one gets a set of k uniform marginal densities. Then, as we will discuss soon, these densities are 'altered' in the following steps, during the search process. The gist of their method is to measure the so-called Shannon entropy of these $\mathcal{D}(\cdot)$ marginal densities, which can be computed as follows:

   $$H(\mathcal{D}_{[a,b]}) = -\int_a^b \mathcal{D}(x) log_2 \mathcal{D}(x) dx \tag{1.2}$$

   Note that we use a continuous domain definition of Shannon entropy, as this is required for all those parameters which are real-valued, like the mutation rate; in order to compare the entropy of distributions of different parameter (as each one may be defined on very different subset of $\mathbb{R}$, the range of all parameters is normalized to the unit interval $[0, 1]$ (Smit and Eiben (2010b)). In this way the uniform distribution has entropy $H(\mathcal{D}_{[0,1]}) = 0$ and definitely it is negative for any other distribution.

   The entropy measure could be intuitively interpreted (in general) as the amount of disorder in a system or, more specifically in information theory, as the uncertainty associated with a random variable. In general, as the 'information' contained in a marginal density grows, the importance of a parameter increases, as the entropy of the marginal distributions is typically considered an indicator of parameter relevance, which can be used by the user to allocate the resources for their tuning by providing more resources to to those with a higher degree of relevance. The *Revac* procedure normally starts with k marginal uniform densities, so with the entropy set for every column to zero. Then, the distributions are updated as each iteration gives a higher probability to regions with higher performance and generally, those with 'sharper' peaks have lower level of entropy: in layman's terms, the difference between the uniform distribution and a maximum entropy distribution for a given level of performance can be interpreted as the minimum amount of 'information' to reach that pre-established level of performance.

Suppose now to start with a X_0 table, whose values have been drawn from a random uniform distribution. The updating process at time $t+1$ from t generates a new table X_{t+1}: basically this process can be broken down in two main steps. First, one should evaluate each parameter configuration among a set of instances (1); then at each generation an evolutionary algorithm is employed to generate a new population, which is expected to be 'better', according to some quality metrics (2).

This 'Meta-EA' generates at each iteration one new parameter configuration (or *child* configuration): the user chooses n (so that $n < m$) parent vectors from current population, it undergoes recombination and mutation and then it replaces an element of the population. The recombination is carried out by means of a multi-parent uniform crossover operator, i.e. creating one child from n parents. The mutation operator is made up as follows: for each parameter a mutation interval is calculated and then a number from a random uniform distribution defined on this interval, whose bounds are basically computed according to the lowest/largest neighbor value of the parameter k. Note that four parameters must be necessarily defined in the Revac procedure, namely the size of population m, the size of crossover and mutation operators n and h, finally the maximum number of executions as a stopping criterion.

Algorithm 2: Revac (simplified)

Input : m: size of parameter configurations
n: size of crossover operator
h: size of mutation operator
max_iter: maximum number of iterations

Generate m configurations $c_{1,\dots,m}$ drawn from a random uniform distribution;
Evaluate each parameter configuration c_j;
$i = 0$
while $i < max\_iter$ **do**
  c_{child}=uniform multi parent crossover;
  c_{child}=uniform multi parent mutation;
  evaluate configuration c_{child};
  replace oldest parameter configuration$\in C$;
  calculate entropy of each parameter $H(\theta_i^j)$
  $i = i + 1$
end

2. *ParamILS*

ParamILS has been proposed in Hutter et al. (2007). The core point of this method is to exploit a well known heuristic, the *Iterated Local Search* (ILS), already developed for many combinatorial optimization problems. The gist of this approach is to carry out a procedure based on two set of phases (Hoos (2012)):

- A number of phases of local search designed to reach a local optima for a given problem instance;
- A number of phases of perturbations interposed with the local search procedure, in order to escape from local optima.

Basically, starting from a local optimum, in each iteration a perturbation phase is performed, followed by a new local search phase, with the purpose of ending up to a new local optimum. Hence, a comparison is required to decide whether to continue the search process or to move back to the previous local optimum (so-called acceptance criterion). Generally, ILS methods are designed so that

the heuristic visits new candidate configurations and meanwhile it stores the best solution found so far.

Here we want to focus only on four main features of the algorithm design:

- The initialization procedure;
- The local search phase;
- The perturbation phase;
- The restart mechanism.

The initialization described in Hutter et al. (2009) is based on combination of a user-defined configuration (for example by-analogy) and some other r configurations randomly chosen; these $r + 1$ configurations are then evaluated and the best-performing configuration is picked as a starting point for the iterated local search. The authors show that for $r > 0$ a better performance can be achieved, compared to a only user-defined parameter setting (i.e. $r = 0$). Hoos (2012) proposes to implement a more advanced technique for the initialization procedure (for instance a racing method).

The ParamILS method performs iterative first-improvement search, which differs from best-improvement local-search in the sense that a solution is chosen uniformly at random in the neighborhood, whereas the best-improvement search looks for the best solution in the neighborhood. The perturbation phase performs s (randomly chosen from a uniform distribution) steps in the same neighborhood used in the local search phases, using typically (Hutter et al. (2009)) a few steps, according to the experiments performed by the authors. Finally, a restart mechanism is added with a low fixed probability p_r; at the end of each iteration the current configuration with probability p_r is replaced with another one, drawn from a random uniform distribution, which serves as a new starting point for the search process. Hoos (2012) pinpoints that this mechanism brings additional diversification to avoid a 'stagnant behaviour'.

3. *Iterated F-Race*

This method has been proposed by Balaprakash et al. (2007) and is meant to solve some of the drawbacks of the F-Race procedure with a supplementary mechanism; here we outline only the key differences and improvements. One of the main shortcomings of F-Race is that it struggles -for a given reasonable amount of computational budget- to manage a large amount of candidate configurations. The standard F-Race procedure is based on a so-called full factorial design, according to which, in oversimplified terms, the initial configuration set C_0 cointains all combinations of values for a set of discrete parameters, so that the computational burden grows rapidly as the number of parameter increases. Furthermore, the FFD requires the user to determine *a priori* the levels of each parameter.

A tweaked version of the plain vanilla F-Race is based on RSD, according to which the initial set configurations C_0 is determined with a sampling process, according to some probability model (usually a uniform distribution). Balaprakash et al. (2007) show that RSD/F-Race outperforms significantly the FFD/F-Race in many combinatorial applications. The Iterated F-Race is based on the iterative application of the F-Race algorithm to find the optimal parameter setting. At each iteration a set of candidate configurations is generated according to a probabilistic model M. Balaprakash et al. (2007) consider a k-dimensional normal distribution and subsequently they assume that the parameters are independent (knowing a value for a particular parameter does not give any information on the values of the remaining ones), so that the k-dimensional normal distribution can be factorized as a product of k univariate independent normal

Algorithm 3: ParamILS (simplified)

Input : r random initializations:
s: perturbations
$p_{restart}$: probability of restarting the search
max_iter: Maximum number of iterations
$\mathcal{LS}()$: Local_Search() function

$i = 0$, c_0= initial parameter setting
for $k \in r$ **do**
  c=random parameter vector
  if $better(c, c_o)$ **then**
    $c_0 = c$
  end
end
$c_{\mathcal{ILS}} = Local\_Search()$
while $i < max\_iter$ **do**
  $c = c_{\mathcal{ILS}}$
  for $j = 1 \in s$ **do**
    c=Local_Search()
    %first improvement search
    if $better(c, c_{\mathcal{ILS}})$ **then**
      $c_{\mathcal{ILS}} = c$
    end
    if $r_{\mathcal{U}[0,1]} < p_{restart}$ **then**
      $c_{\mathcal{ILS}}$=random parameter vector
    end
  end
  $i = i + 1$
end
return *best parameter configuration* c^*

densities. Then, a standard F-Race is performed on this set: the survived candidates are reused to update the model and will be sent to the ensuing iteration. The idea here is to concentrate/spend the computational budget (which is also the stopping criterion) around a 'good' region by reusing survived candidates, in order to bias the search process towards better candidates at each iteration.

1.2.3.3 High level generate-evaluate methods

Huang et al. (2020) discuss a relatively new trend in tuning methods: this framework essentially is based on the generation and evaluation of elite (high-level) candidates, by means of which one could cut computational cost, i.e. by reducing the time spent to perform a thorough evaluation of candidates from the beginning. In other words, the idea is to quickly generate a set of high quality parameter configurations with a little amount of computational resources and then spending them in the evaluation process (i.e. the evaluation and selection process is performed among elite candidates). In this way, they argue, time could be saved in the parameter space exploration in favor of evaluation of elite configurations.

1. *Post-selection mechanism*

   The essence of the *post-selection* mechanism is summarized in Algorithm 4, which is a reworked version of the one proposed in the pioneering contribution of Yuan et al. (2013): at its core, the so-called post-selection mechanism serves as tuning method, which is organized in a two-phase process, namely an *elite qualification* phase and an *elite selection* phase. In the former, the elite configurations are collected by running one or more configurators simultaneously.

The configurators give back then a set of elite candidates; in the latter phase an evaluation method selects the best configuration c^* from the set of elite candidates C_e. Note that, as Huang et al. (2020) point out, this approach is entirely based on existing tuners, which both identify and select elite configurations.

Algorithm 4: Post-selection

Input : C: Set of candidate configuration
$\mathcal{E}_e$: Elite_evaluation_method()

Phase 1: elite qualification.
while $t <$ *budget* **do**
  Run the code with one or more configurators simultaneously, collecting the best configurations c_e;
  Store each c_e in C_e;
  Put aside a computational budget R_n depending on the number $n = |C_e|$ of elite configurations d for phase 2;
end
Go to phase 2;
Phase 2: elite selection.
Use the Elite_evaluation_method() (e.g. ParamILS, F-Race,...) to choose the best parameter configuration c^* from C_e.

1.3 Parameter control: a literature review and trends

Some insights of the parameter control problem have been provided in section 1.1; in particular, one of the taxonomies proposed by Eiben et al. (1999) has been discussed, focusing on three different approaches to parameter control, rather than analyzing what is changed. Indeed, a well-known classification identifies four strategies for parameter control, based on a set of questions:

- *What* is changed? (i.e. identify a list of parameters and then manage them, e.g. according to a schedule);
- *How* the change is made (identify a list of mechanisms, like adaptive/self-adaptive control,...);
- *Scope* of change (population level, individual level, sub-population level);
- *Evidence* that informs the change (e.g. by monitoring performance/diversity/...)

In this section we present a mechanism-specific literature review; in particular, we review concisely a set of parameter control techniques based on the previously mentioned classification (*deterministic parameter control*, *adaptive parameter control*, *self-adaptive parameter control*).

1.3.1 A formal definition of the parameter control problem

A parameter control problem addresses the problem of finding a (near)-optimal parameter configuration $c^*(t)$ in the space of configurations C, given an algorithm A, as the search proceeds, with respect to a particular stage of the process, say, time t. Given:

- An algorithm A, with $(p_{10}, \ldots, p_{n0}) \in P$ parameters, initialized at time $t = 0$ with suboptimal parameters;
- A set of problem instances $I = i_1, \ldots, i_m$;

- A feedback strategy $\mathcal{F}$, which could be explicitly based on a reward measure $\mathcal{R}(\cdot)$, estimated at each time t, with the aim of adapting the selection probability of each parameter value given by $\mathcal{P}(t)$ vector (regularly re-evaluted to take into account past and immediate effectiveness of each parameter value) or alternatively it could be inherently embedded in the procedure (i.e. a self adaptive one);

- A quality attribution strategy $\mathcal{Q}$ (for adaptive and deterministic control), i.e. a strategy which summarizes and stores the rewards of each value over time according to a predefined rule in a quality estimates vector $\hat{q}_t = [\hat{q}_{1t}, \ldots, \hat{q}_{nt}]$. For the various strategies implemented in literature, see Aleti and Moser (2016) or Fialho (2010);

- A selection strategy $\mathcal{S}$, which updates the parameter values to use at time $t+1$. Though this step is implicit for self-adaptive parameter control, adaptive strategies usually rely on reinforcement-like procedure, trying to strike a balance between exploration and immediate performance.

Find a configuration $c^*(t)$ for which A performs optimally at time t or generally at a specific stage of the search process on a set I, by setting different values of each parameter p_{it}, resulting in an effective, though temporarily, combination of parameter values. This definition is thus consistent with the deeply-rooted idea that different configurations may be optimal at different stages of the optimization problem.

1.3.2 Evaluating the parameter control algorithm

We find that currently, among many literature reviews available in this field, the most recent and comprehensive ones are those of Karafotias et al. (2015) and Aleti and Moser (2016); in particular the former provides a very broad analysis based on a parameter-specific approach, while the latter introduces in this field the so-called systematic literature review approach, which is unfortunately limited only to adaptive methods. In a nutshell, this procedure collects and assess systematically research in a certain field; typically the purpose of this investigation is to answer to a set of research questions (e.g. classify the current research activity (1), evaluate it in light of that classification (2) and determine a trend of research according to findings in points (1) and (2)).

Before diving in the analysis of parameter control mechanisms, we want to point out some of their features in this brief introduction. First of all, a well developed and abundant literature (Angeline (1995), Eiben et al. (1999), De Jong (2007), Eiben et al. (2007), As stated in the introduction, this approach presents many advantages (Karafotias et al. (2015)):

1. EAs are dynamic and adaptive algorithms, so using different parameter values at different stages of the search process is likely to provide better results (e.g. one can manage more easily the EvE balance, di Tollo et al. (2015));

2. It is a practical way to overcome the parameter tuning problem and, more generally, one does not have to set 'adequate' parameter values at all;

3. It allows EA itself to learn and store information on the ongoing process, so that it can be exploited for the following iterations, perhaps adjusting its behaviour;

4. The optimization process (De Jong (2007)) can evolve from a global one to a more focused/local converging one;

5. It allows the EA to deal with dynamic problems, i.e. adjusting a changing fitness landscape (Karafotias et al. (2015)).

Some authors (e.g. Eiben and Smith (2015)) point out that a further distinction is helpful for evaluating and classifying the performance of the algorithm. In particular, they consider the fourth question at the beginning of this section, i.e. the evidence that informs the change of parameter value. This criterion is crucial to monitor and dissect the performance of the algorithm; indeed, they note that the information used as feedback to adjust the parameters deserves a distinction between two potential cases:

- Absolute evidence, namely the rule used to change the parameter value is applied when a predefined event occurs, e.g. the mutation rate is increased deterministically or by feedback from search (e.g. when the population diversity hits a given threshold), i.e. the user has to design and to establish the desired direction of the given parameter;
- Relative evidence, i.e. parameter values are compared and the better configurations are rewarded. The direction of the change of the parameter is not specified in advance, rather it is relative to the performance of other values.

Table 1.2: A taxonomy of parameter control strategies, Eiben and Smith (2015)

	Deterministic	Adaptive	Self-adaptive
Absolute	✓	✓	✗
Relative	✗	✓	✓

This taxonomy, which is a refined version of the discussion provided at the beginning of this section, consists consequently of six combinations of strategies to implement properly a parameter control mechanism, though two of them are practically impossible, as shown in table 1.2.

1.3.3 Parameter control methods

In this section, we discuss briefly some parameter-specific control methods; given the breadth of this topic, the focus of this investigation is not to provide a thorough and complete analysis of existing methods, rather we want to look at some methods that we find meaningful to motivate a further general discussion on the most relevant problems and trends in this field. Our review is based on three 'subpillars', i.e. a subclassification of parameter control techniques: we will indeed explore the state-of-the-art strategies with a parameter specific approach. We will investigate particularly a few contributions on control designed for the following parameters:

- Fitness function (constrained problems management via penalty methods);
- Population;
- Variation operators (operator selection and probability setting)

Note that deterministic methods have enjoyed a decent popularity in the early days of research in this field; recently much more effort has been devoted to develop adaptive or self-adaptive methods. Fialho (2010) argues that there are basically two main reasons to prefer adaptive/self-adaptive methods:

- A deterministic parameter control technique implements a schedule without any feedback from the search process, requiring the user to guess in advance how long the algorithm takes to achieve a target solution, which is for sure a challenging quest;
- Furthermore, there is no question that the EvE balance is not manageable by deterministic approaches; rather, a feedback-based strategy should be preferable, as it can monitor and manage many different properties of the search process.

1.3.3.1 Deterministic parameter control

1. *Fitness function: managing constrained problems*

   In this field, the two seminal contribution of Joines and Houck (1994) and Michalewicz and Attia (1994) are for sure the most relevant; in particular, the latter introduces the groundbreaking *Genocop II*, an hybrid optimization system (GA+dynamic penalty method) to handle non-linear constrained programming problems.

   Let us consider a constrained optimization problem, with m inequality constraints and p equality constraints. The problem can be written as follows:

$$\begin{aligned} &\min \quad f(x) \\ &\text{s.t.} \\ &g_i(x) \leq 0 \quad \text{for} \quad i \in [1, m] \\ &z_j(x) = 0 \quad \text{for} \quad j \in [1, p] \end{aligned} \tag{1.3}$$

   Consider two arbitrary sets, containing *feasible* and *infeasible* solutions:

$$\begin{aligned} \text{feasible set} \quad & \mathcal{F} = \{x : g_i(x) \leq 0 \quad \text{for } i \in [1, m] \text{ and } z_j(x) = 0 \text{ for } j \in [1, p]\} \\ \text{infeasible set} \quad & \mathcal{F} = \{x : x \notin \mathcal{F}\} \end{aligned} \tag{1.4}$$

   A penalty approach modifies the fitness function of an individual x according to some constraint violation measure; if one of the constraints in 1.4 is not respected, these unfeasible solution $x_{\notin}$ are allowed but their fitness amount is subjected to penalty, as follows (we call this an exterior point approach, see Simon (2013)):

$$\begin{aligned} &\min \quad \phi(x) = f(x) + \sum_{i=1}^{m} r_i G_i(x) + \sum_{j=1}^{p} c_j Z_j(x) \\ &\text{s.t.} \\ &G_i(x) = [\max(0, g_i(x))]^{\beta} \\ &Z_j(x) = [\max(0, |z_j(x)| - \epsilon]^{\beta} \end{aligned} \tag{1.5}$$

   Typically the equality constraints are unforgiving (Simon (2013)), so they are reformulated as above, with the inclusion of a parameter ϵ, which should be properly calibrated to get meaningful results (i.e. the equality constraint $z_j(x) = 0$ is rearranged to get $|z_j(x)| \leq \epsilon$). Note that for every $x \in \mathcal{F}$ we have $\phi(x) = f(x)$ and if $x \notin \mathcal{F}$ we have $\phi(x) > f(x)$ due to constraint violation. Joines and Houck (1994) propose a dynamic penalty method, with $\beta = 1$ or 2 and $r_i{=}c_j{=}(ct)^{\alpha}$, with t the $t-th$ generation, so that the penalty amount increases as times goes by, with $c = 0.5$ and $alpha = 1$ or 2.

   Rearranging the minimization problem 1.5 we obtain:

$$\begin{aligned} \phi(x) &= f(x) + (ct)^{\alpha} M(x) \\ M(x) &= \sum_{i=1}^{m+p} G_i(x) \end{aligned} \tag{1.6}$$

   Simon (2013) proposes to adjust the penalized fitness function by normalization:

$$\begin{aligned} &\phi(x) = f'(x) + (ct)^{\alpha} M'(x) \\ &\begin{cases} M'(x) = M(x)/max_x M(x) \\ f'(x) = f(x)/\max|f(x)| \end{cases} \end{aligned} \tag{1.7}$$

Note that the value discussed in the literature are often the result of tuning by analogy or by hand; generally, the value of c should be consistent with the number of generations (i.e. if it is low, one may also consider to use $c = 1$ or $c = 1.5$).

Genocop II (Michalewicz and Attia (1994)) uses annealing penalties, i.e. the penalty coefficients are based on a 'freezing schedule' $\frac{1}{2\tau(t)}$, which in turn depends on the generation count and on the temperature τ itself.

Kazarlis and Petridis (1998) adjust the penalty factor dynamically in a so-called *varying fitness function* framework. The gist of this approach is to replace the term ct with a a function $V(t)$, which again depends on the generations count; moreover, the authors identify seven shapes for the function $V(g)$ (e.g. linear, exponential, cubic,...).

2. *Population*

The literature available on this topic is quite broad, so here we discuss only a few interesting approaches for managing the population parameter. Costa et al. (1999) discuss growth and shrink schemes in what they call *Random Variation of Population Size* (RVPS), in order to vary the population diversity and the selective pressure as well. The underlying idea is to increase diversity by including randomly generated individuals in the population and at the same time to increase (reduce) selection pressure by replicating the worst (best) individuals.

Shrink schemes can decrease the diversity of the population (i.e. by removing individuals), but they can vary the selection pressure as much as growth ones do. In their experiments, they authors propose to investigate on a limited set of test functions the potential of deterministic control schedule (RVPS), by analyzing its impact on some performance measures. Eiben and Schut (2008) find out that this method performs poorly on a set of different performance metrics (Success Rate (SR), Mean Best Fitness (MBF), Average number of Evaluations on Success (AES)) when compared to other methods.

Algorithm 5: Random Variation of Population Size (RVPS)

Input : $\mathcal{T}$: number of deterministic resizing decisions
n: number of individuals
$\mathcal{I}$: number of iterations for each resizing decision

Generate the initial population P_0
Evaluate the fitness of P_0
for $t = 1 : \mathcal{T}$ **do**
  for $i = 1 : \mathcal{I}$ **do**
    $Parents_i = Selection(n, P_i)$
    $Offspring_i = Crossover(Parents_i)$
    $Offspring_i = Mutation(Offspring_i)$
    $P_i = Offspring_i$
    $Pop\_Fitness_i = Evaluate(P_i)$
  end
  %Resize population according to a deterministic rule (e.g. linear)
  $Resize\_Population(P_{\mathcal{I}}, t)$
end
Evaluate quality of the population $P_{\mathcal{T}}$ after $\mathcal{T}$ runs (e.g. with MBF)

This approach, which is probably both the most straightforward and easy-to implement among populations control methods, according to the taxonomy proposed by Karafotias et al. (2015), belongs to those mechanisms which control directly the parameter on-the-fly (in this case, the population is shrinked or

grows in a way that is consistent with a generation-based schedule); for sure, in the collection of deterministic methods, some other approaches are viable.

Koumousis and Katsaras (2006) propose a genetic algorithm which employs a more sophisticated scheme, based on a variable population size and a partial reinitialization of the population; the insertion of randomly generated individuals makes the population variation scheme peculiarly 'saw-toothed': the authors argue that this method is expected to improve some performance metrics (i.e. maximum/mean fitness). Suppose to have a mean population size $\bar{n}$, which corresponds to the fixed population size GA having the same computing cost (Koumousis and Katsaras (2006)) and consider an amplitude parameter D, which should be tuned accurately (see below). Furthermore, take into account a period of variation T: then, at generation t the population size $n(t)$ is:

$$n(t) = int\left\{\bar{n} + D - \frac{2D}{T-1}\left[t - Tint\left(\frac{t-1}{T}\right) - 1\right]\right\} \tag{1.8}$$

A few observations are worthy of remark: first of all, note that $n(0) = \bar{n} + D$, $n(T) = \bar{n} - D$ and $n(T+1) = \bar{n} + D$. Moreover, note that the performance of the schedule presented here is heavily affected by parameter selection, so again the parameter tuning problem arises. For $D = 0$ we have a constant population size GA, whichever the value of T, whereas for every D greater than zero, the population size decreases with a constant decay (of course, the bigger the value of D, the stronger the impact).

Finally, note that a graphical interpretation of T and D could be given: indeed, at an arbitrary generation t, T represents roughly the amount of 'teeth' for a given number of generations t on the x-axis (i.e. number of random reinitialization) and D represent somehow the 'size' of each 'tooth', i.e. in plain terms the random restarts make the population size $n(t)$ in the range $[0, t]$ more variable, while the restart frequency depends on the user-defined value of T.

Laredo et al. (2009) develop a three-step schedule called *Simple Variable Population Sizing* (SVPS) to determine the optimal amount of individuals at each generation. This method consists in decreasing the population according to a deterministic schedule, based on speed and severity parameters. The steps can be summarized as follows:

(a) A bisection method estimates the optimal size n^* of the initial population P_{init};

(b) A refinement procedure is enforced to improve the optimal size n^*;

(c) The SVPS-GA uses the following deterministic function to alter the population size:

$$n(t) = \begin{cases} n(0) \cdot \left(1 - (1-\rho)(\frac{t}{t_{max}})^{\tau}\right) & t \leq t_{max} \\ n(t_{max}) & t > t_{max} \end{cases}$$

With $n(t)$ denoting the population size at generation t, $n(t_{max})$ the population size when a maximum number of generations of the schedule is reached, estimated on the runtime of constant population size GA. τ and ρ represent, respectively, the speed and severity parameter; the greater the value of ρ the faster the reduction of the population size, $\rho \in [0, 1]$ where $\rho \to 0$ implies that the run ends with an almost empty population, while for $\rho \to 1$ the initial population size is not modified.

3. *Operator selection*

As argued by Fialho (2010), the deterministic schedule approach is not very suitable to manage the operator selection, as it can not properly take into account,

for instance, the EvE balance, which is hardly manageable deterministically, i.e. without a feedback from the process.

As a consequence, in this subsection we review briefly a few approaches that are somehow outdated and not discussed in the most recent experimental studies.

Davis (1989) has for sure paved the way, in general, to operator selection strategies, whichever the technique adopted. In particular, he proposes to update the probability of selecting operators via a fitness-based decay mechanism, which is in charge of the credit assignment process, up to a pre-established number of generations; the method does not include a feedback mechanism, so it has been used to obtain a deterministic and time-varying schedule.

Hatta et al. (1997) discuss a method in which the crossover operator is chosen according to a measure of fitness, i.e. the so called *elite degree* of an individual, which is basically the ratio of the sum of all its 'elite ancestors' (up to some predefined number of generations before) and the total number of ancestors. An individual is an elite member if her/his fitness is greater than an amount defined by $\bar{n} + \alpha\sigma$, with $\bar{n}$ the average fitness of the current population, σ its standard deviation and α is a parameter that magnifies the impact of the standard deviation, according to the user's preference.

Once the credit assignment scheme has fully determined a set of credits for each operator, then the operator selection is based on a deterministic and feedback-free schedule: if the sum of the elite degrees of both parents is higher than a fixed threshold, then less disruptive operators are chosen (Fialho (2010)).

Tuson and Ross (1998) have devised an approach for which the user (before carrying out the experiments) defines a set of static probabilities, which are subsequently and deterministically assigned to the operators, according to a ranking based on a dynamic credit assignment scheme. The authors do not disclose how to define *a priori* the set of probabilities, which serves as input for the operator selection.

1.3.3.2 Adaptive parameter control

1. *Fitness function: managing constrained problems adaptively*

   In subsection 1.3.3.1 we have stressed the fact that deterministic methods are based on a predefined schedule, so they often require a significant amount of parameter tuning, at least to get meaningful results; furthermore, managing the penalty coefficient correctly is far from easy, because it in general leads to complex trade-offs (i.e. too high values prevent the algorithm from searching infeasible areas, whereas too low values lead to slow/poor convergence towards 'good' results).

   To overcome this problem, which is actually common and not specifically related to penalty methods, some solutions have been proposed.

   Ben Hadj-Alouane and Bean (1997) propose an adaptive method, i.e. a feedback from the population is used stepwise to adjust the penalty weights of equation 1.5 as follows:

$$c_j(t+1) = r_i(t+1) = \begin{cases} r_i(t)/\beta_1 & \text{if } x_{best} \in \mathcal{F} \text{ in a } k \text{ rolling window} \\ \beta_2 r_i(t) & \text{if } x_{best} \notin \mathcal{F} \text{ in a } k \text{ rolling window} \end{cases}$$

   Where t identifies the generation count, k is a rolling window to be tuned affecting the speed of adaptation. Note that the gist of this method is to decrease (increase) the penalty weight if the best individual is (not) feasible, allowing more infeasible individuals in the population. This method, differently from those shown above, allows the algorithm to use a feedback from the search

process, so that it is adjusted accordingly in the following generations, i.e. by tightening/loosening the penalty weights.

A segregated genetic algorithm is proposed in Le Riche et al. (1995); the idea behind this is method is to use two values of the penalty parameters r_i and c_j, which are associated with two populations, performing at their best with respect to their own penalty settings. These two groups are segregated in terms of rankings, but at each iteration they are mated, in order to get a solution which is deemed to be more robust, as new individuals are generated both from the individuals in the group with a tight penalty parameter (say, group 1) and from those in the group with a loose (group 2) one, so that one obtains an auxiliary population arising from a mixture of feasible (group 1) and infeasible individuals (group 2).

2. *Population*

Adaptive population sizing is for sure one of the research areas that has attracted most interest from scholars. For a thorough and recent review of contributions in this field, we refer the reader again to Karafotias et al. (2015); they propose a new subclassification of population sizing, which takes into account the most recent developments as well:

- Theoretical studies on population size and benefits of dynamic sizing;
- Approaches which manage the population size parameter by exploiting a new operator;
- Mechanisms that approximative a good population size during the run;
- Methods that control directly parameters on-the-fly.

Here we limit ourselves to consider only the most relevant strategies; we start from the Genetic Algorithm with Varying Population Size (GaVaPS) (Arabas et al. (1994)); the authors actually remove the population size as a parameter. In this review direct approximation of population size will be also considered, as it is deemed as a wholly different family of strategiies.

This strategy removes the population size as a parameter as follows. The strategy allocates to each individual at his/her birth (i.e. generated either in the initial population or by means of a variation operator) a maximum lifetime which depends both on her/his individual fitness and her/his average fitness. At each step, one year is added to an age counter; hence, when the maximum lifetime is reached, the individual is removed from the population. Furthermore, a ρ parameter, called *reproduction ratio*, keeps the offsprings/population size ratio constant, so that also the selection pressure is kept constant, namely only a fraction of the current population is allowed to the mating phase. Note that for different values of ρ, the population may grow rapidly or may become extinct.

The core implication of this design is that the population size is not properly a parameter and there is not an explicit selection operator, as the individuals are chosen to reproduce with equal probability, so selection does not depend on fitness value. However, given that the individuals die once they exceed their lifetime value, on long term and on average, those who are fitter live longer (i.e. a further indirect selection is performed). The idea is obviously to get a fitness-driven selection and meanwhile to add some diversity to the population. Overall, the selection pressure does not vary.

Arabas et al. (1994) identify a set of strategies that could be implemented (proportional, linear and bilinear) to get different selection pressure levels, relying on two parameters, $MinLT$ and $MaxLT$, i.e. minimum and maximum lifetime value allowed. Higher difference between the two values leads to higher values of

Algorithm 6: GAVaPS

Input : P_0: Initial population size
ρ: Reproduction ratio
$\mathcal{T}$: stopping criterion

```
t = 0, P(0) = P_0
Evaluate P(0)
while t < T do
    increase age of each individual i
    recombine P(t) with a consistent ρ
    evaluate P(t)
    if age_i < lifetime_i then
        remove individual i from P(t)
    end
    t = t + 1
end
```

selection pressure; the lifetime parameter for the $i-th$ individual is computed as follows:

- proportional allocation:
  $\min(MinLT + \eta\frac{Fitness[i]}{AvgFit}, MaxLT)$
- linear allocation:
  $MinLT + 2\eta\frac{Fitness[i] - |FitMin|}{|FitMax| - |FitMin|}$
- bilinear allocation:

$$\begin{cases} MinLT + \eta\frac{Fitness[i] - MinFit}{AvgFit - MinFit} & \text{if} \quad AvgFit > Fitness[i] \\ 0.5(MinLT + MaxLT) + \eta\frac{Fitness[i] - AvgFit}{MaxFit - AvgFit} & \text{if} \quad AvgFit < Fitness[i] \end{cases}$$

where $\eta = (MaxLT - MinLT)/2$, $MinFit$, $MaxFit$ and $AvgFit$ stand for the minimum, maximum and average fitness of the population. Though the user has to input the initial population size, the authors claim that the initial population has little influence on the performance. However, a study of Eiben et al. (2004) finds out that GAVaPS is highly sensitive to the rate ρ (which has to be inputted by the user too), leading either to population extinction or to exponential growth.

A slight variation of GAVaPS is presented in Bäck et al. (2000), called Genetic Algorithm with Adaptive Population Size (APGA). The difference involves basically the reproduction rate ρ, which admits for each iteration only two new offsprings, a results which stems from the experimental results in their work. Essentially, the idea is that this value should prevent the reproduction strategy from growing out of control; their findings lead to a very stable population, though only with a few individuals included (unless a high value of $MaxLT$ is set) and not very responsive to the evolution of the search process.

Eiben et al. (2004) present an approach known as Population Resizing on Fitness Improvement GA (PRoFIGA), which is very similar to a traditional GA, but at the end of selection, reproduction and mutation steps, the population size can grow or shrink on the ground of improvements of the fittest individual contained in the population. The population grows if there is an improvement in the fittest individual or if there is not an improvement in the fittest for many iterations; otherwise the population is shrinked by a user-defined percentage (usually a small amount).

It is easily noticeable that behind this design there is an attempt to keep the EvE balance under control, as this strategy typically leads to large populations during the exploration phase and to small populations in the exploitation phase.

Overall, this method introduces six parameters, which require non-trivial tuning; though the authors suggest some values, they are mainly problem-specific:

(a) initial population size;

(b) increaseFactor (a number in the interval $[0, 1]$);

(c) V, number of generations without improvement;

(d) decreaseFactor (a number in the interval $[0, 1]$);

(e) minPopSize;

(f) maxPopSize.

The growth rate X is determined as follows:

$$X = increaseFactor \cdot (maxEvalNum - currEvalNum) \cdot \frac{maxFit_{new} - maxFit_{old}}{initMaxFit} \tag{1.9}$$

3. *Adaptive Operator Selection*

Adaptive operator selection (AOS) copes with the problem of choosing an optimal symbolic parameter (see 1.1) of the variation operator. An encompassing discussion is proposed in Maturana et al. (2010), in which the problem is tackled with a very general framework which allows a lot of implementation flexibility. In a nutshell, they propose a framework in which four modules are embedded in a *controller*, as follows. First, a module stores the successive application of the operators with a k-sized memory; then, the following module computes a reward for each operator, according to a strategy (e.g. fitness-based reward, diversity-based reward,...). Hence, the impact is passed to a credit assignment mechanism, which computes a score for each operator, outputted finally to an operator selection module, whose goal is to choose the operator to be used in the following generation according to a pre-specified rule (e.g. Probability Matching, Adaptive Pursuit, UCB,...).

As far as operator selection is concerned, the simplest strategy is the so-called Probability Matching (PM) strategy, which chooses the optimal operator at generation t according a to roulette wheel selection strategy, i.e. the probability $\mathcal{P}_i$ of choosing an operator i is proportional to its credit $\mathcal{U}_i$ to the total amount of credits of all operators K. Note that a minimum probability of selection P_{min} is typically enforced, in order to retain all those operators which could turn out to be useful in later stages of the search process; due to the non-stationarity of the environment considered, i.e. the probability distribution that specifies the reward generated by operators is unstable (Thierens (2007)).

Formally we have the following rule to update the probabilities:

$$\mathcal{P}_i(t+1) = P_{min} + (1 - K \cdot P_{min}) \cdot \frac{\mathcal{U}_i(t+1)}{\sum_{i=1}^{K} \mathcal{U}_i(t+1)} \tag{1.10}$$

This approach typically ends up choosing suboptimal operators; an alternative strategy is proposed by Thierens (2007), the so called Adaptive Pursuit (AP), a winners-take-all strategy, which proportionally selects an operator to execute according to a probability vector $\mathcal{P}(t)$; therefore, the selection probability of the operator with maximal reward i_t^* is increased and all other probabilities are decreased. The main idea is to adapt the probability vector $\mathcal{P}(t)$ in a greedy way:

Algorithm 7: Probability Matching (PM)

Input : $\mathcal{T}$: termination criterion
$\mathcal{A} = \{a_1, \ldots, a_K\}$: set of K operators
P_{min}: minimal value of selection probability for each K
α: adaptation rate

for $i = 1 : K$ **do**
  % initial probability vector
  % initial credit vector
  $\mathcal{P}[i] = 1/K$
  $\mathcal{U}[i] = 1.0$
end
while *termination criterion* $\mathcal{T}$ *not met* **do**
  $s = Select\_Operator(\mathcal{P})$
  $R_s[t] = Get\_Reward(s)$
  $\mathcal{U}_s[t+1] = \mathcal{U}_s[t] + \alpha(R_s[t] - \mathcal{U}_s[t])$
  %update probability vector
  for $s = 1 : K$ **do**
    $\mathcal{P}_s[t+1] = P_{min} + (1 - K \cdot P_{min}) \dfrac{\mathcal{U}_s[t+1]}{\sum_{i=1}^{K} \mathcal{U}_i[t+1]}$
  end
end

$$\begin{cases} i_t^* = \arg\max[\mathcal{P}_i(t), i = 1, \ldots, K] \\ \mathcal{P}_i^*(t+1) = \mathcal{P}_i^*(t) + \beta(P_{max} - \mathcal{P}_i^*(t)) \\ \mathcal{P}_i(t+1) = \mathcal{P}_i(t) + \beta(P_{min} - \mathcal{P}_i(t)) \end{cases}$$

A third selection method, proposed by Costa et al. (2008), is inspired from the Multi-Armed Bandit problem (MAB), so that the operator selection problem is expressed as an EvE dilemma. In particular, the adopted approach (actually, many options are available) is to employ an Upper Confidence Bound (UCB1 variant) algorithm. Each variation operator is treated as an arm of a MAB problem; then denote with $n_{i,t}$ the number of times the $i-th$ arm has been played up to generation t and with $p_{i,t}$ the average corresponding reward. With C we denote the scaling factor. At each time step t the algorithm chooses the arm maximizing:

$$p_{i,t} + C\sqrt{\frac{log \sum_k n_{k,t}}{n_{j,t}}} \tag{1.11}$$

The first term tilts the search process towards exploitation (by favoring individuals with a greater average reward up to time t) while the second one, often interpreted as a 'variance' term, enforces exploration.

They also suggest to use a dynamic version of the algorithm: indeed, they argue, if an operator becomes less efficient during the run, the probabilities are adjusted quite slowly. As a consequence, the authors adopt a *Page-Hinkley* test, coupled with the UCB algorithm. As soon as the test detects a change in the distribution, the MAB algorithm is restarted, in order to overcome the slow convergence of MAB. They refer to this method as Ex-DMAB, i.e. a dynamic MAB algorithm with the AOS combination.

For a complete discussion of these topics, which include also the setting problem of the credit assignment mechanism, we refer the reader to the contribution of Maturana et al. (2010).

1.3.3.3 Self-adaptive parameter control

1. *Fitness function: self-adaptive penalty methods*

   In this section we discuss two self-adaptive penalty methods: given that the approaches discussed are quite sophisticated, here we limit ourselves to outline the basic steps.

   Farmani and Wright (2003) discuss a self-adaptive penalization method in two steps. If any individual $x \notin \mathcal{F}$ has an unpenalized fitness value that is better than the best individual $x \in \mathcal{F}$, then the fitness value of each individual $x \notin \mathcal{F}$ is penalized. Furthermore, if $\hat{x}$=$\{x : x \notin \mathcal{F}\}$, $\tilde{x}$=$\{x : x \in \mathcal{F}\}$ and we have that $f(\hat{x}) > f(\tilde{x})$ for all $\hat{x}$ and for the best value in $\tilde{x}$, then none of the infeasible individuals are penalized. Let us define a total infeasibility measure for each individual x as follows (see equation 1.5):

   $$\tau(x) = \frac{1}{m+p} \sum_{i=1}^{m+p} G_i(x) / \max_{x \in \hat{x}} G_i(x) \tag{1.12}$$

   Then we denote with x_b the best individual, with x_{wf} the worst in terms of feasibility (i.e. an individual such that $\tau(x)$ is maximized) and with x_{wc} the worst in terms of fitness. Moreover, the infeasibility metric is normalized:

   $$\tilde{\tau}(x) = \frac{\tau(x) - \tau(x_b)}{\tau(x_{wf}) - \tau(x_b)} \tag{1.13}$$

   The first penalized step is defined as follows:

   $$\phi(x) = \begin{cases} f(x) + \tilde{\tau}(x)(f(x_b) - f(x_{wf})) & \text{if } \exists x \in \hat{x} \text{ such that } f(x) < f(x_b) \\ f(x) & \text{otherwise} \end{cases}$$

   A second step of penalization is then introduced. All the infeasible individuals are penalized and in particular the worst penalized fitness is assigned to the individual with the greatest violation. First of all, we define an auxiliary exponential penalized fitness function:

   $$\phi'(x) = \phi(x) + \gamma|\phi(x)|\left(\frac{exp(2\tilde{\tau}(x)) - 1}{exp(2) - 1}\right) \tag{1.14}$$

   $$\gamma = \begin{cases} (f(x_{wc}) - f(x_b))/f(x_b) & \text{if} \quad f(x_{wf}) < f(x_b) \\ 0 & \text{if} \quad f(x_{wf}) = f(x_b) \\ (f(x_{wc}) - f(x_{wf}))/f(x_{wf}) & \text{if} \quad f(x_{wf}) > f(x_b) \end{cases}$$

   The scaling factor γ makes sure that $\phi'(x_{wf}) \geq \phi'(x)$ for all x.

   A different approach is discussed in Tessema and Yen (2006), where a five step procedure is presented:

   (a) Normalize the fitness function for each x:

   $$f'(x) = \frac{f(x) - \min_x f(x)}{\max_x f(x) - \min_x f(x)} \tag{1.15}$$

   The normalized fitness is then $f'(x) \in [0, 1]$ fo all x.

   (b) Compute $\tau(x)$ which is a normalized measure of constraint violation, so $\tau(x) \in [0, 1]$ for all x, so that if $x \in \mathcal{F}$, then $\tau(x) = 0$ and if $x \in \hat{x}$ then $\tau(x) > 0$.

(c) Compute the distance value for each x:

$$d(x) = \begin{cases} \tau(x) & \text{if } \mathcal{F} = \emptyset \\ \sqrt{f'^2(x) + \tau^2(x)} & \text{if } \mathcal{F} \neq \emptyset \end{cases}$$

The idea here is to compute a distance value that is equal to the constraint violation of x if there are not any feasible individuals in the population, whichever the fitness value; otherwise, if there are feasible individuals, then the distance value is computed as a combination of fitness and constraint violation.

(d) Compute the following auxiliary fitness functions:

$$X(x) = \begin{cases} 0 & \text{if } \mathcal{F} = \emptyset \\ \tau(x) & \text{if } \mathcal{F} \neq \emptyset \end{cases}$$

$$Y(x) = \begin{cases} 0 & \text{if } x \in \mathcal{F} \\ f'(x) & \text{if } x \notin \mathcal{F} \end{cases}$$

(e) Compute the final penalized cost function:

$$\phi(x) = d(x) + (1 - r)X(x) + rY(x) \tag{1.16}$$

where $r \in [0, 1]$ is the ratio of feasible individuals to population size. Note that $\phi(x)$ strikes a balance between the relevance of the auxiliary fitness function $X(x)$ and $Y(x)$, in such a way that if there are many feasible individuals, the latter is emphasized instead of $X(x)$. Note that $Y(x)$ contains fitness-based penalties on infeasible individuals, whereas $X(x)$ includes constraint violation penalties on infeasible individuals.

Note that this approach does not require to determine the penalty coefficient; basically, the algorithm looks for a good mix between feasible and infeasible individuals, so that infeasible individuals are exploited in the search process. To put it simply, the distance value is the metrics which guides the whole process, as it helps to compare infeasible and feasible individuals, according to a rigorous metrics, which takes into account a proper balance of fitness and constraint violation in the $(\tau(x), f'(x))$ plane, according to which -potentially- infeasible individuals are allowed to be have a better (penalized) fitness value, due to a lower distance.

2. *Population*

Controlling the population size by self-adaptation is a relatively new trend in parameter control: as Eiben et al. (2006a) highlight, the traditional literature on self-adaptation has flourished in variation operators management, i.e. self-adapting mutation and recombination, while a relatively little focus has been put on what they call 'global parameters'.

They find evidence that handling the population size with a self-adapting procedure can be rewarding, given two well-known benefits arising from self adaptation, namely the size is adapted in a evolutionary and smooth way (not heuristically) (1) and broad evidence in the literature of adequate parameter control in evolution strategies (2). As a consequence, they concentrate their efforts on carrying out some experimental studies on population with simple strategies, as the one described below.

The idea is to derive the population size via an aggregation method, so that it is determined from local parameters, i.e. collectively by the individuals in the population. They propose the following procedure:

(a) An extra parameter $p \in [p_{min}, p_{max}]$ is assigned to each individual, which is interpreted by the authors as an individual 'vote' for a collective decision regarding the global parameter P;

(b) An aggregation mechanism calculating P from individual p values is chosen. The authors opt for the sum of the local votes:

$$P = \lceil \sum_{i=1}^{N} p_i \rceil \tag{1.17}$$

with $p_i \in [p_{min}, p_{max}]$ and N the population size;

(c) The extra parameter p is added to each individual chromosome, i.e. it is encoded as an extra gene. As we explain below, this design supports self-adaptation itself, because the 'local' parameters are forced to undergo mutation/variation just like the other parameters, so that the user does not have to come up with a specific heuristic to control the parameter size.

In order to obtain meaningful values of p, they propose the following mutation mechanism:

$$p' = \left(1 + \frac{1-p}{p} \cdot e^{-\gamma \cdot N(0,1)}\right)^{-1} \tag{1.18}$$

With γ we denote the learning speed, i.e. it controls the adaptation speed. Note that for $p \in (0, 1)$, then $p' \in (0, 1)$.

Another approach is the one of Baluja and Caruana (1995), called Population-based incremental learning (PBIL) which removes the population size parameter rather than focusing on population size approximation. Basically, they employ a probability vector over each individual chromosome to represent the population. In particular, they use a binary encoding strategy to represent each solution; then, they also store the proportion of ones and zeroes at each gene position (which they initialize to 0.5) and move away from the initial solution, towards 0 or 1 as the search progresses. Then, at each step they update the probability vector with a simple rule, based on a learning rate:

$$p_{i+1} = p_i \cdot (1 - LR) + individual_{ij} \cdot LR \tag{1.19}$$

The idea is to reuse this updated probability to generate new solutions, by tilting the probability vector toward the fittest of the generated solution. Though we do not here elaborate too much on PBIL, we note that this mechanism is implemented in such a way that the number of individuals to update from nv and the stopping criterion are all parameters of the algorithm, i.e. self-adapted.

Harik et al. (1999) propose a variation of the procedure described in Baluja and Caruana (1995) with the so-called 'Compact Genetic Algorithm': basically they modify the generation step, by generating only two individuals each time and then picking a winner according to the result of a competition. Furthermore, they opt for an update step with a constant rule, with the aim of converging to a solution, reducing at the same time the computational burden.

3. *Self-adaptive operator selection*

The issue of selecting an optimal operator at each step has been discussed for a long time and, as we have noted before, although adaptive operator selection research has experienced in the last decade a steady growth, there also a few contributions in the self-adaptive field which are worth mentioning.

A seminal contribution here is the one of Spears (1995), in which a simple operator selection is implemented. First, a bit representation is adopted, so the individual amount of fitness is encoded in a bit string. Hence, a bit is

Algorithm 8: Population-based incremental learning (PBIL)

Input : p: Initial probability vector
P_0: Initial population with l elements for each solution
$\mathcal{LR}$: Learning Rate
nv: subset of n individuals used in the update step

for $i = 1 : l$ **do**
  % initial probability vector
  $p[i] = 0.5$
end
while *termination criterion* $\mathcal{T}$ *not met* **do**
  for $i = 1 : n$ **do**
    %generate n individuals with consistent p vector
    $individual[i] = generate(p)$
  end
  for $i = 1 : n$ **do**
    $evaluate(individual[i])$
  end
  rank individuals according to fitness
  %update probability vector
  for $i = 1 : nv$ **do**
    for $j = 1 : l$ **do**
      $p[i] = p[i] \cdot (1 - \mathcal{LR}) + individual[j][i] \cdot \mathcal{LR}$
    end
  end
end

added to each chromosome, which serves as an indicator bit, i.e. choice among two crossover operators; in particular, they author implements the operator selection among two crossover methods, i.e. two points crossover and uniform crossover. Though a bit outdated, these operators show different properties, namely the latter is considered highly disruptive while the former is deemed less disruptive of material, so that a 'mixture' of these characteristics may lead to more 'balanced' individuals.

As a consequence, each individual receives an extra bit, so that the last column of the population represents the space of operators (e.g. 0 stands for uniform crossover and 1 for two-point crossover). Thus, the last column is manipulated by crossover and mutation operators just like the whole string of bits. Many specific implementations are then allowable: once the last columns is generated randomly, then local or global adaptation can be adopted. If ther former approach is chosen, then the choice of the crossover operator is tied to a particular individual, whereas if the latter strategy is picked, the crossover operator is linked to the population results (i.e. the last column can be used in a roulette-wheel fashion, in order to choose each time which crossover technique should be performed).

Montero and Riff (2011) discuss two self-adaptive control techniques, which we briefly review here. The key idea is that the representation of each individuals includes the parameter value of genetic operators, namely the operators probability value; basically each operator receive a reward when its application generates a better offspring than his or her parents; otherwise she/he receives a penalty. We formally summarize the procedure, which they call *light-self adaptive control* (LSA) and *Self-Adaptive Control* (SA) as follows. We denote with S_a a success measure for the operator O_k in its a-th application, $S_a(O_k)$:

$$S_a(O_k) = F(C_j) - F(P) \tag{1.20}$$

Where $F(C_j)$ and $F(P)$ denote respectively the fitness of the child and the average fitness of her/his parents. Then we denote with $Max - i_l$ and with $Max - d_l$, respectively, the 'positive behaviour' of an operator and the 'negative behaviour' of an operator during the l generations, for a given set of operators O_k and for a success measure S_a:

$$Max - i_l = \arg \max_{a=1,\ldots,A_k} (S_a(O_k)) \tag{1.21}$$

where A_k is the number of applications of the operator O_k in the last l generations.

$$Max - d_l = \arg \max_{a=1,\ldots,A_k} (|S_a(O_k)|) \tag{1.22}$$

Finally, given the a-th application of an operator, we define the probability of selecting as follows:

$$P_{c_j}(O_k) = \begin{cases} 1 + \delta \cdot P_h(O_k) & \text{for reward } LSA \\ 1 - \delta \cdot P_h(O_k) & \text{for penalty } LSA \\ (1 + \dfrac{S_a(O_k)}{Max_l} \cdot P_h(O_k) & \text{for } SA \end{cases}$$

with δ a random number in (0,1), P_h the parent probability and:

$$Max_l = \begin{cases} Max - i_l & \text{if } S_a(O_k) \geq 0 \\ Max - d_l & \text{otherwise} \end{cases}$$

Note that both LSA and SA determine the probability of choosing an operator for a child C_j according to a reward/penalty mechanism, but the only the latter takes into account an 'absolute' amount of improvement/degradation, depending on the successive application of the operator O_k. The key idea is that LSA detects only an improvement or a deterioration of the fitness value, and subsequently penalizes the operator probability; the SA technique is instead more sophisticated, as it employs a measure of reward/penalty which is proportional to the quality of the generated child compared to the fitness of her/his parents.

1.3.3.4 Some features of parameter independent methods

We conclude this section on parameter control by taking a look to some recent developments in parameter independent methods, i.e. generic control methods that suit any numeric EA parameter. We provide a qualitative discussion, without elaboring too much on specific implementations; we outline this topic by mentioning a few influential works.

To our knowledge, a seminal contribution in this field is the one of Eiben et al. (2006b), in which they propose to use the feedback from the search process, performing on-the-fly adjustement of the parameter values, just like self-adaptation; the innovation in their work, which actually has opened a wholly new area of research, is the application of reinforcement learning techniques to calibrate EAs, with a combination of SARSA (i.e. *on-policy reinforcement learning*) and Q-Learning (*off-policy reinforcement learning*) techniques. The state of the EA-process is described by a vector of numbers which reflects the characteristics of the population, then some actions are undertaken in order to maximize a reward measure; the fundamental idea in this case is to learn the functions which map states into actions, in order to maximize the reward.

Karafotias et al. (2014) extend the framework presented in Eiben et al. (2006b) to a variety of state-of-the-art EAs for generic parameter control, with more encouraging results.

A different approach for generic and parameter-independent control is suggested by Aleti and Moser (2011): in their work they predict future parameter values according

to a linear time series forecasting strategy, based on OLS regression: instead of using past winners to compute the probability of using the parameter value in the next iterations, the authors fit a linear regression model with those values in order to derive a forecast for the future, of course assuming linearity of data. They show a strong and robust performance in their experimental setups.

Aleti and Moser (2016), in their sweeping systematic literature review, note that a challenge still to be addressed both by adaptive and generic quality attribution parameter control (e.g. Arabas et al. (1994), Bäck et al. (2000), Eiben et al. (2004) Maturana et al. (2010), Fialho (2010), Aleti and Moser (2011)) is the interaction between different parameters, as the quality of one algorithm parameter typically depends on other settings. These methods model the quality of the parameter setting based on a certain strategy (e.g. quality in Maturana et al. (2010) or di Tollo et al. (2015) is expressed in terms of the combination of fitness and diversity based on Hamming distance), in order to describe its performance based on past observations. Actually, to our knowledge, only Aleti and Moser (2011) have introduced a linear forecasting strategy, which instead of directly translating success ratios for selection probabilities in the future, propose a fitted model to predict the success ratio in the next iteration.

Chapter 2

Basics on portfolio selection

In this chapter we discuss briefly some basic results about portfolio selection: in particular, in section 2.1 we highlight some essential properties of risk measures, we shed light on the sensitivity of portfolios to input estimation and then we examine some regularization techniques in literature. Finally, we discuss the mixed-integer model that we use extensively in Chapter 4. Then, in section 2.2 we comment the issue of constraint handling in the context of EAs: in particular, we focus on penalty methods and after that we propose our approach. In what follows, for a given wealth W, we denote with $\mathbb{X} = \{X_1, \dots, X_N\}$ a set of investment choices in equity markets, i.e. a portfolio is a N-dimensional vector $x' = (x_1, \dots, x_N) \in \mathbb{R}^N$, such that each element in the portfolio x_i, with $i = 1, \dots, N$, represents a long/short position in a specific asset in the stock market. Furthermore, when dealing with practical constraints, it is not uncommon to deal with amounts of assets represented as multiples of trading lots, by imposing a round lot constraint. Note that it is immediate to relate the GA terminology to that of portfolio selection problem (PSP), as the population represents a collection of portfolios (individuals), composed of N assets (genes).

2.1 Basic formulation of the PSP and formal properties of risk measures

For a start, consider a general problem of this form, denoting in this case with X the set of feasible solutions and with $\mathbf{x}$ the asset weights:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \phi(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{x} \in X \end{aligned} \tag{2.1}$$

The formulation of the well-known quadratic programming problem proposed by Markowitz (1959) is the following:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \frac{1}{2}\sigma_p^2 = \frac{1}{2}\mathbf{x}^T \Sigma \mathbf{x} \\ \text{s.t.} \quad & \mathbf{x}^T \mathbf{r} = E_p \\ & \mathbf{x}^T \mathbf{1} = 1 \end{aligned} \tag{2.2}$$

With $\mathbf{\Sigma}$ denoting the covariance matrix and with $\mathbf{E_p}$ denoting the level of desired expected returns. Note that by imposing a nonnegativity constraint (i.e. $x_i \geq 0$, for $i = 1, \dots, N$), there is not a closed-form solution for the quadratic programming problem and KKT conditions must be satisfied, due to the presence of an inequality constraint (see Appendix A). In our work, we consider mainly non-convex and difficult objective functions, with many local minima which cannot be tackled with standard optimization techniques, like gradient-based methods (see, for instance, Gilli et al.

(2011)). Certain risk measures, like the *Omega ratio* or *VaR* can be reformulated to obtain a linear program. Though a rich literature has dealt with this possibility, the approaches presented over time typically do not accomodate integer constraints (Gilli et al. (2011)).

Another critical issue involves the identification of a set of properties that a risk measures should satisfy; the design of the objective function in portfolio optimization problems is of the utmost importance, since it may lead to unattractive results. For instance, a portfolio based on *VaR* minimization (see Table 2.1) displays unpleasant and unnatural properties (Lwin et al. (2017)); first all, it does not satisfy subadditivity (in a nutshell, given two vectors of returns r_1 and r_2 and a linear combination of them r_p, it does not always hold true that $VaR(r_p) < VaR(r_1) + VaR(r_2)$). Moreover, it does not take into account losses beyond VaR itself. Also the variance is non-subadditive unless the correlation between the assets is zero or negative. As a consequence, Artzner et al. (1999) have proposed the notion of *coherent* risk measure, i.e. a function $\mathcal{R}(\cdot) : \mathbb{R} \to \mathbb{R}^N$ of N random variables that satisfies:

1. **Translational invariance**: for a given constant c and a random variable, say, a vector of returns $X \in \mathbb{R}$, $\mathcal{R}(X + c) = \mathcal{R}(X) + c$;
2. **Positive homogeneity**: for a given real number c, $\mathcal{R}(cX) = c\mathcal{R}(X)$;
3. **Subadditivity**: for two given random variables $X \in \mathbb{R}$ and $Y \in \mathbb{R}$, $\mathcal{R}(X+Y) \leq \mathcal{R}(X) + \mathcal{R}(Y)$;
4. **Monotonicity**: $\mathcal{R}(X) \leq \mathcal{R}(Y)$ when $X \leq Y$.

We look into a variety of both coherent and non-coherent risk measures: we include in the set traditional measures assessed in earlier literature, like the *mean-variance* approach proposed by Markowitz (1959) and more recent developments, i.e. *equal risk contribution (ERC)* portfolios (Maillard et al. (2010)). In general, we adopt risk measures which satisfy as much as possible formal properties and meanwhile present attracting properties. A well-known issue of *mean-variance* portfolios is that they treat below-target and above-target expected returns equally, while alternative approaches attempt to include more realistic features into the model, like relaxing the normality hypothesis, in order to take into account a few stylized facts in financial practice, i.e. especially fat-tailedness and asymmetry of asset returns. Moreover, the practice of optimizing plain mean-variance portfolios with historical data, is susceptible to one more critical issue, which can be decomposed in multiple parts. It is well-known, indeed, that they are particularly sensitive to the estimation of input parameters, which are highly noisy, with small perturbations in the value of each input leading to large swings in the portfolio composition, leading to counterintuitive and highly concentrated portfolios. This is mainly due to the impact of the estimation error of both the covariance matrix and the vector of expected returns, resulting in unstable optimal portfolio weights at best, leading usually to precarious out-of-sample performance. The upshot of this discussion is that there is a large upside potential in overcoming these central issues in portfolio management.

A concise summary of cost functions used in our work is proposed in Table 2.1. Note that, as di Tollo and Roli (2008) point out, the distinction between objective function and cost function is crucial, as the former represents the function to be optimized, while the latter is the function guiding the search process. This problem occurs whenever a bicriteria mean-risk optimization model is considered to represent the investor preferences, whose risk aversion is typically handled by a proper trade-off coefficient: we will explore this topic more in depth, in section 4.1.1.

2.1.1 Dealing with input sensitivity and unstable solutions

In what follows, we consider a few techniques which tackle both the issue of managing estimation sensitivity and the problem of designing risk measures equipped with desirable and realistic properties in the optimization process.

Table 2.1: Summary of cost functions

Risk Measure	Cost Function	Notes
Omega Ratio	$\Omega = \frac{\int_{-\infty}^{r_d}(r_d - r)F(r)dr}{\int_{r_d}^{\infty}(r - r_d)F(r)dr}$	r_d is a threshold set by the investor, $F(r)$ is a probability density function of the returns r. This risk/reward ratio, devised by Keating and Shadwick (2002), takes into account the entire probability distribution of returns. The programming problem is non-convex, though it can be reformulated as a LP.
Mean-Variance	$MV_\lambda = \lambda r_p - (1-\lambda)\sigma_p^2$	We denote this quadratic programming problem with r_p and with σ_p^2 the portfolio return and the portfolio variance; it is a reformulated version of the classic MV problem (Markowitz (1959)), as we include the desidered return constraint in the objective function (see also Chang et al. (2000)), where $\lambda \in [0,1]$ is a risk-aversion parameter.
Mean-MAD	$M - MAD_\lambda = \lambda r_p - (1-\lambda)MAD(r_p)$	$MAD = \frac{1}{n}\sum_{i=1}^{n}\lvert r_{p_i} - \bar{r}_p\rvert$ is the average absolute distance between each return observation and the mean value of returns $\bar{r_p}$. A mean-MAD optimization problem has been formulated by Konno and Yamazaki (1991) as a linear program; we design the optimization problem by including the desidered return constraint in the objective function.
Two-sided	$\rho_{a,p}(r) = a\lVert(r_p - E(r_p))^+\rVert_1 + (1-a)\lVert(r_p - E(r_p))^-\rVert_p - E(r_p)$	This risk measure, proposed by Chen and Wang (2008), takes into account both sides of the loss distribution, with $(r_p - E(r))^- = \max(\bar{r}_p - r_p, 0)$ and $(r_p - E(r))^+ = \max(r_p - \bar{r}_p, 0)$ denoting, respectively, the downside and the upside of r_p. Therefore, the 'two-sided' approach takes a convex combination of the 1-norm of the upside and the p-norm of the downside, where a and p denote a pair of parameters by which it is possible to model the investor's risk attitude. Chen and Wang (2008) prove that the two-sided risk measure is also coherent.
Equal risk contribution	$f_{ERC}(\mathbf{x}) = \sum_{i=1}^{n}\left\lvert\frac{x_i(Cx)_i}{x^TCx} - \frac{1}{K}\right\rvert$	Equal risk contribution (ERC) strategies are based on the idea that weights $\mathbf{x}$ may be adjusted so that each asset equally contributes to portfolio risk, i.e. for K assets we have $\mathcal{RC}(\mathbf{x})_i = \frac{\sigma_p(\mathbf{x})}{K}$, as the portfolio volatility can be decomposed in $(Cx)_i$ risk components, therefore $\sigma_p(\mathbf{x}) = \sum_{i=1}^{n} x_i\frac{(Cx)_i}{\sqrt{x^TCx}}$. The seminal study of Maillard et al. (2010) show that RP portfolios are located between minimum variance and $1/n$ portfolios in terms of risk.
$VaR_{(1-\beta)}$	$VaR_{(1-\beta)} = F^{-1}(1-\beta)$	The value at risk is the $(1-\beta)$ quantile of the distribution function F of the portfolio loss (parametric VaR). The mean-VaR optimization problem is cumbersome, since VaR is a non-convex function of portfolio assets, with multiple stationary points; moreover, it does not display sub-additivity (Lwin et al. (2017)).
Expected Shortfall	$ES = VaR + \frac{1}{\beta}\int_{VaR}^{\infty}(1 - F(r))dr$	The expected shortfall is the conditional mean value of the losses exceeding $VaR_{(1-\beta)}$. It is largely known as a risk measure with many appealing properties, since it is a coherent risk measure that can be easily reformulated in a linear program. Scenario-based CVaR minimization problems via LP have been proposed first by Rockafellar and Uryasev (2000).

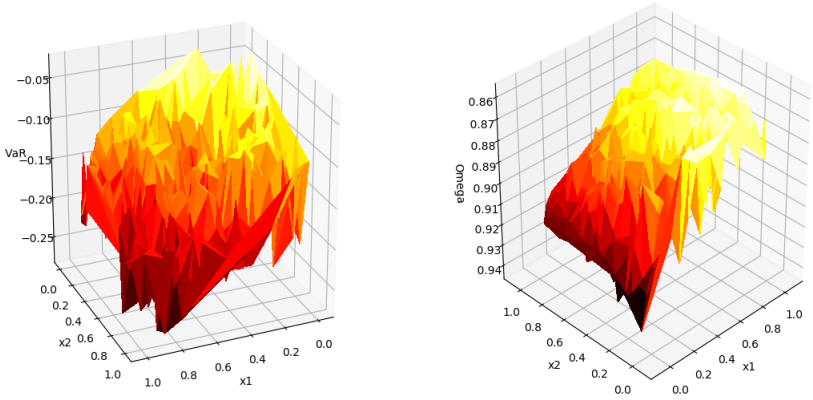


Figure 2.1: Cost functions for VaR and Omega, for a three asset optimization problem (shortselling allowed). The surfaces are non smooth, with many local maxima in both cases.

2.1.1.1 Measuring tail risks

Fishburn (1977) shed light on a comprehensive family of risk measures called *lower partial moment (LPM)*, i.e. below-target risk measures capturing the 'bad tail' of the distribution of returns, normally used as ex-post measures of performance. Nonetheless, some contributions (e.g. Gilli et al. (2006), Gilli et al. (2011), Lwin et al. (2017)) have dealt both with downside risk optimization and with risk/reward optimization in which risk and reward are treated, respectively, as lower and upper partial moments. Certain risk measures, as discussed below in Table 2.1, combine partial moments of the same order (e.g. *Omega ratio*), while others (e.g. the *Two-sided* risk measure) mix partial moments of different orders.

The general idea of partial moment is to catch return asymmetry around a threshold r_d, i.e:

$$\begin{aligned} \mathcal{M}_\gamma^+ &= \frac{1}{n}\sum_i^n (r_i - r_d)^\gamma \mathbf{1}_{\{r_i > r_d\}} \\ \mathcal{M}_\gamma^- &= \frac{1}{n}\sum_i^n (r_d - r_i)^\gamma \mathbf{1}_{\{r_i > r_d\}} \end{aligned} \tag{2.3}$$

With $+$ and $-$ denoting respectively the left and the right tail of an empirical return distribution and γ the order of each partial moment. Another way of dealing with the 'bad' tail the distribution is to consider quantiles and superquantiles (also known respectively as VaR and CVaR in finance), for a given distribution of losses l:

$$\begin{aligned} q(\alpha) &= F^{-1}(\alpha) \\ \bar{q}(\alpha) &= E[l | l > q(\alpha)] \end{aligned} \tag{2.4}$$

Another possibility is given by two-sided approaches (Rockafellar et al. (2006), Chen and Wang (2008)), in which upper and lower moments of different orders are combined to obtain a family of new risk measures. The idea is to extend the good properties of LPM and quantile-based measures, which are often not coherent though, to both sides of the return distribution. Compared to variance, which puts same weights to positive and negative deviations from the mean, the reasons behind the formulation of VaR or CVaR are well founded, as it has been shown empirically that investors value differently losses and gains (see e.g. Borges and Knetsch (1998), Kahneman and

Tversky (2013)). However, as Chen and Wang (2008) note, in economic terms, the strict separation of downside risk and upside potential may affect negatively the investment decision, by discarding potentially useful data and by representing portfolio risk only in terms of downside risk, which is somehow an 'incomplete' representation of the investment opportunity. Furthermore, to model correctly some stylized facts in financial markets, it is essential for the risk measure to capture the deviation from the mean in an asymmetrical way and to satisfy coherence. The attitude towards risk can be tuned accordingly with risk aversion parameters (see 2.1 for further details on the two-sided risk measure). This broad family of deviation measures proposed first by Rockafellar et al. (2006) is the following:

$$\mathcal{D}(r_p) = \|a(r_p - E(r_p))^+ + b(r_p - E(r_p))^-\|_p \tag{2.5}$$

For any $p \in [1, \infty]$, $a \geq 0$, $b \geq 0$ and $a + b > 0$.

Finally, another possibility is to give up on estimating expected returns, focusing on the so-called smart beta strategies, which focus only on managing risk. In this family, minimum variance, equally weighted and equally-weighted risk contribution portfolios Maillard et al. (2010) are included: the general idea, indeed, is to make the strategy less reliant on input estimates, with the purpose of constructing more robust and stable portfolios.

2.1.1.2 Assessing the sensitivity of MV portfolios to input estimation

The stability of mean-variance portfolios is of course one of the key themes of portfolio management, which we only sketch here for space limits. We introduce the topic by recalling a few elementary results, following the discussion in Roncalli (2013). First of all, consider the following standard quadratic portfolio optimization problem, without the non-negativity constraint, with γ denoting a risk-aversion parameter:

$$\mathbf{x}^* = \text{argmin}\, \frac{1}{2}\mathbf{x}^T\Sigma\mathbf{x} - \gamma\mathbf{x}^T\mathbf{r} \tag{2.6}$$

and its closed-form solution is $\mathbf{x}^* = \gamma\Sigma^{-1}\mathbf{r}$. Consider now the sample return vector $\mathbf{r}$ and the sample covariance matrix $\hat{\Sigma}$ and note that the solution to problem 2.6 is a function of the inverse of the covariance matrix, which we denote, as in Roncalli (2013), with $\mathcal{I} = \Sigma^{-1}$. The covariance matrix, as any diagonalizable matrix, can be factorized as follows:

$$\hat{\Sigma} = P\Lambda P^T \tag{2.7}$$

which is the eigendecomposition of $\hat{\Sigma}$, where P is an orthogonal matrix consisting of the n columns of the eigenvectors of $\hat{\Sigma}$ and $\Lambda = diag(\lambda_1, \ldots, \lambda_n)$ is the diagonal matrix of the eigenvalues, with $\lambda_1 > \lambda_2 > \cdots > \lambda_n$.

$$\begin{aligned} \hat{\Sigma^{-1}} = \mathcal{I} &= (P\Lambda P^T)^{-1} \\ &= \left((P^T)^{-1}\Lambda^{-1}P^{-1}\right) \\ &= (P\Lambda^{-1}P^T) \end{aligned} \tag{2.8}$$

The eigenvectors of $\mathcal{I}$ are the same of $\hat{\Sigma}$, while the eigenvalues of $\mathcal{I}$ are the inverse of th eigenvalues of $\hat{\Sigma}$. We denote each eigenvector as $\mathbf{e_i}$ and each vector $\mathbf{y_i} = \mathbf{e_i}^T\mathbf{r}$ is called principal component. It follows that:

$$\sum_i^n Var(\mathbf{r_i}) = tr(\hat{\Sigma}) = \sum_i^n \lambda_i = \sum_i^n Var(\mathbf{r_i}) \tag{2.9}$$

The key implication is that each eigenvalue of $\hat{\Sigma}$ explains a percentage of total variance, with $\dfrac{\lambda_i}{\lambda_i + \cdots + \lambda_n}$ denoting the amount of variance explained; moreover, an

interesting implication is that each component has an economic meaning, with the first one representing the market risk factor, the next eigenvectors denoting the common risk factors and the last eigenvectors representing the noise factors (Roncalli (2013)).

This result is clearly useful for two reasons. On the one hand, it could be exploited to obtain more robust portfolios, by removing noisy components and keeping the most informative factors. Furthermore, if we consider the solution to problem 2.6, we have that $\mathbf{x}^* = \gamma\Sigma^{-1}\mathbf{r} = \mathbf{x}^* = \gamma P\Lambda^{-1}P^T\mathbf{r}$, it follows that $P^T\mathbf{x}^* = \gamma\Lambda^{-1}P^T\mathbf{r}$, so we observe that the Markowitz closed-form solution of the quadratic program 2.6 is actually proportional to the vector of returns and inversely proportional to the eigenvectors. The unpleasant consequence is that the Markowitz approach has small exposure to common risk factors and to the market factor, with greater concentration on noisy factors. This well-founded argument provides a more than robust evidence to the poor out-of-sample performance of plain mean-variance portfolios, as they basically gets large exposure to noise, resulting in unstable and counterintuitive compositions. In figure 2.2 we propose an empirical test that backs all the previously mentioned results, based on the Hang Seng covariance matrix at December 2014. On the *y-axis* the fraction of explained variance of the ith component is reported, with the noisy factors explaining a large part of variance in the case of the information matrix, whereas in the case of the covariance matrix about 40% of variance is explained by the market factor.

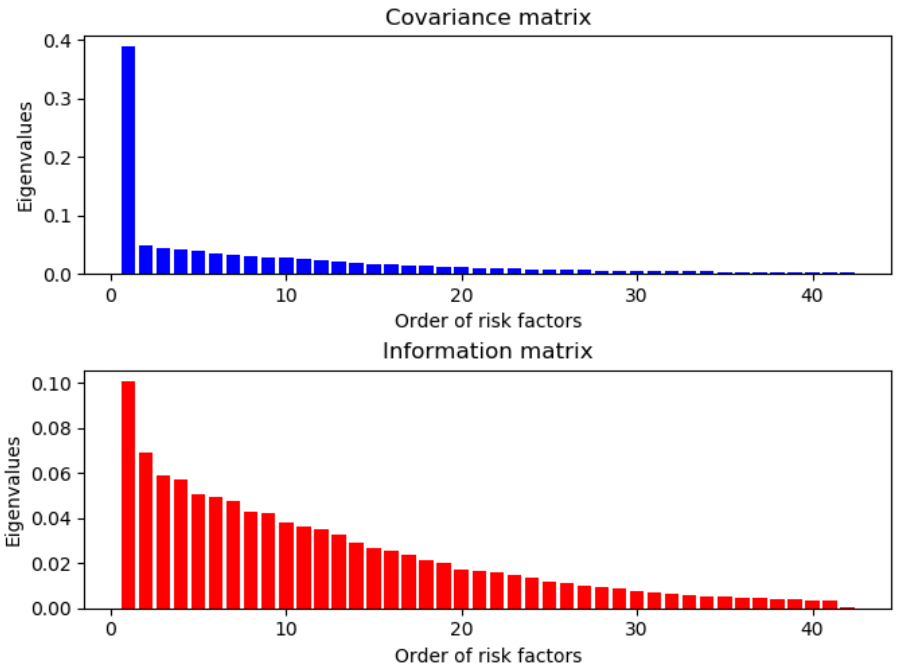


Figure 2.2: Eigenvalues of covariance and information matrices of stock returns (Hang Seng index). This example, based on our own data, is largely similar to the test reported in Roncalli (2013), section 1.2.

So far, we have considered a regularization technique of the covariance matrix both to explain the poor out-of-sample performance of standard mean-variance portfolios and to improve the performance itself by making the solution more stable. In what follows, we sketch a few more regularization strategies, following DeMiguel et al. (2009).

First of all, recall that the solution of problem 2.6 is $\mathbf{x}^* = \gamma\Sigma^{-1}\mathbf{r}$. Jagannathan and Ma (2003) propose to add i no-short sales constraints $\mathbf{x} > 0$ to the minimum

variance portfolio in order to regularize it: although the proof goes beyond the scope of this chapter, it can be shown that it is equivalent to determining an unconstrained minimum variance portfolio for which the sample covariance matrix is $\bar{\Sigma} = \hat{\Sigma} - \lambda \mathbf{1}^T - \mathbf{1}\lambda^T$, where $\lambda \in \mathbb{R}$ is the vector of Lagrange multipliers for the shortsale constraint. Given the KKT conditions, the constraint $\lambda_i \geq 0$ must be satisfied and if it is *active*, the sample covariance of asset i is reduced (or shrinked) by λ_i.

A similar approach has been proposed by Ledoit and Wolf (2003), in which the sample covariance matrix is replaced with a weighted average of the sample covariance matrix itself and a highly structured target estimator, which could be a covariance matrix with constant correlation or a multiple of the identity matrix:

$$\hat{\Sigma}_{shrink} = \alpha \hat{\Sigma} + (1 - \alpha)\hat{\Sigma}_{target} \tag{2.10}$$

with $0 < \alpha < 1$, $\hat{\Sigma}$ an unbiased estimator of the covariance matrix with large estimation error, $\hat{\Sigma}_{target}$ a biased estimator with little estimation error.

Finally, we provide a sketch of the family of penalization methods (see e.g. DeMiguel et al. (2009)), which are based on regularization problems of the linear regression model, with widespread applications in machine learning. The ℓ_1-norm constrained portfolio (also known as Lasso penalty) adds a penalty term, which is equal to the sum of the absolute value of the weights:

$$\begin{aligned} \mathbf{x}^* = \text{argmin } & \frac{1}{2}\mathbf{x}^T\hat{\Sigma}\mathbf{x} - \gamma\mathbf{x}^T\mathbf{r} + \lambda\|\mathbf{x}\| \\ \text{s.t.} \quad & \mathbf{1}^T\mathbf{x} = 1 \end{aligned} \tag{2.11}$$

It tends to generate sparser portfolios compared to standard mean-variance optimization; as a result, portfolios generated with the Lasso approach are generally more stable. The effect of penalizing each weight is to shrink the weights towards zero and forcing some of them to be equal to zero, provided λ is large enough, by performing variable selection. Though there is no closed-form solution, this approach is particularly useful because it typically leads to simpler and highly interpretable models, especially when large-scale optimization problems are taken into account. Jagannathan and Ma (2003) show that, under certain conditions, the solution of the ℓ_1 norm-constrained portfolio is equivalent to the solution of the portfolio with the short-sale constraint.

The ℓ_2-norm constrained portfolio (also called Ridge penalty) is defined as follows:

$$\begin{aligned} \mathbf{x}^* = \text{argmin } & \frac{1}{2}\mathbf{x}^T\hat{\Sigma}\mathbf{x} - \gamma\mathbf{x}^T\mathbf{r} + \lambda\mathbf{x}^T\mathbf{x} \\ \text{s.t.} \quad & \mathbf{1}^T\mathbf{x} = 1 \end{aligned} \tag{2.12}$$

Roncalli (2013) shows that problem 2.12 could be reformulated as follows:

$$\mathbf{x}^* = \text{argmin } \frac{1}{2}\mathbf{x}^T(\hat{\Sigma} + 2\lambda\mathbf{I_n})\mathbf{x} - \gamma\mathbf{x}^T\mathbf{r} \tag{2.13}$$

He notes indeed that equation 2.13 is a mean-variance portfolio with a modified covariance matrix, i.e. $\bar{\Sigma} = \hat{\Sigma} + 2\lambda\mathbf{I_n}$, which amounts to adding the quantity 2λ to the diagonal elements of the covariance matrix, which is actually similar to the shrinkage approach of Ledoit and Wolf (2003). The main advantage of ridge penalty is rooted in the tradeoff between bias and the estimation error: by imposing a constraint $\lambda\mathbf{x}^T\mathbf{x}$ the coefficients of the model are shrinked towards zero, making the model more robust and stable. The key difference between Lasso penalty and Ridge penalty is that it the latter shrinks all the coefficients slowly, whereas in the case of Lasso some coefficients decrease more quickly towards zero.

2.1.2 Mixed-integer programming (MIP) problems

We briefly summarize here the formulation of the optimization problem with integer constraints that we use in section 4.3 to perform a set of tests: here we focus briefly on

some issues involving the design of the programming problem; we take into account practical integer constraints for a start, then we propose our (general) optimization model. As clarified before, we extend the basic models with real-world constraints, which reflect better a practical approach of active asset management (Lwin et al. (2017)); most importantly, we can now cope with an alternative and more complex portfolio design, which integrates a variety of risk measures into the framework of mixed-integer portfolios. We consider two standard integer constraints:

- *Cardinality constraint*: cardinality constraints limit the number of K assets composing the portfolio; in this way we select a relatively small subset of the available assets in the problem instance. Imposing a constraint to the number of assets in the portfolio usually leads to better out-of-sample performance, as it reduces the complexity of portfolios, leading to a way more sparse representation, which in turn has two main benefits: on the one hand, sparse portfolios are more robust, as they are less sensitive to input parameters; on the other hand, a sparser representation dramatically reduces turnover;
- *Floor and ceiling constraints*: the floor and ceiling constraints are needed to further limit the proportion of each asset held in the portfolio, so we assume that each asset weight lies between bound l_i and bound u_i. This constraint (as well as the cardinality constraint) has been introduced by Chang et al. (2000) for portfolio selection problems (PSPs) and it has some of the benefits presented for the cardinality constraint, as enforcing a lower and an upper bound to each asset in the portfolio leads to more robust results (and likely better out-of-sample performance), as well as lower transaction costs. Especially in a Markowitz-like framework, upper and lower bounds are expected to favor a more stable solution, as it avoids extreme and unnatural concentrations in a few assets.

Therefore, we can now write the general mixed-integer portfolio selection problem as:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \phi(\mathbf{x}) \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i = 1 \\ & x_i \geq 0 \qquad\qquad i = 1, \dots, n \\ & \sum_{i=1}^{n} \delta_i = K \\ & \delta_i l_i \leq x_i \leq \delta_i u_i \quad i = 1, \dots, n \\ & \delta_i \in \{0, 1\} \qquad i = 1, \dots, n \end{aligned} \tag{2.14}$$

where $\delta_i \in \{0, 1\}$ represent N integrality constraints, namely δ_i is a binary variable to include or exclude asset i in the portfolio; moreover, with $\delta_i l_i \leq x_i \leq \delta_i u_i$ we impose N floor and ceiling constraints and with $\sum_{i=1}^{n} \delta_i = K$ we denote the cardinality constraint, i.e. we impose that exactly K active position are chosen from the market index. Finally with $x_i \geq 0$ we denote the no-short selling constraint and with $\sum_{i=1}^{n} x_i = 1$ we ensure that all the capital is invested in the portfolio. This formulation will turn out to be particularly useful in section 2.2, in which we study a penalty approach to incorporate the above mentioned integer constraints into the objective function.

2.2 A reformulation of the mixed-integer portfolio selection problem based on the exact penalty function

Recall that the solution of the portfolio selection problem (PSP) is given by a vector of N variables $x_1, \dots, x_N$, where each x_i represents a fraction of the amount invested in

asset i. Furthermore, we have already specified that heuristic approaches generally can deal with unconstrained problems; therefore, various strategies have been proposed to tackle both equality and inequality constraints. di Tollo and Roli (2008) propose a sweeping classification of search processes based on infeasibility handling:

1. *All feasible approach*: each candidate solution must satisfy the constraints at each step of the search process (Chang et al. (2000));

2. *Repair approach*: infeasible solutions are immediately 'repaired' in order to satisfy the constraints; consequently, the search algorithm basically is not allowed to visit unfeasible solution in the search space, as it is forced to correct the values of infeasible solutions (Lwin et al. (2017));

3. *Penalty approach*: the search algorithm is allowed to visit infeasible solutions, whose cost function receives an additional penalty depending on the amount of constraint violation (Corazza et al. (2021)).

In what follows, we sum up some key facts about the penalty strategies, with the aim of providing a general framework for the application of the exact penalty function to portfolio selection problems. In particular, in subsection 2.2.1 we introduce some results on penalty approaches, while in subsection 2.2.2 we define a specific model that we subsequently use to perform a set of tests in Chapter 4.

2.2.1 A brief introduction to penalty methods

The main references for this subsection are Nocedal and Wright (2006) and Bazaraa et al. (2013). The general idea behind penalty methods is to replace a (nonlinear) constrained optimization problem with an unconstrained one, in which the constraints are included into the objective function by means of a penalty term. We consider essentially three strategies:

- The *Quadratic penalty method* includes the constraints in the objective function by adding a multiple of the squared violation of all the constraints. The constrained problem is typically solved with a sequence of unconstrained penalized problems;

- The *Nonsmooth exact penalty method* replaces the original constrained problem with a single unconstrained problem, i.e. with a *nonsmooth exact* penalty function. It is possible to deal with the nonsmoothness of the penalty function by reformulating the nonlinear problem as a quadratic one, which can be tackled with a standard quadratic solver; otherwise, it is also possible to apply derivative-free methods;

- The *Augmented Lagrangian method*, which presents some similarities with the quadratic penalty methods and deals with some of its drawbacks, by introducing Lagrange multiplier estimates into the objective function.

2.2.1.1 Quadratic penalty method

Consider the following nonlinear minimization problem with equality and inequality constraints:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \phi(\mathbf{x}) \\ \text{s.t.} \quad & g_i(\mathbf{x}) = 0 \quad i = 1, \dots, m \\ & k_i(\mathbf{x}) \leq 0 \quad i = m+1, \dots, p \end{aligned} \tag{2.15}$$

Where $\mathcal{I} = \{i : k_i(\mathbf{x}) = 0\}$ is the set of *binding* constraints for which $k_i(\mathbf{x}) \leq 0$ is satisfied with equality. The problem can be reformulated as a quadratic penalty

function, which is defined as:

$$\mathcal{Q}(\mathbf{x}, \mu) = \phi(\mathbf{x}) + \frac{\mu}{2} \sum_{i=1}^{m} g_i^2(\mathbf{x}) + \frac{\mu}{2} \sum_{i \in \mathcal{I}} ([k_i(\mathbf{x})]^-)^2 \tag{2.16}$$

where $\mu > 0$ denotes a penalty parameter and where $[k_i(\mathbf{x})]^- = \max[0, k_i(\mathbf{x})]$. As Nocedal and Wright (2006) note, the algorithmic framework require a proper design. In order to ensure a fast convergence of the algorithm towards the minimum, the choice of μ_k at a given k iteration is particularly critical; for instance, given a tolerance threshold $\|\nabla_x \mathcal{Q}(x, \mu_k)\| \leq \tau_k$ as a stopping criterion, convergence towards the optimal value may not be achieved when the penalty parameter is not large enough.

The problem formulated in equation 2.16, as any unconstrained problem, can be tackled with a variety of algorithms, though, as reported in Nocedal and Wright (2006), the key drawback of this method is that it is particularly sensitive to the *ill-conditioning* in the Hessian $\nabla^2_{xx} \mathcal{Q}(x, \mu_k)$. We sketch briefly this problem, but we do not investigate the poor performance of some well-known unconstrained minimization algorithms when tacking this formulation of the problem. For the sake of simplicity, consider now the quadratic penalty function without inequality constraints: the gradient and the Hessian are respectively given by:

$$\begin{aligned} \nabla_x \mathcal{Q}(\mathbf{x}, \mu_k) &= \nabla \phi(\mathbf{x}) + \sum_{i=1}^{m} \mu_k g_i(\mathbf{x}) \nabla g_i(\mathbf{x}) \\ \nabla^2_{xx} \mathcal{Q}(\mathbf{x}, \mu_k) &= \nabla^2 \phi(\mathbf{x}) + \sum_{i=1}^{m} \mu_k g_i(\mathbf{x}) \nabla^2 g_i(\mathbf{x}) + \mu_k \nabla g_i(\mathbf{x})^T \nabla g_i(\mathbf{x}) \end{aligned} \tag{2.17}$$

We set $H = \nabla^2_{xx} \mathcal{Q}(\mathbf{x}, \mu_k)$ and we define the *condition number* of H as $\kappa(H) = \frac{|\lambda_{max}(H)|}{|\lambda_{min}(H)|}$, i.e. the ratio between the largest and smallest eigenvalues: informally, we say that for large values of $\kappa(H)$, matrix inversion is sensitive to error in input, namely the Hessian is ill-conditioned. The singular value decomposition of H is $H = U \Sigma V^T$, and if H is non-singular, we have $H^{-1} = V \Sigma^+ U^T$, where Σ^+_{ij} is equal to $\frac{1}{\Sigma_{ij}}$ if $i = j$, otherwise $\Sigma^+_{ij} = 0$. Intuitively, since $\lambda_{max} = \lambda_m = \Sigma_{mm}$, the source of potential instability is now apparent[1].

2.2.1.2 Nonsmooth exact penalty method

The second class of penalty functions is called *exact* because, for certain values of the penalty parameter, it requires one unconstrained minimization to yield an optimal solution, so that it does not rely upon the parameter update strategy discussed above. Quadratic penalty functions, instead, are not exact because their minimizers are not a solution of the nonlinear program for any positive μ. We comment briefly the ℓ_1 penalty function, which is nonsmooth due to the presence of the absolute value. As a consequence, it is not twice differentiable:

$$C_1(\mathbf{x}, \mu) = \phi(\mathbf{x}) + \mu \sum_{i=1}^{m} |g_i(\mathbf{x})| + \sum_{i \in \mathcal{I}} [k_i(\mathbf{x})]^- \tag{2.18}$$

We refer the reader to subsection 2.2.2, where we briefly discuss an important theorem, which ensures the exactness of the ℓ_1 penalty function in the context of portfolio selection problems: for a proof, see e.g. Bazaraa et al. (2013). Typically, in order to handle this class of penalty functions, C_1 is reformulated as a smooth

[1] Recall, for instance, that the Newton step for $\mathcal{Q}(\mathbf{x}, \mu_k)$ involves the computation of the inverse of the Hessian, i.e. $p = -\nabla^2_{xx} \mathcal{Q}(\mathbf{x}, \mu_k)^{-1} \nabla_x \mathcal{Q}(\mathbf{x}, \mu_k)$.

quadratic programming problem, while here we directly optimize the C_1 function via evolutionary algorithms.

Nocedal and Wright (2006) point out that the exactness of the ℓ_1 penalty function follows directly from its nonsmoothness. They provide an informal proof, which is sketched briefly below. Consider the ℓ_1 penalty function with one equality constraint $g_1(\mathbf{x}) = 0$:

$$C_1(\mathbf{x}, \mu) = \phi(\mathbf{x}) + \mu h(g_1(\mathbf{x})) \tag{2.19}$$

where h is a real-valued function, $h(\cdot) \geq 0$ and $h(0) = 0$. Supposing that h is continuously differentiable, given that h has a minimizer in zero, so $\nabla h(0) = 0$. If $\mathbf{x}^*$ is a solution of the problem, $g_1(\mathbf{x}^*) = 0$ and $\nabla h(g_1(\mathbf{x}^*)) = 0$. Finally, we have:

$$\nabla C_1(\mathbf{x}^*, \mu) = 0 = \nabla \phi(\mathbf{x}^*) + \mu \nabla g_1(\mathbf{x}^*) \nabla h(g_1(\mathbf{x}^*)) = \nabla \phi(\mathbf{x}^*) \tag{2.20}$$

However, as the authors argue, it is not generally true that $\nabla \phi(\mathbf{x}^*) = 0$ for constrained problems, therefore h is not differentiable and $C_1(\mathbf{x}, \mu)$ is not smooth.

2.2.1.3 Augmented Lagrangian method

Finally, we outline the Augmented Lagrangian method: for the sake of simplicity, we consider only equality constraints and we provide only an intuition behind this approach, which is related to the quadratic penalty function. It introduces also a key difference, by including explicit Lagrangian multiplier estimates in the objective function. Now, let us recall one more result in the context of quadratic penalty functions: Nocedal and Wright (2006) show that if a $\mathbf{x}^*$ is a KKT point there is a sequence K such that:

$$\lim_{k \in K} -\mu_k g_i(x_k) = \lambda_i^* \quad \text{for all } m \text{ equality constraints} \tag{2.21}$$

where λ_i^* is the Lagrangian multiplier vector for which the KKT conditions hold. The equality constraints are approximated as follows:

$$g_i(x_k) \approx -\mu_k \lambda_i^* \quad \text{for all } m \text{ equality constraints} \tag{2.22}$$

This result is crucial because it shows that a sequence $\{x_k\}$ achieves an inexact but increasingly accurate minimization of the quadratic penalty function $\mathcal{Q}(\cdot, \mu_k)$, i.e. we have that $g_i(\mathbf{x}) \to 0$ for $\mu_k \to \infty$. The Augmented Lagrangian method works around this undesirable property, by including the estimate 2.22 of the Lagrangian multipliers in the objective function with quadratic penalty. So we have that:

$$\mathcal{L}(\mathbf{x}, \lambda, \mu) = \phi(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i g_i(\mathbf{x}) + \frac{\mu}{2} \sum_{i=1}^{m} g_i^2(\mathbf{x}) \tag{2.23}$$

It is possible to prove that under certain conditions, when the exact Lagrange multiplier λ^* is known, $\mathbf{x}^*$ is a strict minimizer of $\mathcal{L}(\mathbf{x}, \lambda^*, \mu)$, with $\mu > \hat{\mu}$ and $\hat{\mu}$ a threshold value (Nocedal and Wright (2006)). Given that practically λ^* is not known, it is still possible to obtain a good estimate of $\mathbf{x}^*$, under the condition that λ^* is a good estimate of the true Lagrangian multipliers as well, even for small values of μ. In order to perform minimization with respect to $\mathbf{x}$, it is again necessary design a proper algorithmic framework, in which at every k iteration the penalty parameter $\mu_k > 0$ is increased and an estimate λ^k is determined. The optimality condition, denoting with $\mathbf{x_k}$ the approximate minimizer of $\mathcal{L}(\mathbf{x_k}, \lambda^k, \mu_k)$, is the following:

$$0 \approx \nabla \mathcal{L}(\mathbf{x_k}, \lambda^k, \mu_k) = \nabla \phi(\mathbf{x_k}) - \sum_{i=1}^{m} [\lambda_i^k - \mu_k g_i(\mathbf{x_k})] \nabla g_i(\mathbf{x_k}) \tag{2.24}$$

It follows that $\lambda_i^* \approx \lambda_i^k - \mu_k g_i(\mathbf{x_k})$ and by rearranging the expression one obtains $g_i(\mathbf{x_k}) \approx -\frac{1}{\mu_k}(\lambda_i^* - \lambda_i^k)$, so that if λ^k is close to the true value λ^* the infeasibility

of $\mathbf{x_k}$ is smaller than $\frac{1}{\mu_k}$. Finally, we note that the convergence of the Augmented Lagrangian method is ensured without setting $\mu \to \infty$: this approach makes the algorithm less dependent on the choice of μ_k and generally less prone to ill conditioning; the choice of the starting is less relevant as well. We refer to Nocedal and Wright (2006) for deitailed proofs of the above mentioned results.

2.2.2 Reformulating the portfolio selection problem

In this study, we focus on penalty methods and in particular we adopt the framework outlined by Fletcher (2013) and then extensively applied by Corazza et al. (2013) and Corazza et al. (2021) to solve complex portfolio selection problems; here we recap the main steps. Given a cost function $\phi : \mathbb{R}^N \to \mathbb{R}$ and two vector functions $g = \{g_1, \dots, g_m\} : \mathbb{R}^N \to \mathbb{R}^m$ and $k = \{k_{m+1}, \dots, k_p\} : \mathbb{R}^N \to \mathbb{R}^{p-m}$, the general nonlinear programming problem is the following:

$$\min_{x \in S} \phi(\mathbf{x}) \tag{2.25}$$

subject to the constraints

$$g_i(\mathbf{x}) = 0 \quad i = 1, \dots, m \tag{2.26}$$

$$k_i(\mathbf{x}) \leq 0 \quad i = m+1, \dots, p \tag{2.27}$$

Given the two class of constraints 2.26 and 2.27, we follow the notation adopted by Fletcher (2013) and we denote with S the feasible region:

$$S = \{\mathbf{x} |\ g_i(\mathbf{x}) = 0, i = 1, \dots, m; k_i(\mathbf{x}) \leq 0, i = m+1, \dots, p\} \tag{2.28}$$

Intuitively, the idea is to balance the need of minimizing the cost and meanwhile staying inside the feasible region. As noted by Fletcher (2013), it is possible to combine ϕ, g and k, i.e. by minimizing the former quantity coupled with a penalty for constraints g and k violation.

It is possible to associate to the constrained problem (2.25, 2.26 and 2.27) the following ℓ_1 penalty function:

$$C(\mathbf{x}, \varepsilon) = \phi(\mathbf{x}) + \frac{1}{\varepsilon} \left[\sum_{i=1}^{m} \|g_i(\mathbf{x})\|_1 + \sum_{i=m+1}^{p} \|k_i(\mathbf{x})\|_1 \right] \tag{2.29}$$

For portfolio selection problems, we rearrange the ℓ_1 penalty function as follows:

$$\begin{aligned} C(\mathbf{x}, \mathbf{z}, \varepsilon) =& \phi(\mathbf{x}) + \frac{1}{\varepsilon} \left[\left| \sum_{i=1}^{N} x_i - 1 \right| + \max \left\{ 0, \sum_{i=1}^{N} z_i - K \right\} + \sum_{i=1}^{N} \max\{0, z_i l_i - x_i\} \right. \\ & \left. + \sum_{i=1}^{N} \max\{0, x_i - z_i u_i\} + \sum_{i=1}^{N} \left| z_i (1 - z_i) \right| \right] \end{aligned} \tag{2.30}$$

With ε denoting a penalty parameter. Finally, our ℓ_1 penalty problem is:

$$\min_{\mathbf{x} \in \mathbb{R}^N, \mathbf{z} \in \mathbb{R}^N} C(\mathbf{x}, \mathbf{z}, \varepsilon) \tag{2.31}$$

The exact penalty function 2.30 is actually non-smooth due to the ℓ_1 penalization terms, which actually makes a genetic algorithm an attractive choice for the optimization problem at hand, given that the derivative of $C(\mathbf{x}, \mathbf{z}, \varepsilon)$ is not required. This will turn out to be particularly useful in section 4.3, in which rather than reformulating the problem as a smooth one, we will straightforwardly optimize the exact non-smooth function.

Furthermore, it is possible to prove (see for details the proof of Theorem 9.3.1 in Bazaraa et al. (2013)) that under certain conditions, there is a value of ε for which a solution $(\mathbf{x}^*, \mathbf{z}^*)$ of problem 2.31 is also a local minimum of 2.14. The theorem is stated as below:

Theorem 2.1. *Consider the general problem 2.25 and denote with* $\mathbf{x}^*$ *a KKT (Karush-Kuhn-Tucker) point (see appendix A) with Lagrange multipliers* u_i *for* $i = 1, \ldots, m$ *and* λ_i *for* $i \in \mathcal{I}$*, where* $\mathcal{I}$ *is the set of* binding *constraints, for which* $k_i(\mathbf{x}) = 0$*. Furthermore, suppose that* $\mathbf{x}^*$ *is a* regular *point (i.e. it satisfies regularity conditions provided in appendix A). Then, for* $\varepsilon \geq \max\{\lambda_i, i \in \mathcal{I}, |u_i|, i = 1, \ldots, m\}$*,* $\mathbf{x}^*$ *also minimizes the* ℓ_1 *penalized objective function* $C(\mathbf{x}^*, \varepsilon)$ *defined by 2.29.*

Chapter 3

A literature review of crossover operators

As outlined in Chapter 1, a dynamic strategy based on parameter control (which, of course, could involve both numeric and symbolic parameters) could be devised to tackle those optimization problems for which certain parameters give back a better performance at different stages of the search process and/or for different problem instances. In a nuthshell, the issue discussed and empirically studied in this section and in the AOS-related literature (see Maturana et al. (2010) and di Tollo et al. (2015)) is to select certain search operators dynamically in order to achieve an effective exploration/exploitation policy. In particular, we focus on a well-known subclass of EAs, i.e. Real Coded Genetic Algorithms (RCGA from now on). In broad terms, RCGAs are based on a real-encoding strategy to define a population of solutions, i.e. the solutions are first generated and then iteratively evaluated in an unbounded real-valued search space $\mathbb{R}^N$. Instead, in binary-coded genetic algorithms (BCGAs) binary-encoded solutions are represented according to a specific encoding scheme, so that specific crossover operators are required to manage them in a binary search space, namely $\mathbb{B}^l$, with $\mathbb{B} = \{0, 1\}$ and $l-$ string length (see Beyer and Deb (2001)). Herrera et al. (1998) stress the fact that it is more natural to represent optimization problems with variable in continuous search spaces, so that the solution is a vector of floating point numbers. Furthermore, Herrera et al. (1998) and Herrera et al. (2005) find out that binary coding is not suitable for large-scale dimension problems or for all the problems that require a high numerical precision (i.e. because the fixed string length affects the precision of the solution and actually an appropriate length of the stringth is not known *a priori*. With RCGAs, instead, the vector size equals the dimension of the problem at hand, so that each gene represets a variable of the problem; finally, Herrera et al. (1998) argue that another advantage is their ability to exploit the graduality of the functions with continuous variables, i.e. small changes in the variables lead to a small change in the objective functionm whereas in a BCGA framework usually a so-called *Hamming cliff* problem arises; this issue refers to the fact that two neighboring solutions may have very different binary representation, leading potentially to convergence towards local optima under certain conditions (see Goldberg (1988) and Herrera et al. (1998)). The chapter is structured as follows: in section 3.1 we explore some design principles for crossover operators and we propose some guidelines; then, in section 3.2 we propose a taxonomy of real-coded crossover operators in the context of GAs, largely based on the contribution of Herrera et al. (1998) and Herrera et al. (2005).

3.1 A discussion of crossover operators design principles

The point of choosing a broad set of different crossover operators is to take advantage of the potential of each operator in a specific stage of the search process: indeed, as noted by Herrera et al. (2005), the quality of the solutions in the visited region is problem-dependent (which is of course consistent with the NFL theorem, Wolpert and Macready (1997)) or could even depend on the stage of the search process, for a given problem. The authors propose, as a consequence, a comparative study based on hybrid crossover, in order to combine many features of different crossover operators; indeed, each operator is characterized by a search bias. In order to be more effective, it should be adjusted to adapt to the structure of the problem. Certain crossover operator could turn out to be more suitable than others in solving specific problems (Herrera et al. (2005)). Finally, we mention also the contribution of Yoon and Moon (2002), who dig into a slightly dissimilar approach, based on synergies between operators.

3.1.1 Some guidelines

In the last twenty years, many authors have posed the problem of defining a set of 'guidelines' (Herrera et al. (2005), Kita and Yamamura (1999)) to design correctly genetic operators and more specifically, much attention has been put in particular on the crossover ones.

Here we limit ourselves to highlight the key points from these literature surveys, which we set as referring point in the process of building up an operator list for dynamic and adaptive operator selection.

First of all, Kita and Yamamura (1999) propose a 'functional specialization hypothesis', which basically states that the selection operator should gradually narrow the probability density function (p.d.f.) of the population, while the crossover operator should be designed to preserve the probability density function. The authors motivate this statement as follows: if the crossover operation widens the p.d.f. of the population, an amount of computation is wasted to search in vain to look for solution in an area already cut out by the selection operator; instead, if the crossover operator narrows the p.d.f., it looks for solutions in a subregion specified in the selection process, so that the remaining search space is utterly ignored, which is for sure undesirable.

A broader strategy for designing crossover operators is the one presented by Herrera et al. (2005), who suggest the following set of guidelines:

1. The statistics of the population, i.e. the mean vector and the variance-covariance matrix, should be invariant with respect to the crossover operations;

2. Offsprings should have as much diversity as possible, given the constraint in Guideline 1;

3. Guideline 1 should be dropped if the selection operator does not work 'properly' (i.e. it fails to suggest a good region); the user may consider to design operators in such a way that the population variance is increased for each breeding process.

Moreover, some authors suggest a set of postulates (e.g. Beyer and Deb (2001), Deb et al. (2002)) which, according to their argument, should represent a referring point for future development of crossover operators.

They argue that certain crossover operators display a self-adaptive behaviour, i.e. they generate additional diversity along time (explorative crossover) or use diversity to create better individuals (exploitative crossover). Then they propose the following postulates:

1. *Postulate 1:* Under a variation operator, the expected value of fitness should not change.

2. *Postulate 2:* The variance of the resulting children population should be greater than that of the parent population.

The first postulate is based on a simple idea: the variation operator should not use fitness information (though that is not always the case), therefore the population average fitness should not vary as well; their point is that the task of the crossover operator is simply to lead the population towards a suitable direction, while increasing the variance of the resulting population.

The second postulate is grounded in an intuitive explanation too, which is by the way similar to that of Kita and Yamamura (1999): given that the selection operator tends to reduce the population variance, it should be increased by the crossover operator, with the aim of keeping a reasonable amount of diversity in the population. These two properties should always be satisfied when designing recombination operators.

From this perspective, Herrera et al. (2005) discuss a very encompassing taxonomy of crossover operators, which is rearranged here for a general discussion: our purpose is indeed to use the classes proposed in the literature to favour a formal analysis in the context of adaptive and dynamic operator selection for the PSP. We will focus our attention on the so-called 'Neighborhood-based' crossover class, by including some more recent contributions to the field, which has for sure displayed promising results for optimization problems.

3.2 A taxonomy of RCGA crossover operators

In this section we discuss a broad assortment of real-coded search operators, in a very general and not-too technical framework and we provide a reasoned analysis of them. Though the search space has been originally based on a binary encoding (Goldberg (1988)), we stick to the tradition of GA-based portfolio selection literature, which typically focuses on a real representation of each solution vector in the population, which is of course a very straightforward approach for continuous problems. We recall that a key building block of GAs is the *mating* process, which consists in sharing information between vectors during the search process, by combining the features of each parent, which in turn pass their 'genetic' material to the offsprings. As pointed out by Herrera et al. (2005), the crossover operator is the main search operator which leads the search process towards different areas of the solution space. The core idea is to generate offspring solutions in a neighborhood of the parents; the degree of proximity is typically relevant to guide the search process properly. In this section, assets and chromosomes, genes and weights have exactly the same meaning, as discussed in Chapter 2.

3.2.1 Discrete crossover operators: uniform and n-point recombination

The class of discrete crossover operators is borrowed from Herrera et al. (2005) and it includes all the crossover operators applied in the binary encoded GA, for which a version for RCGAs is admitted, mainly the *n-point crossover* and the *uniform crossover*. Basically, the value of each k-th gene $y_i(k)$ in the offspring is always included in his parents' genetic material. According to the classification of the authors, given two parents' genes $x_i(k)$ for $i = 1, 2$ and denoting with $\alpha_i = \min\{x_1(k), x_2(k)\}$ and with $\beta_i = \max\{x_1(k), x_2(k)\}$, for an arbitrary interval $[a_i, b_i]$ with $a_i < \alpha_i$ and with $\beta_i < b_i$, then the intervals $[a_i, \alpha_i]$, $[\beta_i, b_i]$ are considered explorative and the interval $[a_i, b_i]$ is deemed exploitative Herrera et al. (1998). Note that this approach is highly flexible, as the user can easily detect *a priori* the expected behaviour of an operator; potentially, she/he is also allowed to tweak accordingly some tunable parameters, which are normally designed to determine the desidered level of exploration.

In this section we do not provide many insights, instead we focus solely on practical issues; note that discrete crossover operators (see for instance Holland (1992)) are well-known for being the most exploitative ones, given that, at each step of the optimization process, the offspring population is made up with the same genetic material inherited from the parents. As a consequence, those methods are somewhat lacking in visiting certain subregions of the search space. In what follows, we use a set of acronyms to identify each operator. A sweeping classification of each of them is available for instance in Herrera et al. (1998).

- **Single point crossover [OPX]**

  The single point crossover is the simplest crossover operator, which has been first proposed in the seminal works of Holland (1992) and Goldberg (1988) generates an offspring by mating two parents by means of a *cutting point*, i.e. a randomly chosen position in the portfolio, by which the genetic material of each parent is split into two parts. It has been widely used in PSP literature and among many contributions we refer the reader, for instance, to Chang et al. (2000), whereas Lin and Liu (2008) manage the vector of assets for a PSP with minimum lots with one-point crossover. Given a population of N individuals $\{x_1, x_2, \ldots, x_N\}$, each individual is characterized by n genes; let x_i denote a parent and with y_i an offspring, then we have:

$$x_i = [x_i(1), x_i(2), \ldots, x(N)] \tag{3.1}$$

$$y_i = [y_i(1), y_i(2), \ldots, y(N)] \tag{3.2}$$

  Consider now two parents x_a and x_b and a cutting point $k \in \mathbb{N}$ randomly chosen in the interval $[1, n-1]$. Then, the two parents are mated to generate two offsprings, which in turn inherit the genes from their parents in a random fashion, as follows:

$$y_1(n) = [x_a(1), x_a(2), \ldots, x_a(k), x_b(k+1), \ldots, x_b(N)] \tag{3.3}$$

$$y_2(n) = [x_b(1), x_b(2), \ldots, x_b(k), x_a(k+1), \ldots, x_a(N)] \tag{3.4}$$

  In a nutshell, the cutting point k determines how many genes the parent x_a is expected to pass to the first offspring; then the residual amount of genetic material is passed to the second offspring. A similar procedure is carried out for parent x_b in the mating process.

- **Multiple point crossover [MPX]**

  The multiple point crossover, discussed by Goldberg (1988), is pretty much an extension of the single point crossover and it is based on a very similar idea. Basically, the procedure includes more than one cutting point in the procedure. Consider, for instance, a three point crossover; three numbers are randomly drawn, so that we three different cutting point are considered (say k_1, k_2, k_3). It goes without saying that cutting points are drawn one after another, in order that the inequality $k_1 < k_2 < k_3$ is verified and they are defined over the $[1, n-1]$ interval. Hence, once the cutting point are determined, the offspring inherits, say, the first portion of genes (assets for PSP) $[1, k_1]$ from parent x_a, the second portion $[k_1, k_2]$ from parent x_b, the third one $[k_2, k_3]$ from parent x_a and finally the last portion $[k_3, N-1]$ from parent x_b, as follows:

$$\begin{aligned} y_i = &[x_a(1), \ldots, x_a(k_1), x_b(k_1+1), \ldots, x_b(k_2), x_a(k_2+1), \\ &\ldots, x_a(k_3), x_b(k_3+1), \ldots, x_b(N)] \end{aligned} \tag{3.5}$$

  The remaining values in the solution vectors, i.e. the first and the third portion of genes of parent x_a, the second and the last portion of genes of parent x_b may be allocated to a second offspring.

Algorithm 9: Single point crossover

Input : $\mathcal{T}$: termination criterion
$P(\cdot)$: population of size $\mathcal{S}$

$t = 0$, $i = 0$
initialize and evaluate $P(t)$
while *termination criterion* $\mathcal{T}$ *not met* **do**
  $t = t + 1$
  while $i < \mathcal{S}$ **do**
    select two parents x_a and x_b from $P(t-1)$
    draw a cutting point $k \in \mathbb{N}$ randomly chosen in the interval $[1, N-1]$
    'mate parents x_a and x_b:'
    $x_a = [x_a(1), x_a(2), \ldots, x_a(N)]$, $x_b = [x_b(1), x_b(2), \ldots, x_b(N)]$
    'generate an offspring y_i:'
    $y_i = [x_a(1), x_a(2), \ldots, x_a(k), x_b(k+1), \ldots, x_b(N)]$
    $i = i + 1$
  end
  recombine $P(t)$
  evaluate $P(t)$
end

- **Uniform crossover [UX]**

  The uniform crossover, devised by Syswerda (1993), generates an offspring by mating two parents. For instance, Chang et al. (2009) apply this variation operator to a set of non-convex PSPs with integer constraints. For a The $k-th$ gene is passed either from parent x_a with probability $p = 0.5$ or by parent x_b with probability $1 - p = 0.5$, for each $k \in [1, n]$ position, as follows:

$$y_i(k) = \begin{cases} x_a(k) & \text{with probability } p = 0.5 \\ x_b(k) & \text{with probability } (1-p) = 0.5 \end{cases}$$

- **Global uniform crossover [GUX]**

  The global uniform has been proposed by Simon (2013) and it is a generalization of the two-parent uniform crossover. The point of this approach is to extend the parent pool to the whole population, so that the $k - th$ gene is selected with probability $p = 1/N$ from the $i - th$ parent. This standard algorithm, though, chooses weights in a completely random fashion; as a consequence, one may redefine the probability of selecting the $k-th$ in accordance with a fitness-based criterion (for instance, the most popular method allocates the normalized fitness of each parent in the pool to a "roulette-wheel", encouraging and biasing the selection toward more fit individuals).

- **Queen bee crossover [QBX]**

  This method, discussed in Sung (2007), mimics the queen bee evolution in natural evolution; the author tests this recombination tool with a combinatorial problem and two (continuous) function optimization problems. We note here, just as done by the author, that a major flaw in this operator is its fast convergence to (potentially) local solutions. The idea is the following. The "queen-bee" represents the fittest parent, which is mated, for each step, with a randomly chosen parent, which in turn is dropped at the following step and he or she is replaced with another parent and so on. The author points out that, in order to have the parents mated, a wide array of choices is available (e.g. simple crossover techniques, like uniform or multiple point crossover strategies).

Algorithm 10: Multiple point crossover

Input : $\mathcal{T}$: termination criterion
$P(\cdot)$: population of size $\mathcal{S}$

$t = 0$, $i = 0$
initialize and evaluate $P(t)$
while *termination criterion* $\mathcal{T}$ *not met* **do**
  $t = t + 1$
  while $i < \mathcal{S}$ **do**
    select two parents x_a and x_b from $P(t-1)$
    draw $\{k_1, \ldots, k_n\} \in \mathbb{N}$ cutting points randomly chosen in the interval $[1, n-1]$, such that $k_1 < \cdots < k_n$
    'mate parents x_a and x_b:'
    $x_a = [x_a(1), x_a(2), \ldots, x_a(N)]$, $x_b = [x_b(1), x_b(2), \ldots, x_b(N)]$
    'generate an offspring y_i:'
    $y_i = [x_a(1), x_a(2), \ldots, x_b(k_1), \ldots, x_a(k_2), \ldots, x_b(k_3), \ldots, \ldots, x_b(N)]$
    $i = i + 1$
  end
  recombine $P(t)$
  evaluate $P(t)$
end

3.2.2 Aggregation-based crossover operators

The key idea behind this class is to group those crossover operators which breed individuals by means of an aggregation function which combines the gene values of the parents in order to determine the gene values of the offsprings, i.e. for two parents x_i^1 and x_i^2 we have $y_i^1 = f(x_i^1, x_i^2)$. A wide range of solutions has been proposed over time and many benefit are associated with the flexibility given by one or more tunable parameters.

- **Arithmetic and average crossover [AX]**

  The arithmetic crossover, devised by Michalewicz (1992), produces two offspring by mating two parents for each iteration. The resulting offsprings, are generated, gene by gene, as a weighted average mean of his or her parents genes, definitely displaced in the same position of the resulting offspring gene, as follows:

  $$y_i(k) = \beta x_a(k) + (1 - \beta) x_b(k) \tag{3.6}$$

  A second offspring is generated similarly by inverting the β weights. Herrera et al. (2005) argues that this search technique is an exploitative one, since every offspring generated by this procedure is displaced into a space bounded by their parents genes.

  The average crossover, mentioned in Michalewicz (1992) and also discussed in Simon (2013), simply derives the $k-th$ gene in the offspring as an average of his parents genes, drawn exactly from the same position in the vector representing an individual, as follows:

  $$y_i(k) = (x_a(k) + x_b(k))/2 \tag{3.7}$$

  Note that the average crossover is a special case of the arithmetic one, indeed $\beta = 0.5$. The general idea behind the arithmetic crossover lends itself for many other variants. Some of them are mentioned in the following paragraph, but note that it is possible to tweak the strategy by mating many more parents (i.e. a multi-parent crossover) rather than two and then by taking the average of them.

Algorithm 11: Uniform crossover

Input : $\mathcal{T}$: termination criterion
$P(\cdot)$: population of size $\mathcal{S}$

$t = 0$, $i = 0$
initialize and evaluate $P(t)$
while *termination criterion* $\mathcal{T}$ *not met* **do**
  $t = t + 1$
  while $i < \mathcal{S}$ **do**
    select two parents x_a and x_b from $P(t-1)$
    for $j = 0 : N$ **do**
      draw a random number $r \in U[0, 1]$
      if r>*0.5* **then**
        $y_i(j) = x_a(j)$
      else
        $y_i(j) = x_b(j)$ % generate a N dimensional offspring y_i
      end
    end
    $i = i + 1$
  end
  recombine $P(t)$
  evaluate $P(t)$
end

- **Geometrical crossover [GX]**

  The geometrical crossover has been first proposed by Michalewicz et al. (1996) and it could be also generalized, by including more than two parents in the mating pool. In the above mentioned article, some simple test are carried out to compare its performance to standard crossover operators. We limit ourselves here to note that, by mating two parents, an offspring could be generated in this way:

  $$y_i(k) = \sqrt{x_a(k) \cdot x_b(k)} \qquad \text{for each } k - th \text{ gene} \tag{3.8}$$

  They show that, consistently with the NPL theorem Wolpert and Macready (1997), this recombination operator turns out to be a useful problem-specific operator for some particular constrained optimization problems; clearly, geometrical crossover can be applied only for those problems where the decision variables are entirely non-negative. They show that for a few test cases the GA combined with the geometric crossover shows a promising behaviour in boundary search. Furthermore, it outperforms the arithmetic crossover in terms of speed of convergence.

- **Simplex crossover [SPX]**

  The design of the simplex crossover has been proposed by Renders and Bersini (1994)) and it can be viewed as a generalization of the arithmetic crossover (see Michalewicz and Schoenauer (1996)). A simplex is a geometrical figure with $(n + 1)$ vertices in a n-dimensional space. The methods starts with an initial simplex and then it evolves through a series of geometric transformations (reflection, contraction, extension), so that the initial simplex adapts itself to the fitness landscape. At each iteration, the algorithm compares the relative order in terms of fitness of the points, replacing at each transformation the current worst point with a better one.

  The first step involves the computation of a centroid: the operator selects $k > 2$ parents, then the best and the worst individual (in terms of fitness) are deter-

Algorithm 12: Global uniform crossover

Input : $\mathcal{T}$: termination criterion
$P(\cdot)$: population of size $\mathcal{S}$

```
t = 0, i = 0
initialize and evaluate P(t)
while termination criterion T not met do
    t = t + 1
    while i < S do
        select all parents x_a,...,x_N from P(t − 1)
        for k = 0 : N do
            draw a random number r ∈ U[0, 1]
            if z/N < r < (z + 1)/N then
                y_i(k) = x_z(k)
            end
        end
        i = i + 1
    end
    recombine P(t)
    evaluate P(t)
end
```

mined within the group G, then its centroid c is computed by removing the worst individual x_i^w from it, as follows:

$$c = \sum_{x_i \in G - x_i^w} x_i/(k-1) \tag{3.9}$$

Then the 'reflected point' is computed:

$$x_r = c + (c - x_i^w) \tag{3.10}$$

If x_r is better (in terms of fitness) than the best selected individual x_{best}, an expanded point x_e is determined:

$$x_e = x_r + (x_r - c) \tag{3.11}$$

If x_e is better than x_r, the offspring is x_e, else the offspring is x_r; otherwise if x_r is not better than x_{best} but it is better than x_w, then the offspring is x_r.

Renders and Bersini (1994) argue that their goal is to enhance the local exploitation properties of GA by biasing, in particular, the crossover operator; furthermore, they note that the simplex crossover, being dependent on fitness information, includes a 'hill climbing' mechanism, enhancing as a consequence the local exploitation properties of GA. According to their tests, they find out that the standard GA focus on the 'global facet' (i.e. it is efficient in detecting areas of the search space likely to be high fitness), while it looks weak on the 'local facet'. Conversely, they claim that the hill climbing algorithms rely heavily on local exploitation.

- **Linear crossover [LNX]**

  The linear crossover has been devised by Wright (1991) to overcome some drawbacks of the n-point crossover. The authors propose to generate three offsprings: each $k-th$ gene in the offspring is computed as a linear combination of the $k-th$ gene in parent 1 and in parent 2. In this way, the algorithm typically gives back an offspring which behaves like the one determined by average crossover, and

Algorithm 13: Queen bee crossover

Input : $\mathcal{T}$: termination criterion
$P(\cdot)$: population
$\mathcal{I}_q$: queen bee
$\mathcal{I}_m$: selected bees

```
t = 0
initialize and evaluate P(t)
while termination criterion T not met do
    t = t + 1
    select P(t) from P(t − 1)
    P(t) = {I_q(t − 1), I_m(t − 1)}
    Recombine P(t)
    do crossover
    evaluate P(t)
end
```

Algorithm 14: Arithmetic crossover

Input : $\mathcal{T}$: termination criterion
$P(\cdot)$: population of size $\mathcal{S}$
β: a constant value $c \in [0, 1]$

```
t = 0, i = 0
initialize and evaluate P(t)
while termination criterion T not met do
    t = t + 1
    while i < S do
        select two parents x_a and x_b from P(t − 1)
        for k = 0 : N do
            y_i(k) = β x_a(k) + (1 − β) x_b(k)
        end
        i = i + 1
    end
    recombine P(t)
    evaluate P(t)
end
```

it is cointained in the interval bounded by the parents, while the other two are generally located outside it. Finally, the authors choose to bias the selection, by picking one or two offspring out of three according to their fitness. The essence of this approach is to keep the randomness of the crossover operator under control, by letting the search operator itself looking for best individuals; the resulting population is definitely expected to be fitter and the whole process is expected to converge rapidly to a solution, at the expense of some (useful) diversity in the population.

$$\begin{cases} y_i(k) & = 0.5 \cdot x_1(k) + 0.5 \cdot x_2(k) \\ y_{i+1}(k) & = 1.5 \cdot x_2(k) - 0.5 \cdot x_2(k) \\ y_{i+2}(k) & = -0.5 \cdot x_1(k) + 1.5 \cdot x_2(k) \end{cases}$$

- **Direction-based crossover [DBX]**

  The direction-based crossover, devised by Arumugam et al. (2005), is a variant

Algorithm 15: Geometrical crossover

Input : $\mathcal{T}$: termination criterion
$P(\cdot)$: population of size $\mathcal{S}$
β: A constant value $c \in [0, 1]$

```
t = 0, i = 0
initialize and evaluate P(t)
while termination criterion T not met do
    t = t + 1
    while i < S do
        select two parents x_a and x_b from P(t − 1)
        for k = 0 : N do
            y_i(k) = sqrt(x_a(k) · x_b(k))
        end
        i = i + 1
    end
    recombine P(t)
    evaluate P(t)
end
```

of the linear crossover in the sense that the population is biased towards fitter individuals, i.e. problem-specific information is introduced in the search process.

The direction-based crossover uses the fitness function information to determine the direction of the search in the following way. First, we denote with $f(\cdot)$ a generic fitness function and with r a random number such that $r \in U[0, 1]$. For two given parents x_a and x_b, assuming that $f(x_b) < f(x_a)$, the operator generates an offspring y_i according to the following rule:

$$\begin{cases} y_1 = r(x_b - x_a) + x_b & \text{if } f(x_b) < f(x_a) \\ y_1 = r(x_a - x_b) + x_a & \text{otherwise} \end{cases}$$

- **Heuristic crossover [HX]**

  The heuristic crossover, devised by Wright (1991), is a more sophisticated variant of the direction-based crossover; in this case the crossover is still based on a fitness criterion, but it proposes a slightly different approach when dealing with constrained optimization problems. The parents are chosen in the mating pool and their fitness, before running the usual procedure of gene transmission, is compared. Afterwards, a random number r is generated from a uniform distribution $U[0, 1]$ and the genes are chosen accordingly with the rule displayed below.

  The heuristic crossover has been heavily applied by Michalewicz and Schoenauer (1996) and Michalewicz (1995) for constrained optimization problems. In this case, it produces one offspring, but it may also not produce any offspring; if the offspring vector is not feasible, another random number is generated and another offspring is produced. After w attempts, if no feasible solution is generated (i.e. the constraints are not met), then the crossover operator gives up and the GA moves on with the ensuing step.

  Herrera et al. (2005) shows that a considerable shortcoming is the tendency to converge prematurely; this procedure generates systematically offsprings in the exploration zone, which is located on the left side of the best parent. Wright (1991) suggests to apply the heuristic crossover operators in advanced stages of the GAs. Michalewicz and Schoenauer (1996) find that a modified version of the *Genocop* system, including the heuristic crossover, displays a superior

Algorithm 16: Simplex crossover

Input : $\mathcal{T}$: termination criterion
$P(\cdot)$: population of size $\mathcal{S}$
β: A constant value $c \in [0, 1]$

$t = 0$, $i = 0$
initialize and evaluate $P(t)$
while *termination criterion* $\mathcal{T}$ *not met* **do**
  $t = t + 1$
  while $i < \mathcal{S}$ **do**
    select three parents x_k, with $k = 1, \ldots, 3$ from $P(t-1)$
    compute the centroid $c = \sum_{x_k \in G - x_k^w} x_k/(K-1)$
    compute the reflected point $x_r = c + (c - x_k^w)$
    denoting with $f(\cdot)$ a generic fitness function, if $f(x_r) > f(x_{best})$, compute an expanded point:
    $x_e = x_r + (x_r - c)$
    if $f(x_e) > f(x_r)$, $y_i = x_e$, else $y_i = x_r$, otherwise if $f(x_w) < f(x_r) < f(x_{best})$, then $y_i = x_r$
    $i = i + 1$
  end
  recombine $P(t)$
  evaluate $P(t)$
end

performance with respect to the precision of the solution; overall, their tests show that it is particularly effective for fine local tuning. Note that the resulting population is expected to display less diversity and a greater mean vector after the crossover stage. For maximization problems we have:

$$y_i(k) = x_b(k) + r(x_a(k) - x_b(k)) \quad \text{if } f(x_b) > f(x_a) \tag{3.12}$$

3.2.3 Neighborhood-based crossover operators: mean and parent-centric strategies

In this section we discuss a very broad family of crossover operators: actually the 'neighborhood-based' family of operators proposed in Herrera et al. (2005) covers so many strategies that we feel the importance of making a distinction between the *mean-centric* and the *parent-centric* operators (for a thorough discussion, see Deb et al. (2002)). Indeed, the authors consider the crossovers which determine the genes of the offspring from intervals defined in neighborhoods associated with the genes of the parents, by means of a probability distribution. This is actually a very general definition, which does not take into account two different approaches extensively studied in literature. We recall the importance of the guidelines presented at the beginning of this chapter, in particular, the two postulates discussed in Beyer and Deb (2001) are still considered the cutting-edge criteria for the design of crossover operators.

On the one hand, the mean-centric recombination preserves the mean vector of the population, i.e. offsprings are displaced near the centroid of the parents; on the other hand, the parent-centric recombination creates offsprings near the parents. The idea is still to preserve implicitly the mean vector once the crossover operation is carried out, because the expected population mean of the offspring population is equal to that of the parent population (Deb et al. (2002)): guideline 1 is definitely fulfilled. The only difference among these approaches is the region to be biased, the centroidal or the parental one.

Algorithm 17: Linear crossover

Input : $\mathcal{T}$: termination criterion
$P(\cdot)$: population of size $\mathcal{S}$

$t = 0$, $i = 0$
initialize and evaluate $P(t)$
while *termination criterion* $\mathcal{T}$ *not met* **do**
  $t = t + 1$
  while $i < \mathcal{S}$ **do**
    for $k = 0 : N$ **do**
      $y_i(k) = 0.5 \cdot x_1(k) + 0.5 \cdot x_2(k)$
      $y_{i+1}(k) = 1.5 \cdot x_2(k) - 0.5 \cdot x_2(k)$
      $y_{i+2}(k) = -0.5 \cdot x_1(k) + 1.5 \cdot x_2(k)$
    end
    $i = i + 3$
  end
  recombine $P(t)$
  evaluate $P(t)$
end

Furthermore, García-Martínez et al. (2008) point out that one more significant advantage is implicit when designing parent-centric crossovers, i.e. they are self-adaptive. Parent-centric crossover operators define a probability distribution based on a measure of distance among the parents, hence if they are close the offsprings are expected to be so as well, whereas if they are located far away from each other, they are expected to be distributed sparsely. Implicitly, this strategy self-adapts its behaviour (i.e. action range) by using information inherited from the parents and increase/reduce diversity accordingly, leading to a more diverse population or conversely to more refined offsprings in later stages of the search process. Moreover, these self-adaptive operators are typically tunable by means of 'strategy parameters' (Beyer and Deb (2001)): the authors show that these operators are able to adapt to a variety of fitness landscapes.

- **Flat crossover [FX]**

  The flat crossover, proposed by Radcliffe (1991), generates the offsprings from a uniform distribution, tweaking the classical crossover techniques in order to improve the search process in terms of exploration, since it extends the search process to every possible value contained in a uniform distribution bounded as follows, for a given k-th gene of a generic offspring y_i:

  $$y_i(k) = U[\min(x_a(k), x_b(k)), \max(x_a(k), x_b(k))] \tag{3.13}$$

  Still, note that the offspring is generated from a uniform distribution bounded by the parents genes, which actually is not that beneficial for exploration purposes, according to the framework sketched in Herrera et al. (2005); Radcliffe (1991) finds out that RCGA with flat crossover performs better than traditional crossover techniques on De Jong's test suite.

- **Blend crossover [BLX]**

  The blend crossover, proposed by Eshelman and Schaffer (1993), deals with the limited search capabilities of the flat crossover by adding a parameter α whose aim is to widen or to shrink the search domain. Given two parents x_a and x_b, then the $k-th$ gene of an offspring is drawn from a random uniform distribution,

Algorithm 18: Direction-based and Heuristic crossover

Input : $\mathcal{T}$: termination criterion
$P(\cdot)$: population of size $\mathcal{S}$

$t = 0$, $i = 0$
initialize and evaluate $P(t)$
while *termination criterion* $\mathcal{T}$ *not met* **do**
  $t = t + 1$
  while $i < \mathcal{S}$ **do**
    if $f(x_b) < f(x_a)$ **then**
      $y_1 = r(x_b - x_a) + x_b$
    else
      $y_1 = r(x_a - x_b) + x_a$
    end
    if *'heuristic' and 'unfeasible'* **then**
      draw r again up to w times
      $i = i + 1$
    end
  end
  recombine $P(t)$
  evaluate $P(t)$
end

as follows:

$$x_{min}(k) = \min(x_a(k), x_b(k)), \quad x_{max}(k) = \max(x_a(k), x_b(k))$$
$$\Delta_x = x_{max} - x_{min} \tag{3.14}$$

Then we have:

$$y_i(k) = U[x_{min}(k) - \alpha\Delta_x, x_{max}(k) + \alpha\Delta_x] \tag{3.15}$$

As highlighted by Herrera et al. (2005), a negative alpha encourages exploitation, whereas a positive alpha tilts the crossover operator towards exploration; note that the blend crossover is a simple generalization of the flat one: indeed, for $\alpha = 0$, they are equivalent.

Many articles in the past twenty years have investigated some useful characteristics of the blend crossover; in general, it could be considered as a pioneering operator in the mean-centric crossover family of crossover strategies, satisfying most of its properties identified in literature over time (actually, the taxonomy presented here is more recent and so the crossover design principles). Eshelman and Schaffer (1993) suggest $\alpha = 0.5$ because, they argue, the probability that an offspring lies outside its parents is equal to to the probability that it lies between his/her parents; furthermore, they stress the fact that for that value of α the convergent/divergent tendecies are balanced.

From a different standpoint Deb and Beyer (2001) discuss a compelling proposal for the design of crossover operators; indeed, the argue, a mean-centric operator, to implement a truly self-adaptive recombination, should respect the following properties:

- The extent of the offspring solutions is proportional to the parent solutions;
- The offsprings generated in close proximity to parents are monotonically more likely to be chosen as children solutions than those distant from parents.

The gist of the above mentioned ideas, especially the latter, is to assign a bias to certain solutions in order to implement a self-adaptive crossover strategy.

An interesting property of the blend crossover is the one highlighted in equation 3.14: if the difference in the parent is small, the difference between offsprings and parents is small as well. In Deb and Beyer (2001) this property is deemed crucial because the spread of current population has an impact on the following one, i.e. the offspring solutions are somewhat proportional to the parents solutions, implementing a simple and basic version of self adaptivity.

Though the blend crossover fulfills the first property, the second one is not fully satisfied, because offsprings within a certain distance from parents are favored, but no bias is attached to new solutions (i.e. a fixed probability is assigned to all solutions near parents); as a consequence, they claim that generating uniformly distributed offsprings in the proximity of parents does not fulfill completely the principles behind self-adaption of crossover operators. The tests carried out in the final part of the paper tend to confirm this intuition.

Algorithm 19: Flat and Blend crossover

Input : $\mathcal{T}$: termination criterion
$P(\cdot)$: population of size $\mathcal{S}$
α: a constant value $c \in [0, 1]$

$t = 0$, $i = 0$
initialize and evaluate $P(t)$
while *termination criterion* $\mathcal{T}$ *not met* **do**
  $t = t + 1$
  while $i < \mathcal{S}$ **do**
    select two parents x_a and x_b from $P(t-1)$
    if *'flat'==True* **then**
      $\alpha = 0$
    end
    for $k = 0 : N$ **do**
      $x_{min}(k) = \min(x_a(k), x_b(k)), \quad x_{max}(k) = \max(x_a(k), x_b(k))$
      $\Delta_x = x_{max} - x_{min}$
      $y_i(k) = U[x_{min}(k) - \alpha\Delta_x, x_{max}(k) + \alpha\Delta_x]$
    end
    $i = i + 1$
  end
  recombine $P(t)$
  evaluate $P(t)$
end

- **Simulated binary crossover and fuzzy recombination [SBX] and [FR]**

  The simulated binary crossover, proposed by Deb et al. (1995), generates two vectors of offsprings from a probability distribution, which is dependent on the location of their parents; in other terms, it displays self-adaptive features. Fuzzy recombination is based on a similar hypothesis and it can be derived from the simulated binary crossover, so we discuss their properties jointly. Essentially, FR is only different in that it is based on a triangular probability density. For our discussion we refer mainly to Beyer and Deb (2001).

  The simulated binary crossover is defined as:

$$y_1 = 0.5[(1 - \beta)x_a + (1 + \beta)x_b] \tag{3.16}$$

$$y_2 = 0.5[(1 + \beta)x_a + (1 - \beta)x_b] \tag{3.17}$$

x_a and x_b are independent samples from the population of parents and β is a sample from a random number generator with density:

$$p(\beta) = \begin{cases} 0.5(n+1)\beta^n & \beta \leq 1 \\ 0.5(n+1)\frac{1}{\beta^{n+2}} & \beta > 1 \end{cases}$$

This distribution can be obtained by sampling from a random uniform $U[0,1]$ and then by the following transformation:

$$\beta = \begin{cases} (2u)^{\frac{1}{n+1}} & \text{if } U(0,1) \leq 0.5 \\ [2(1-u)]^{\frac{-1}{(n+1)}} & \text{otherwise} \end{cases}$$

Here we want to highlight that the blend crossover and the fuzzy recombination are special cases of the simulated binary strategy.

For what concerns blend crossover, with a simple reformulation of equation 3.16 we define:

$$\beta = 2\xi - 1 \tag{3.18}$$

applied to 3.16 and 3.17 we obtain:

$$y_1 = (1-\xi)x_a + \xi x_b \tag{3.19}$$

$$y_2 = \xi x_a + (1-\xi)x_b \tag{3.20}$$

Hence ξ is sampled uniformly from $U[-\alpha, 1+\alpha]$, which is the Eshelman and Schaffer (1993)'s blend crossover.

The fuzzy recombination, proposed by Voigt et al. (1995), is very similar to the simulated binary crossover. The density of β has its maximum at $\beta = 1$. while the only difference between them is the shape of the probability density $p(\beta)$, which is triangular for the fuzzy crossover:

$$p_\Delta(\zeta) = \begin{cases} \zeta + 1 & \text{if } -1 \leq \zeta < 1 \\ 1 - \zeta & \text{if } -0 \leq \zeta \leq 1 \end{cases}$$

The β value in 3.16 and 3.17 is then obtained as follows:

$$\beta = 1 + 2\zeta d \tag{3.21}$$

As Beyer and Deb (2001) suggest, d is a 'strategy parameter', which determines how far the offspring should be located from parents. The random number ζ with triangular distribution can be obtained as the sum of two independent numbers $U[0,1]$:

$$\zeta = u_1 + u_2 - 1 \tag{3.22}$$

The p.d.f. (whether $p(\beta)$ or $p_\Delta(\zeta)$) has a non-zero mean and a maximum at $\beta = 1$, whereas for the blend crossover the distribution has a zero mean and it is uniform. This point is definitely crucial: we have mentioned previously the self-adaptive property of certain parent and mean centric operators, which is broadly considered useful and attractive (Kita (2001), García-Martínez et al. (2008)), as it allows to displace offspring on the basis of a measure of distance among parents, which is in turn helpful to define a probability distribution.

Recall that blend crossover generates offsprings on the basis of a distance measure (1), but the probability of creating solutions near parents (2) is fixed, which is considered a main flaw by Deb and Beyer (2001), who argue that near parents solutions should be more likely to be chosen than those that are distant; note that blend and fuzzy crossover satisfy both properties (1) and (2).

Algorithm 20: Simulated binary crossover

Input : $\mathcal{T}$: termination criterion
$P(\cdot)$: population of size $\mathcal{S}$
d: tunable parameter

$t = 0$, $i = 0$
initialize and evaluate $P(t)$
while *termination criterion* $\mathcal{T}$ *not met* **do**
  $t = t + 1$
  while $i < \mathcal{S}$ **do**
    select two parents x_a and x_b from $P(t-1)$
    for $k = 0 : N$ **do**
      $x_{min}(k) = \min(x_a(k), x_b(k)), \quad x_{max}(k) = \max(x_a(k), x_b(k))$
      $\Delta_x = x_{max} - x_{min}$
      $y_i(k) = U[x_{min}(k) - \alpha\Delta_x, x_{max}(k) + \alpha\Delta_x]$
    end
    $y_i = 0.5[(1-\beta)x_a + (1+\beta)x_b]$
    $y_{i+1} = 0.5[(1+\beta)x_a + (1-\beta)x_b]$
    $i = i + 2$
  end
  recombine $P(t)$
  evaluate $P(t)$
end

- **Laplace crossover [LX]**

  The Laplace crossover, proposed by Deep and Thakur (2007), is part of the family of parent-centric crossover operators, sharing many characteristics with the simulated binary crossover. The density function of the Laplace distribution is given by:

$$f(x) = \frac{1}{2b} exp\left(-\frac{|x-a|}{b}\right) \qquad -\infty < x < \infty \tag{3.23}$$

  where $a \in \mathbb{R}$ is the location parameter and $b > 0$ is the scale parameter. For $b = 0.5$ the probability of creating offsprings near the parents is higher and for b=1 distant points are likely to be selected as offsprings. Moreover, the authors proceed by drawing a random number $u \in \mathbb{R}$ from $U[0,1]$; then a random number β is generated from the Laplace distribution. This can be obtained by inverting it:

$$\beta = \begin{cases} a - b\log(u) & u \leq \frac{1}{2} \\ a + b\log(u) & u > \frac{1}{2} \end{cases}$$

  Then, the offsprings are generated as follows:

$$y_1 = x_a + \beta|x_a - x_b| \tag{3.24}$$

$$y_2 = x_b + \beta|x_a - x_b| \tag{3.25}$$

  The idea behind this method is similar to that of other parent centric crossovers: on the one hand the operator displays a self-adaptive behaviour, just like the simulated binary crossover, being compliant with the two guidelines for self-adaptive crossover operators design (in general, if the two parents are far from each other, then the offsprings are expected to be far as well, for given values of a and b). On the other hand, there are two tunable strategy parameters (a and b), which are helpful to determine the shape of the probability density. The main implication still involves the trade-off between exploitation and exploration; one may impose for instance a small value of b to encourage the production of more diversity in the population. Otherwise, in later stages of the search process

Algorithm 21: Fuzzy crossover

Input : $\mathcal{T}$: termination criterion
$P(\cdot)$: population of size $\mathcal{S}$

```
t = 0, i = 0
initialize and evaluate P(t)
while termination criterion T not met do
    t = t + 1
    while i < S do
        select two parents x_a and x_b from P(t − 1)
        for k = 0 : N do
            % the probability that k-th gene y_i(k) is given by p(y_i(k))
            %where φ(x_a), φ(x_b) are triangular p.d.f. having modal values
              x_a, x_b
            p(y_i(k)) ∈ {φ(x_a), φ(x_b)}
            i = i + 1
        end
    end
    recombine P(t)
    evaluate P(t)
end
```

one may prefer fine-tuned solutions, which can be favoured by a convergent population, i.e. by choosing a large value of b.

- **Parent-centric normal crossover [PNX]**

  The parent-centric crossover, devised by Ballester and Carter (2004), seeks to improve the simulated binary crossover by generating offsprings in regions of the search space neglected by SBX, though it essentially fulfills the design principles outlined in Deb and Beyer (2001) and Beyer and Deb (2001); in particular, it is a parent-centric and self-adaptive operator, i.e. the spread of offspring solutions depends on the distance between parents, which decreases as the population tends to converge to a solution.

  Consequently, the parent-centric crossover aims at preserving the many useful properties of the simulated binary crossover and in the process it does not exclude any regions, supporting a more effective and extensive exploration in the search space. Furthermore, the parent-centric normal crossover is not biased towards any direction: rather, it uses an ellipsoidal probability distribution around two parents to generate two offsprings. In order to satisfy the parent-centric property is satisfied, as the offsprings are generated by means of a normal distribution centered at the parents genes. Furthermore, the self-adaptive behaviour is achieved by exploiting the shrinking distance between parents, which are used here to determine the second moment of the distribution. A tunable η parameter increases/decreases further the concentration around the parents:

$$y_1(j) = N(x_a(j), |x_b(j) - x_a(j)|/\eta) \tag{3.26}$$

$$y_2(j) = N(x_b(j), |x_b(j) - x_a(j)|/\eta) \tag{3.27}$$

  where $N(\mu, \sigma^2)$ is a normal random number, η is a tunable parameter. With j we denote as usual the $j - th$ component of parent a or b.

- **Unimodal normal distribution crossover [UNDX]**

  The unimodal normal distribution crossover, devised by Ono and Kobayashi (1999), belongs to the family of mean-centric operators and it uses an ellipsoidal

Algorithm 22: Laplace crossover

Input : $\mathcal{T}$: termination criterion
$P(\cdot)$: population of size $\mathcal{S}$
a, b: tunable parameters

$t = 0$, $i = 0$
initialize and evaluate $P(t)$
while *termination criterion* $\mathcal{T}$ *not met* **do**
  $t = t + 1$
  while $i < \mathcal{S}$ **do**
    select two parents x_a and x_b from $P(t-1)$
    if $\beta = a - b\log(u)$ **then**
    else
      $\beta = a + b\log(u)$
    end
    $y_i = x_a + \beta|x_a - x_b|$
    $y_{i+1} = x_b + \beta|x_a - x_b|$
    $i = i + 2$
  end
  recombine $P(t)$
  evaluate $P(t)$
end

probability distribution to generate offsprings. Three parents are required to run the algorithm, with two offsprings located on the line connecting two parents: two disadvantages typically arise when using this crossover operator. First, offsprings have zero probability of appearing in certain regions of the search space. Moreover, this strategy struggles to find optimal points near to the boundaries (Ono and Kobayashi (1999)). Finally, the authors and Kita et al. (1999) have shown that this approach has a self-adaptive behaviour, with the distance between the parents and the mean vector reducing progressively as the convergence in the population increases.

Basically, $(\mu - 1)$ parents are randomly chosen (say, $\mu = 3$) and their midpoint x_p is computed; thereafter, the difference vector (primary search direction) as $d = x^2 - x^1$. Furthermore a third parent x^3 is picked up randomly, and let D denote the distance between x^3 and the line connecting x^1 and x^2:

$$D = |x^3 - x^1| \times \left(1 - \left(\frac{(x^3 - x^1)^T (x^2 - x^1)}{|x^2 - x^1||x^2 - x^1|}\right)^2\right)^{1/2} \tag{3.28}$$

An offspring is generated by the equation:

$$y_1 = x_p + \xi d + \sum_{i=1}^{n-1} \eta_i e_i D \tag{3.29}$$

where ξ is a random number following a random distribution $N(0, \sigma_\xi^2)$, η_i are $n-1$ random numbers sampled from a normal distribution $N(0, \sigma_\eta^2)$ and e_i an orthogonal basis vector. Ono and Kobayashi (1999) suggest $\sigma_\xi^2 = 0.25$ and $\sigma_\eta^2 = (0.35)^2/n$ and $\mu = 3$ to 7.

Algorithm 23: Parent-centric normal crossover

Input : $\mathcal{T}$: termination criterion
$P(\cdot)$: population of size $\mathcal{S}$
η: tunable parameter

$t = 0$, $i = 0$
initialize and evaluate $P(t)$
while *termination criterion* $\mathcal{T}$ *not met* **do**
  $t = t + 1$
  while $i < \mathcal{S}$ **do**
    select two parents x_a and x_b from $P(t-1)$
    for $k = 0 : N$ **do**
      $y_i(k) = N(x_a(k), |x_b(k) - x_a(k)|/\eta)$
      $y_{i+1}(k) = N(x_b(k), |x_b(k) - x_a(k)|/\eta)$
    end
    $i = i + 2$
  end
  recombine $P(t)$
  evaluate $P(t)$
end

Algorithm 24: Unimodal normal distribution crossover

Input : $\mathcal{T}$: termination criterion
$P(\cdot)$: population of size $\mathcal{S}$
σ_ξ^2, σ_η^2: tunable parameters

$t = 0$, $i = 0$
initialize and evaluate $P(t)$
while *termination criterion* $\mathcal{T}$ *not met* **do**
  $t = t + 1$
  while $i < \mathcal{S}$ **do**
    select three parents x_a, x_b, x_c from $P(t-1)$
    choose randomly two parents (say, x_a and x_b), compute midpoint x_p
    compute the difference vector $d = x_b - x_a$
    compute the distance D between parent x_c and the line connecting x_a and x_b
    generate an offspring $y_i = x_p + \xi d + \sum_{k=1}^{n-1} \eta_i e_k D$
    %where $\xi \sim N(0, \sigma_\xi^2)$, $\eta_k \sim N(0, \sigma_\eta^2)$ and e_k an orthogonal basis vector
    $i = i + 1$
  end
  recombine $P(t)$
  evaluate $P(t)$
end

Chapter 4

Computational analysis

In this chapter, we propose a set of tests in which we evaluate the performance of a variety of crossover operators according to the following framework. We begin the analysis by evaluating in a static fashion the behaviour of each crossover operator in section 4.1. The point of that experiment is to gauge the performance of each of them in a simple experimental setting, and we investigate in particular if each operator is more inclined towards exploration or exploration. A proper classification is indeed useful to evaluate the behaviour of the controller: in section 4.2 we analyze the impact on performance of the adaptive operator selection (AOS), as it is expected to steer the search direction towards the desired direction, by picking the optimal operator for the next iteration accordingly; identyfing the nature of each operator is therefore crucial. Finally, in section 4.3 we perform an out-of-sample test, in which our goal is to gauge the performance of portfolios optimized with an EA based on the adaptive strategy; then we compare them with plain EAs equipped with various crossover operators.

4.1 Test 1: evaluating the crossover performance

The purpose of this section is to present and discuss some preliminary experimental results involving crossover operators. We aim to analyze their behaviour in a static framework, i.e. at the moment we do not consider the dynamic architecture for parameter control. Rather, we run two tests for each operator in the set: basically we observe the behaviour of twenty operators in terms of fitness and diversity, first by optimizing the fitness at each generation, then in the second one we neutralize the impact of the selection operator, which biases the search towards regions with greater fitness, in order to single out the crossover operator behaviour. The whole population is then simply kept for the next generation.

The point of this strategy is to evaluate on the one hand the overall performance of a standard GAs (from now on, with SGAs we denote any basic real-coded genetic algorithm), based on a combination of elitist selection and many different crossover strategies. On the other hand, by considering an algorithm which actually does not look for near-optimal solutions, but rather explores the search space according to a specific strategy, we evaluate solely the random search strategy implemented by the crossover.

These preliminary tests are therefore crucial to introduce the subsequent experiments involving the AOS architecture, given that any dynamic strategy based on the EvE balance necessarily requires a preparatory study of the performance of those operators and a classification of them as well, in terms of exploration, exploitation and neutrality, as proposed by di Tollo et al. (2015). This performance diagnosis will turn out to be useful when we will take into account a controller, which will be employed to implement an effective policy of operator selection. A correct interpretation of the behaviour of each recombination operator is therefore essential to measure the quality of the adopted strategy.

4.1.1 Experimental setting

First of all, a discussion of the basic structure of the SGAs is necessary: in particular, preliminary 'by-hand' tuning tests have shown that the performance of each operator is highly sensitive to 'strategic' parameters, if present. Furthermore, the impact of basic GA numeric parameters could be occasionally relevant, whereas the tests -across different instances- have shown that the choice involving the selection operator is not of particular interest, though having an impact on selective pressure. As a consequence, we opt for an elitist selection strategy, which is a very effective way to ensure that the best individuals are retained in the population from one generation to the following, i.e. by producing $(P - B)$ children at each iteration, with P the population size and B the best or elite individuals. The B best individuals in the population are directly merged with the new population. Pseudocode 25 below outlines the procedure.

Algorithm 25: Elitism

```
Compute fitness F(P) of population P
Detect best individuals B
Children=∅
t = 0
while Children< |Parents − B| do
    Choose a pair of parents for mating
    Mate the parents, generate children c1 and c2
    Children = Children ∪ {c1, c2}
    t = t + 1
end
Parents = Children ∪ B
```

The two performance criteria for the population P are the following:

$$fitness(P) = \frac{\sum_{ind \in P} eval(ind)}{|P|} \tag{4.1}$$

With $eval(ind)$ we denote the individual fitness, so that $fitness(P)$ represents a measure of average fitness.

$$entropy(P) = \frac{-\sum_{i=1}^{n} \sum_{j=0}^{k} \frac{n_{ij}}{|P|} log \frac{n_{ij}}{|P|}}{nlog2} \tag{4.2}$$

where n is the number of individuals in the population and k is the number of genes for each individual. For these experiments, we set $P = 50$, while the number of generation g is set to 1000. Finally, we consider the set of twenty crossover operators discussed in this chapter for the performance analysis.

Furthermore, we propose the following settings for the strategic parameters:

Table 4.1: Strategic parameters

	BLX-α	AX	SBX-α	LX	FR	PNX	UNDX
SP1	$\alpha = 0.5$	$\beta = 0.8$	$\mu = 0.2$	$a = 0$	$d = 0.5$	$\mu = 0.25$	$\sigma_\xi^2 = 0.25$
SP2	/	/	/	$b = 4$	/	/	$\sigma_\eta^2 = (0.35)^2/n$

We refer to the extensive discussion in section 3.1 about the importance of strategic parameters: in this context, it is enough to say that their values have basically an impact on diversity of the offspring population, by increasing or reducing the concentrarion around the parents: for instance, a small value of η for the parent-centric

normal crossover leads to a sparse children population, whereas greater values typically shrink the search space around the parents.

4.1.2 Benchmark instances and setup

Table 4.2: Problem instances

ID	Instance name	Country of origin	#	Number of observations (days)
01	Nikkei 225	Japan	210	1302
02	FTSE 100	United Kingdom	89	1306
03	Hang Seng Index	Hong Kong	42	1306
04	CAC40	France	36	1306
05	FTSEMIB	Italy	32	1397

For these preparatory tests, we have used a basic formulation of the PSP, by considering two basic constraints, i.e. budget and no-short selling constraint, that is:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \phi(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{x} \geq 0 \\ & \mathbf{x}^{\mathbf{T}}\mathbf{1} = 1 \end{aligned} \tag{4.3}$$

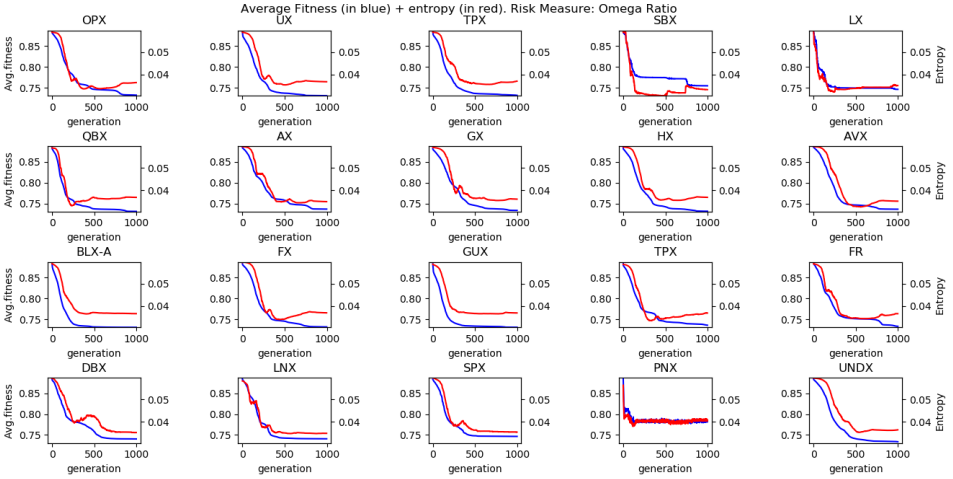
The point is to evaluate and observe the behaviour of each operator for varying datasets and risk measures, so for the moment we do not consider more complex problems, e.g. with integer constraints. In this test suite, we consider mainly non-convex and difficult objective functions, with many local minima which cannot be tackled with standard optimization techniques, like gradient-based methods (see, for instance, Gilli et al. (2011)).

When we formalize the cost function as convex combination of risk and reward, we use the risk aversion parameter $\lambda = 0.5$ (see in Table 2.1 the risk measures *Mean-Variance* and *Mean-MAD*). Moreover, for the *Two-sided* risk measure of Chen and Wang (2008) we use $a = 0.5$ and $p = 2$, while for *VaR* we consider a monthly probability $\beta = 0.05$ that losses can exceed the $VaR_{(1-\beta)}$ threshold itself. A summary of the cost functions used in the ensuing tests is proposed in Table 2.1.

4.1.3 Testing the crossover performance with the selection process

In this subsection we evaluate the performance of each crossover operator with the selection process, i.e. we consider the behaviour of each of them when an optimization process is ongoing, so that the crossover operator has basically the goal of retaining a certain amount of diversity while simultaneously the fitness is being optimized. Ideally, in this first test, we look at two main appealing qualities that an operator should have. First, as a general rule, it should 'balance' the impact of the selection operator: in particular, a good crossover strategy should mantain a reasonable amount of variance in the initial stages, when the operators are expected to focus on the exploration on certain areas of the the search space.

Furthermore, as noted by di Tollo et al. (2015), the correlation of the absolute value of fitness and entropy is generally low, i.e. they are not mutually exclusive. An operator able to increase entropy when the search is stuck in local optima is for sure a good one; the same goes for those crossovers which can manage to increase simultaneously quality and the entropy, displaying a self-adaptive behaviour. This is exactly what we expect, at least from those operators which are 'distribution-based' and allow to a certain extent some flexibility.



(a) Instance: Nikkei 225

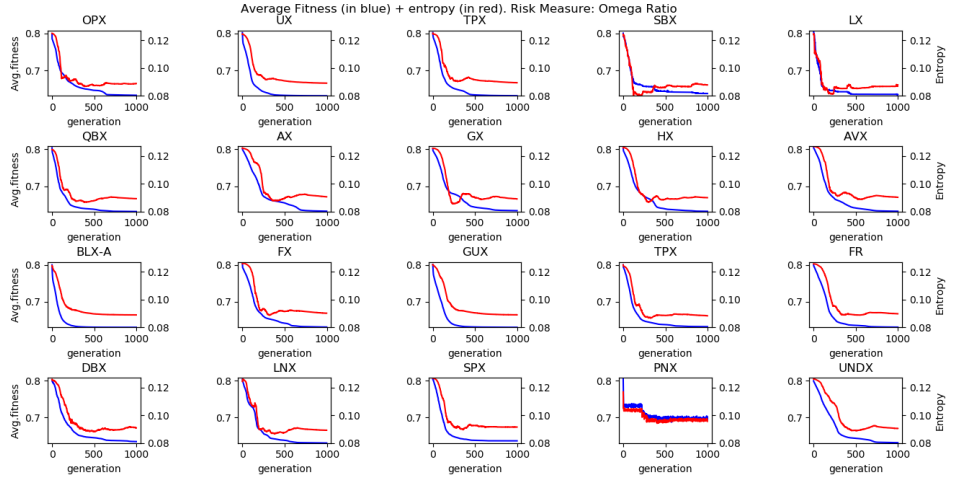
Figure 4.1

Figure 4.1 shows the convergence of the average fitness of the population and its entropy for different configurations of instances and crossovers. We focus on the most interesting combinations of risk measures and benchmarks; the objective function taken into account here is the *Omega ratio*. Due to the stochastic nature of the algorithm, the behaviour of each of them is not completely stable. Nonetheless, note that the behaviour of each operator across various instances is not homogeneous and this could for sure suggest that some operators may be able to adjust to the structure of each benchmark. Though for NFL theorem (Wolpert and Macready (1997)) the crossovers are inherently designed to display a good performance in a subset of problems (or even in certain stages of a problem) rather than others, in general, an effective operator should adjust its search bias to the structure of the problem to be solved (Herrera et al. (2005)).

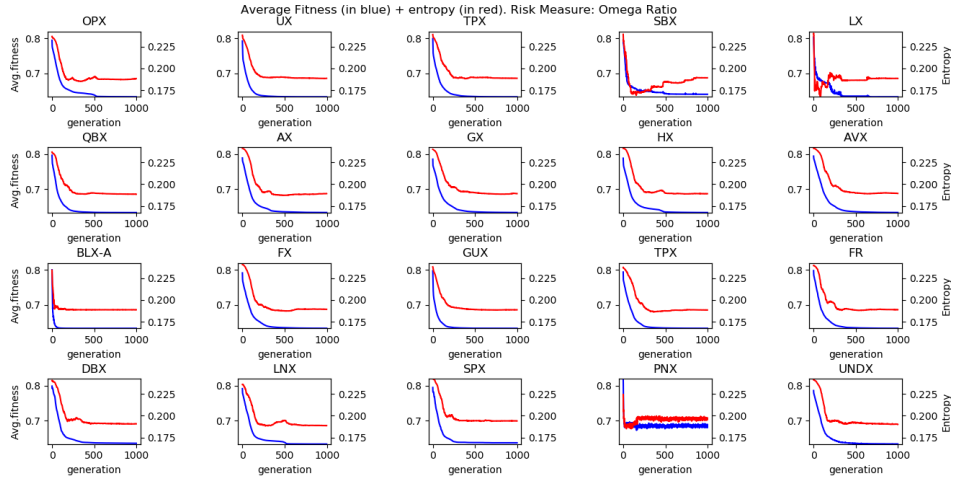
In particular, the operators belonging to the family of self-adaptive crossovers, display some very promising features. For example, note that for all the benchmarks considered, the SBX crossover and, to a certain extent, also the LX crossover, manage to reduce quickly the entropy, with a considerable and fast improvement in the average fitness. Furthermore, when improvements become harder, the operator, in order to escape local optima, seems to be turning to exploration. Though exploration is typically encouraged in the initial stage of the search, this behaviour is what we are looking for, in terms of 'adaptivity' and 'responsiveness'. In some cases, both the SBX and the LX crossover are able to increase the diversity of the population and to improve its fitness at the same time, especially in the latter stages of the search. This is consistent with the low correlation between entropy and fitness displayed by the operators discussed in di Tollo et al. (2015) and Maturana et al. (2010) for combinatorial optimization problems. Finally, note that the two operators above mentioned, in some cases, are also the best performing ones. Recall that the settings of the tunable parameters defined in Table 4.1 are tilted towards exploration.

There are three more 'distribution-based' operators which are expected to display a self-adaptive behaviour or, possibly, to perform a sort of shift between exploration/-exploitation during the search. In this sense, we highlight the slow convergence and 'saw-toothed' shape of the fitness-entropy functions computed with PNX crossover, likely determined by the very explorative setting of the strategic parameter μ (see Table 4.1), which has an impact on the variance of the normal distribution, tending in turn to produce sparse offsprings repeatedly.

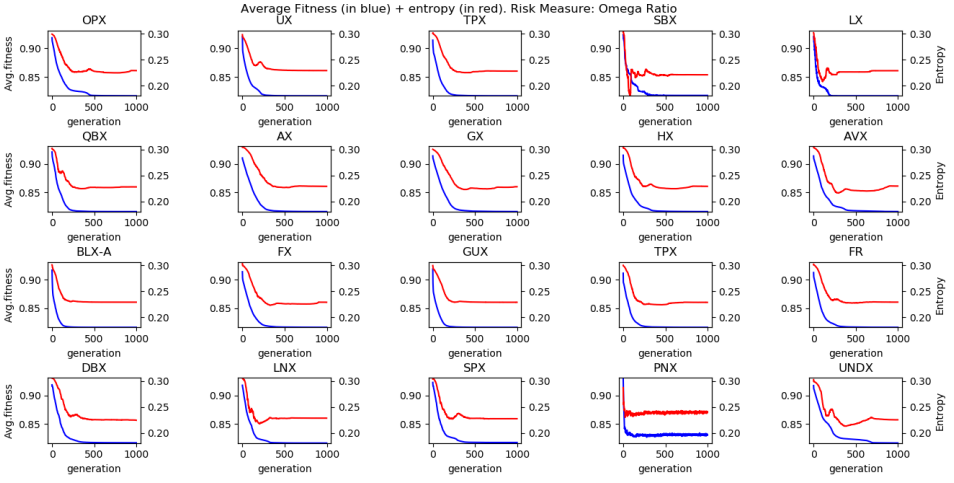
Surprisingly, fuzzy recombination (FR) seems to be enacting a pretty exploitative policy, which, after all, resembles that of the 'basic' crossover operators; basically, it



(b) Instance: FTSE 100

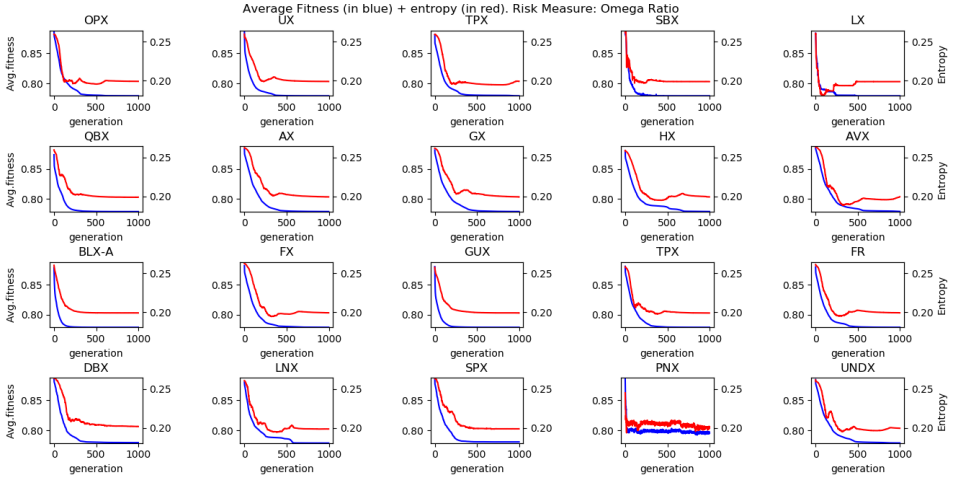


(c) Instance: Hang Seng Index



(d) Instance: FTSE MIB

Figure 4.1



(e) Instance: CAC 40

Figure 4.1: GA average fitness and entropy convergence curves

tends to reduce the diversity of the population very quickly, converging smoothly to a (potential) approximated global optimum.

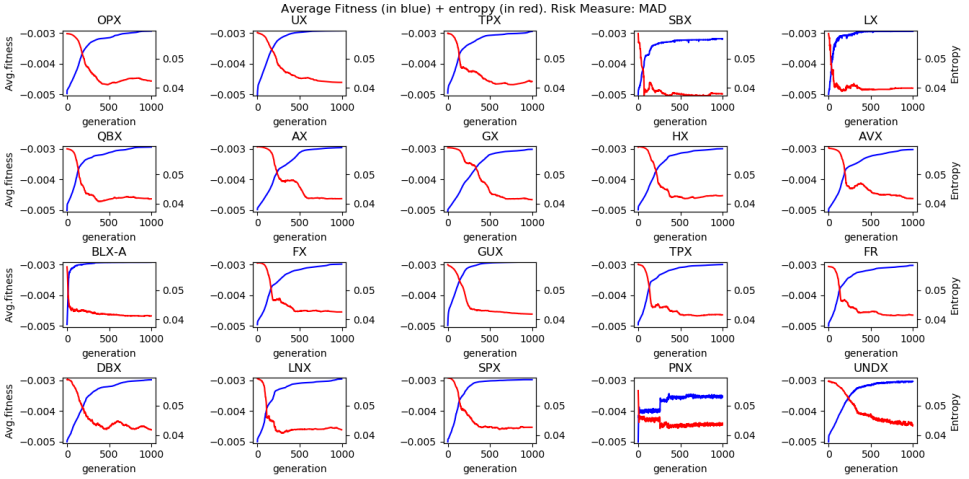
The UNDX displays a different behaviour, which seems to be affected by the given problem instance. In particular, we note that for two out of five instances (Nikkei 225 and FTSE100), it succeeds in retaining some diversity, also in the latter phases of the search; in particular, in the case of Nikkei 225 benchmark, both the fitness and the entropy function do not converge smoothly, rather they tend to show a saw-toothed behaviour in the latter stages of the process.

The FX and the BLX-A crossover, despite being parent-centric strategies, based on the hypothesis of uniformly distributed individuals (i.e. the simplest available), do not present a noteworthy performance. Preliminary tests with different settings of the tunable parameter α have shown a little impact on the behaviour of the blend crossover. This could suggest that our results are consistent with findings in the available literature (e.g. Deb et al. (2002)), which basically show that the use of a uniform p.d.f. for generating offsprings has not the same desirable properties of those strategies which use a biased p.d.f., favoring the regions represented by the parent solutions.

We comment now the operators which, by construction, are designed to use fitness information explicitly, so that they tend to give up some diversification in order to generate fitter individuals.

In this sense, the operators which include fitness information (QBX, LNX, DBX, SPX) converge steadily to an approximated solution, with both the fitness and the entropy basically flat in the last stages of the search, likely due to the fitness-based mating rule, which by design, restricts the search in a very limited region, with few chances of exploring widely the search space. As noted by Lardeux et al. (2006), an efficient crossover is not necessarily the one which quickly improves the whole population, but rather which ensures a good trade-off between quality and entropy. The point of diversification is to allow the algorithm to benefit from a better exploration of the search space, preventing the population from getting stuck in local optima.

Finally, we consider the remaining operators: most of them come from seminal contributions in literature (UX, GUX, OPX, TPX), while others have been largely used to solve constrained optimization problems (AX, GX, HX, AVX). Essentially, we confirm their exploitative nature, as they typically use the same information provided by the parents to generate new individuals or alternatively, the offsprings are created by combining numerically the values of the genes of the parents. The impact over diversity, in these cases, is very limited, as shown in this first experiment.



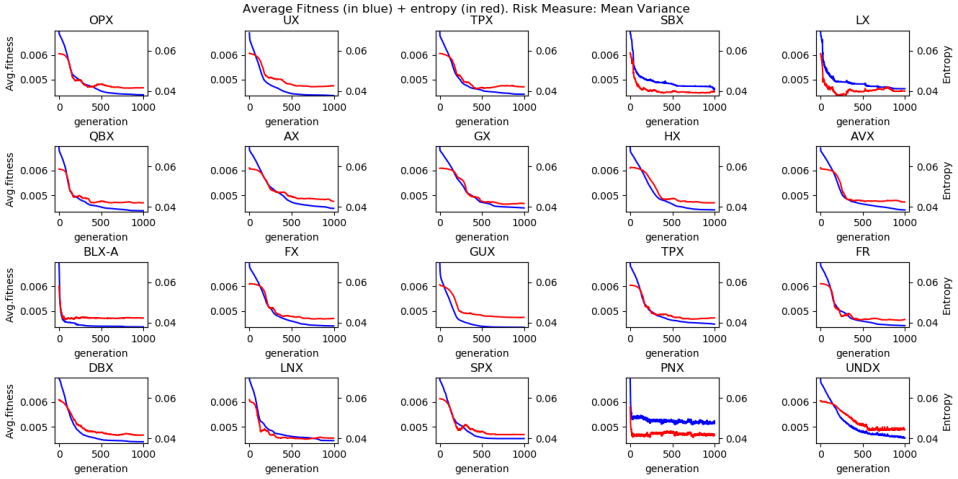
(a)

Figure 4.2

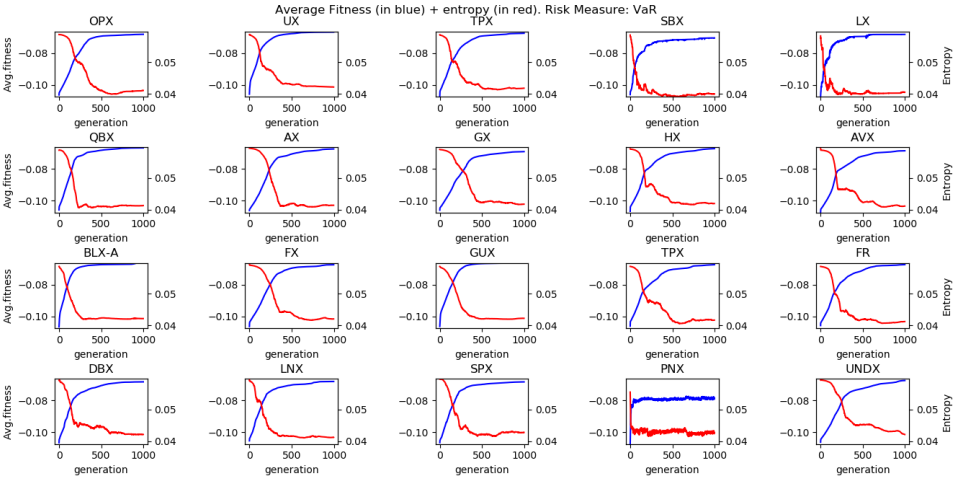
In general, the tests carried out so far seem to confirm that most of the attention in literature has been focused on exploitative strategies for a long time, with an emerging interest in more sophisticated and explorative-oriented designs in more recent literature.

Though the sample of operators discussed in this chapter and used in the tests presented above is limited, further analysis and preliminary tests (in which we have also considered certain strategies that have performed very poorly in tackling simple portfolio selection problems) have suggested that few real-parameter operators are designed for exploration purposes (see also Maturana et al. (2010) with respect to the SAT problem), therefore we suppose that most of the efforts have been put into exploitative or problem-specific crossovers, with some interest in EvE trade-off and hybrid or self-adaptive strategies arising at the end of the 20th century. In particular, the problem of developing a set of theoretical principles to design properly the crossover operators in a general framework, has been posed by e.g. Kita and Yamamura (1999) and Beyer and Deb (2001).

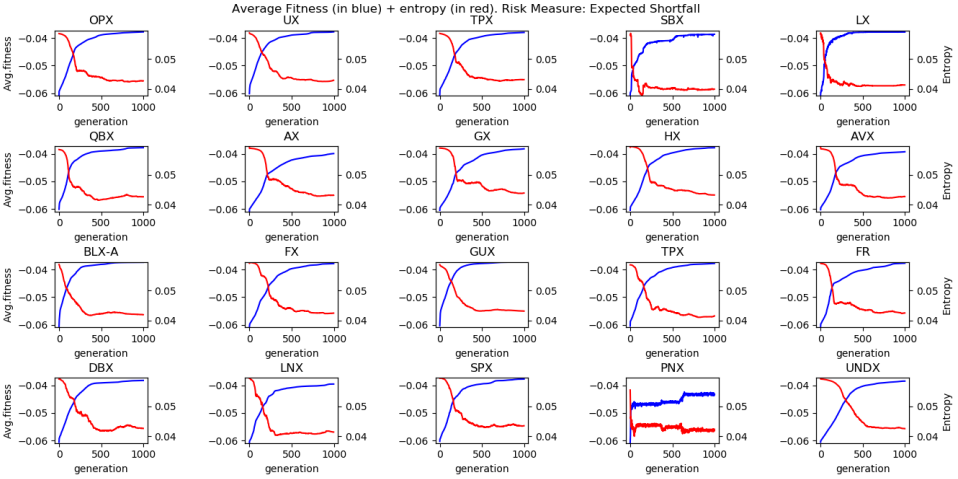
Moreover, we highlight the fact that some operators show, to a certain extent, an appealing behaviour when determining the average fitness of the Nikkei 225-based portfolios, so we consider that instance from now on, by taking into account all risk measures. In figure 4.2 we plot respectively the convergence towards near-optimal solution for all the risk measures described in Table 2.1 and the level of entropy of the population of individuals at each generation. Though the complexity of the fitness landscape is both related to the objective function and the instance considered, we conclude that the results described previously are quite robust for different risk measures, i.e. the operators show a consistent strategy across different risk measures. From this perspective, the test essentially confirms the exploitative nature of many of them, with a few exceptions. Note that in figures 4.2c and 4.2d, many operators manage to improve considerably the average fitness in the latter stages of the search; in particular, we highlight the nice behaviour of the LX crossover, which performs a sequence of 'jumps', reaching in the end a good solution. The PNX and the UNDX display a saw-toothed behaviour, with the former failing to achieve a high-quality solutions. However, they both retain a good amount of entropy and the former, in some cases, manages to improve the fitness and/or the entropy towards the end of the search. Excessive exploration induced by an operator may explain in some cases poor results, however it could turn out to be very useful when tackling problems with a dynamic AOS strategy. In general, we find the performance of the SBX, LX, PNX and UNDX crossover appealing; in particular, the latter configuration is able to retain



(b)

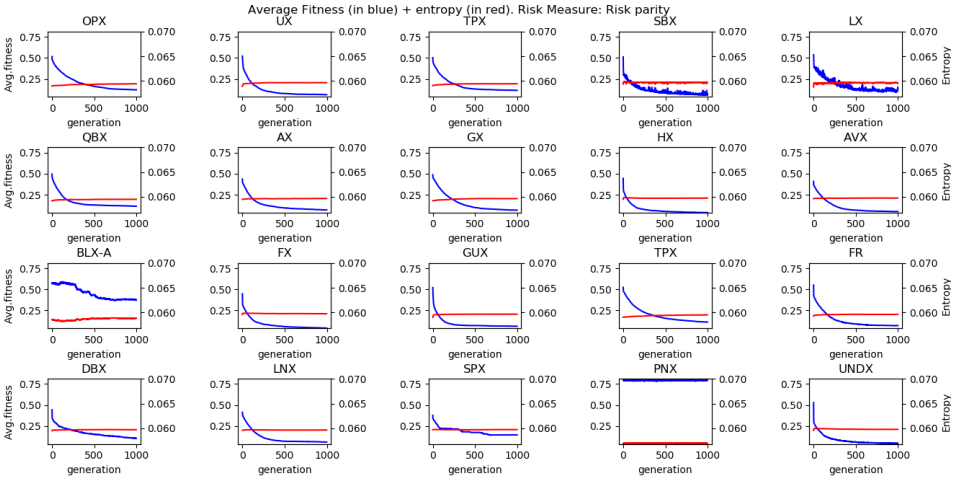


(c)

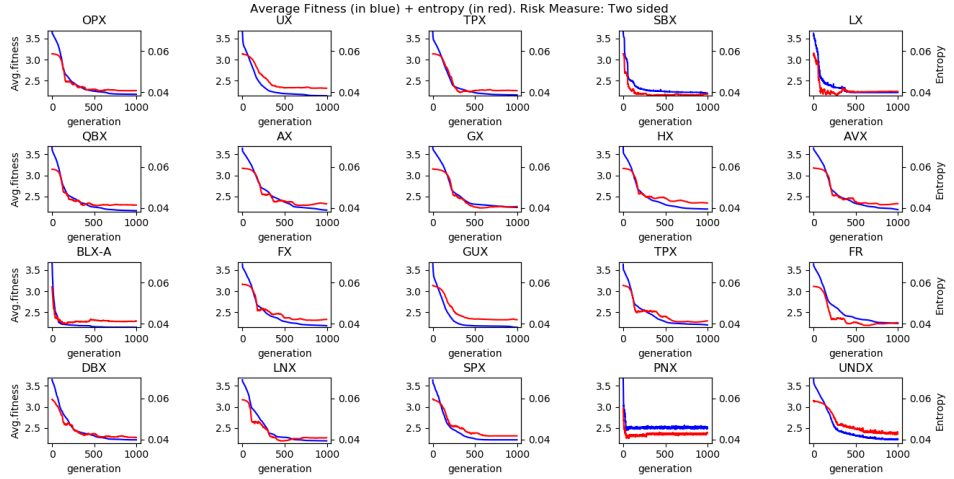


(d)

Figure 4.2

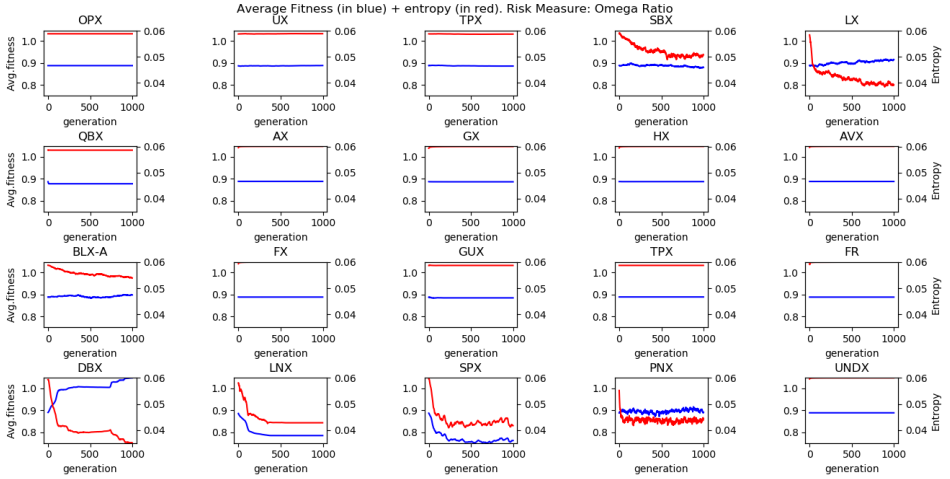


(e)



(f)

Figure 4.2: GA average fitness and entropy convergence curves. All the GAs are trained on the Nikkei 225 problem instance, for vaying cost functions.



(a) Instance: Nikkei 225

Figure 4.3

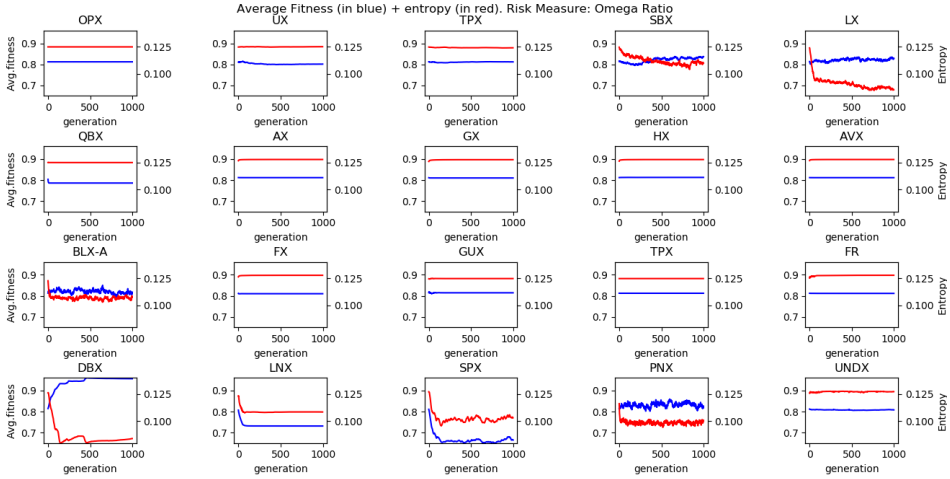
a considerable amount of diversity after generation 500 and to reach good levels of fitness; as it moves towards the final stages of the genetic process, the shape of the average fitness function and especially that of entropy tend to exhibit a less 'stable' and saw toothed shape. In the second test with different risk measures, there is little or no evidence of a strong upward trend of diversity in latter stages of the search for most operators, apart from LX and SBX, which display a weak tendency to raise diversity when the search is stagnating in local optima.

4.1.4 Testing the crossover performance without the selection process

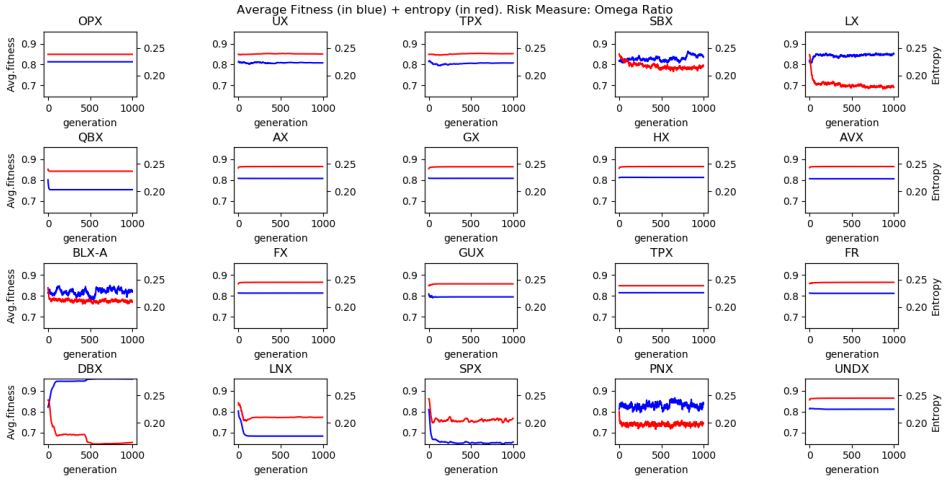
In this subsection we repeat the same experiments proposed in the previous one, by considering one by one the behaviour of each operator when only the crossover operation is performed: this means that neither selection nor mutation is allowed. The new population generated with recombination is sent to the following generation and simply selected as it is. The point of this experiment is to single out the behaviour of the crossover strategy.

In the previous section we have put our attention on the influence of the recombination on the optimization process, with the aim of showing the effectiveness of each crossover operator in striking out a balance between quality and diversity. It is well known that these two factors are strongly related (Arabas et al. (1994)): a too strong selective pressure favors the premature convergence of the EA search, while a weak selective pressure makes the search ineffective. Somehow, we have shed light on a broad variety of strategies (with some operators being too explorative or self-adaptive, others being exploitative and fast-converging): in this way, we have evaluated the dynamic interaction between the operators and in particular we have pointed out the role of the variation operator in keeping a reasonable diversity during the search.

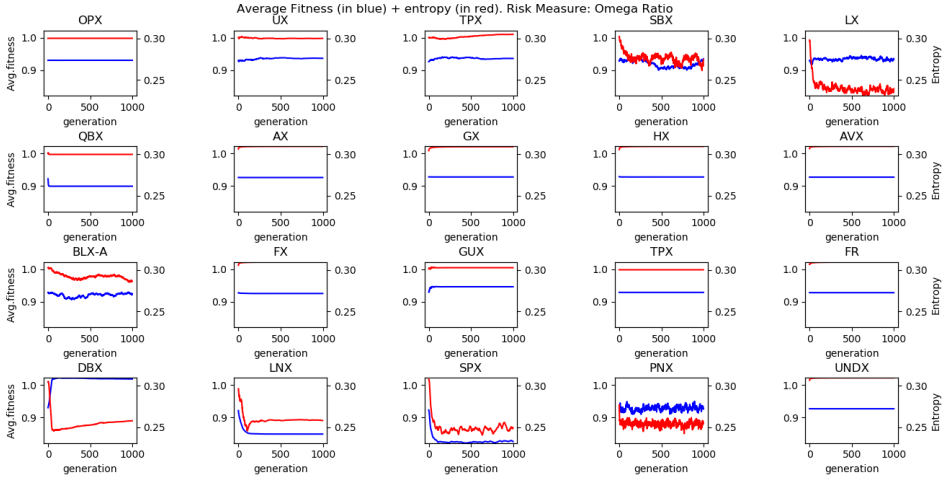
The essence of this test, instead, is to take a look at the performance of the variation operators themselves, in order to check whether some of the guidelines/design principles outlined at the beginning of this chapter are fulfilled. In particular, if the previous test has been helpful to analyze also the overall performance of the standard genetic algorithm, this one is structured exclusively to study whether the operators are able to increase the population variance; moreover, we recall that the crossover operator should not have an impact on average fitness, at least theoretically. In practical terms, we have already pointed out that a few operators using fitness information are included in the sample: they are expected to perform a sort of 'greedy' approach,



(b) Instance: FTSE 100

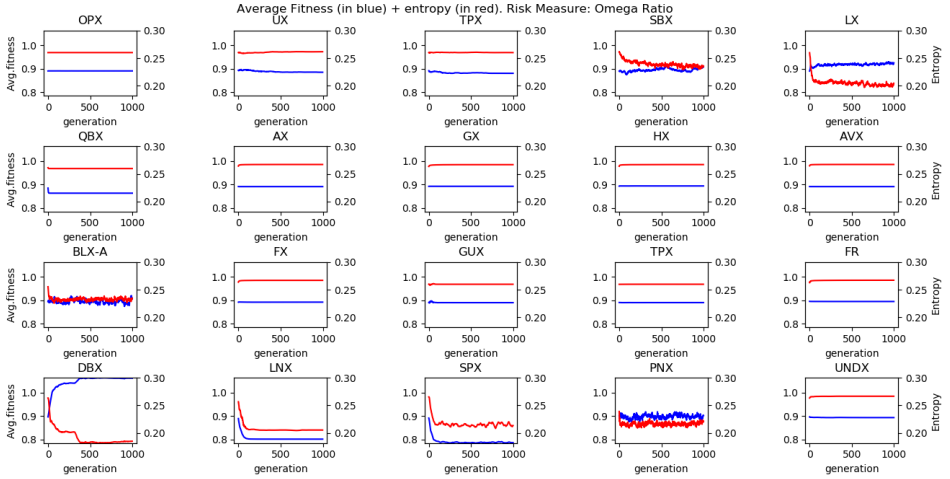


(c) Instance: Hang Seng Index



(d) Instance: FTSE MIB

Figure 4.3



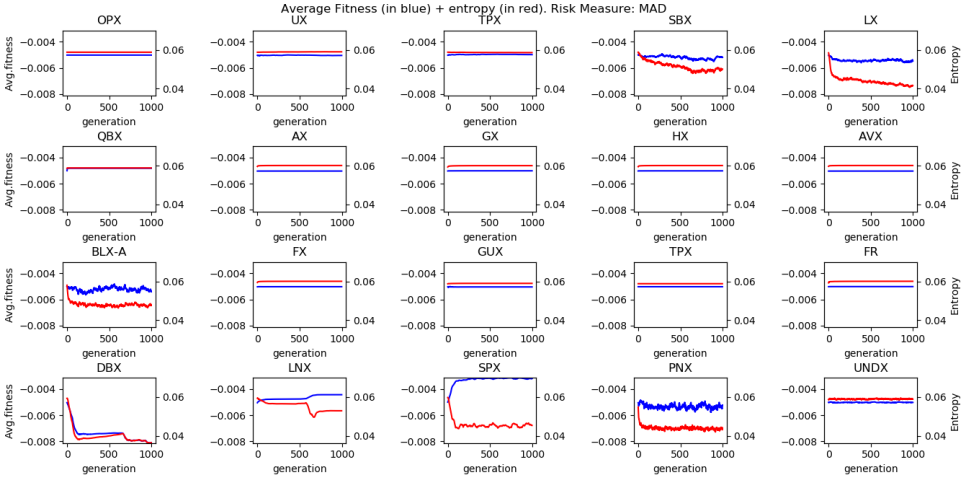
(e) Instance: CAC 40

Figure 4.3: GA average fitness and entropy convergence curves without the selection operator.

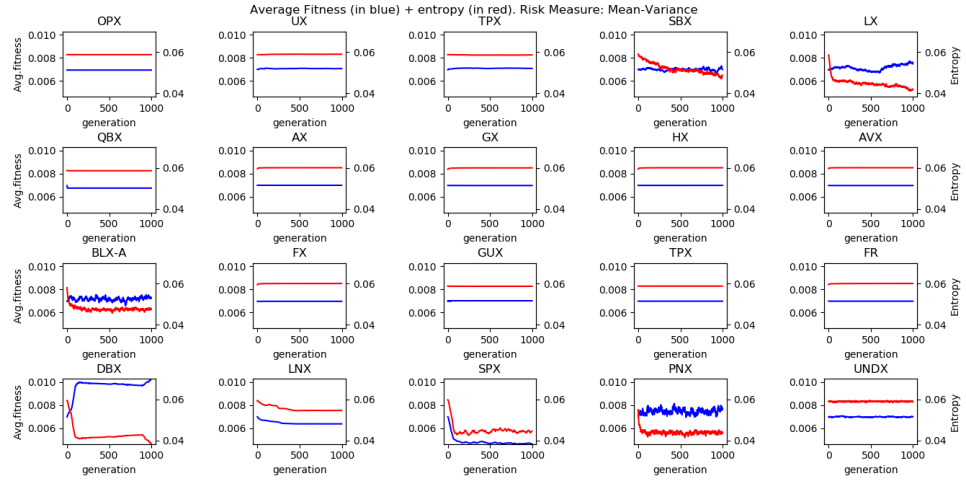
by focusing immediately on the areas of the search space that are of high fitness; as a consequence, certain regions are not visited systematically.

Essentially, each of the three categories of crossover operators manage the population with strategies radically dissimilar to each other. On the one hand the above described approaches include some fitness information: these operators should favor a consistent and stable upward trend of the average fitness. On the other hand the class of 'basic' operators is expected to have little or no impact on the population fitness and diversity, which is indeed their main weakness. As we have pointed out multiple times, the problem is that they focus the search on a limited region from the beginning, which is for sure an important shortcoming, as this is one of the main sources of premature convergence of the search process.

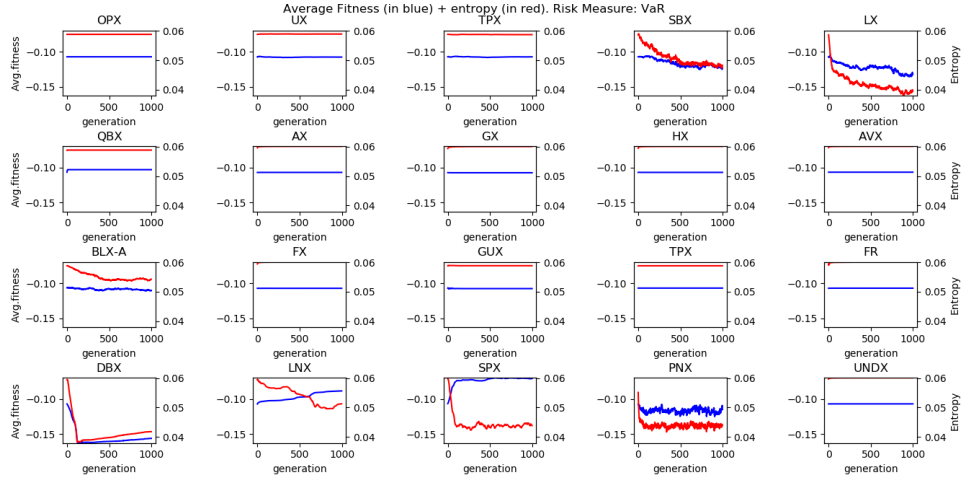
Some further experiments not presented here have confirmed this intuition: a considerable number of operators (in particular: OPX, QBX, AX, AX, GX, HX, AVX, FX, GUX) tested on different instances have shown a wide array of behaviours, but all of them have performed variations in a very limited area of the search space. As a consequence, the average fitness and the entropy tended to move in a restricted interval. UX and TPX, instead, fluctuated in a slightly wider area. Locally, some of them have displayed a kind of oscillating and rising trend of fitness (OPX, TPX, and AX in particular), others have shown -on average- moderate fluctuations in a limited area. Finally, self adaptive operators, which are typically characterized by an underlying distribution hypothesis, are expected to perform larger variations in the search space. Figure 4.3 which report the final tests run on five different instances, provide further evidence with respect to the issues we have sketched in the paragraph above. The behaviour of these twenty operators is consistent across different risk measures and instances: as before, we set the same scale for each operator, in order to favor proper comparisons (though, the 'local' behaviour of certain operators is not visible anymore). Most of them, indeed, present a basically flat entropy/a flat average fitness (in particular OPX, QBX, AX, GX, HX, AVX, BLX-A, FX, GUX), others display modest variation (UX, TPX). Two more operators (QBX and UNDX) display small variations of entropy and fitness, with the former being rather exploitative and the latter explorative. In general, LX, SBX, BLX-A and PNX crossover do not perform greatly as well: on the one hand they do not manage to keep the fitness steady and on the other hand, the entropy function shows a downward trend, whereas both fitness and entropy look almost flat in the case of the fuzzy recombination operator (FR). However, we remark that these operators at least have a tendency to visit larger



(a)

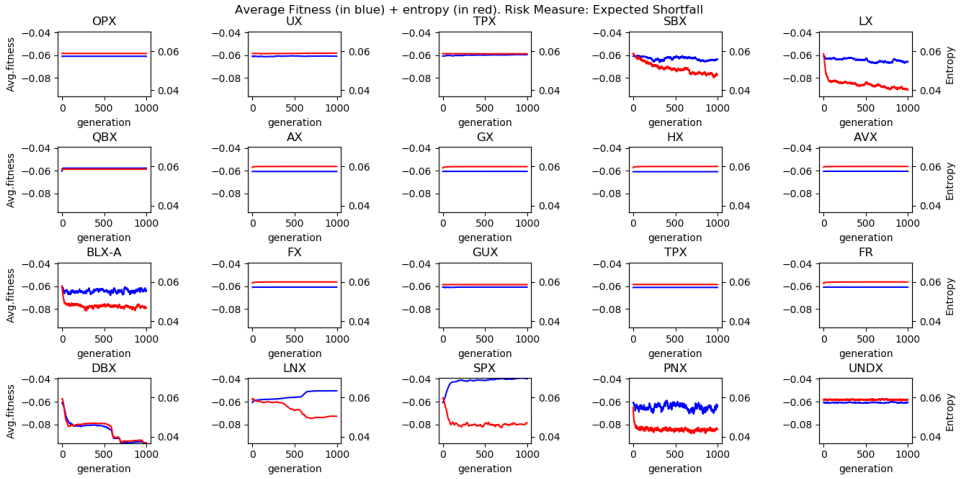


(b)

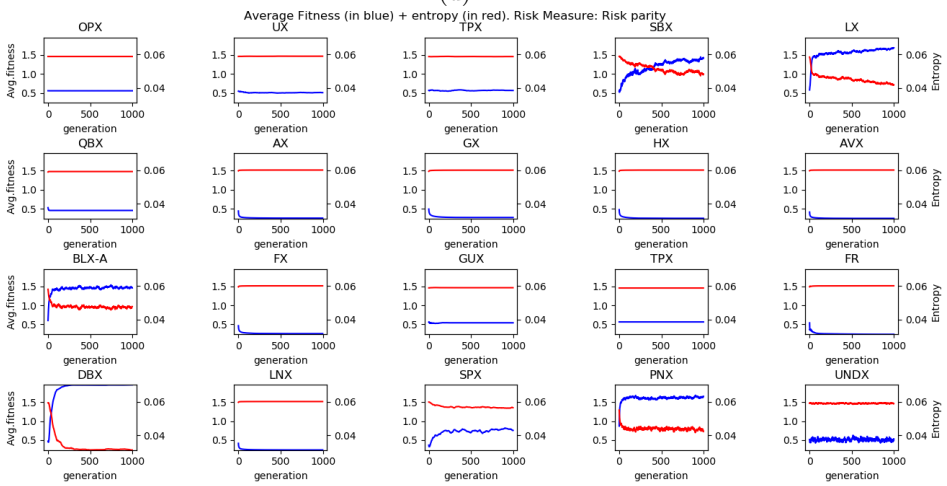


(c)

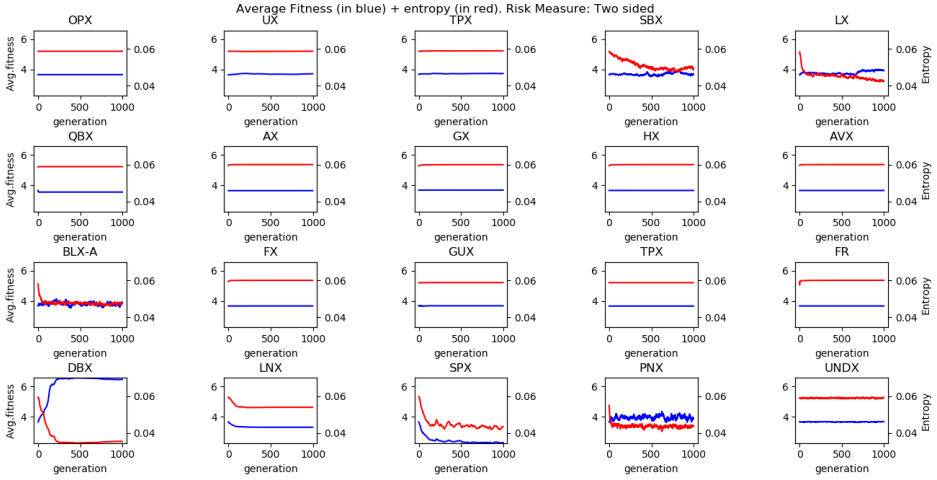
Figure 4.4



(d)



(e)



(f)

Figure 4.4: GA average fitness and entropy convergence curves without the selection operator. All the GAs are trained on the Nikkei 225 problem instance for varying cost functions.

areas in the search space, and that has actually a substantial impact on the variation of fitness and entropy. The diversity in the population, though generally declining, is likely retained by their self-adaptive property, which tends to distribute offsprings by using information inherited from parents, i.e. generally a distance measure between them. Without selective pressure, it is highly predictable that the offsprings are generated far away from each other as well. Apart from DBX, two out of four operators which include fitness information in the recombination strategy improve effectively the quality solution, in particular the simplex crossover (SPX) is able to achieve good results in terms of fitness.

Overall, the results discussed in the present subsection (4.1.4) and in subsection 4.1.3) will turn out to be useful in section 4.2, in which the behaviour of the controller is examined: in particular, the classification we have proposed above, in which we distinguish exploitation and exploration operators, is necessary to assess its actual performance and the its effectiveness in enforcing the desired search direction.

4.2 Test 2: evaluating the operators management

In this section we assess the performance of an adaptive operator selection (AOS) strategy, a dynamic approach by which we perform on-line selection of the best operator: we discuss briefly the structure of the *controller* (i.e. the architecture which tackes AOS) briefly sketched in Chapter 1 and then we present some results. In particular, we aim to address the EvE trade-off in a dynamic way: given a set of operators, the controller tries to strike a balance between exploration and exploitation at a specific stage of search. By construction, some operators manage self-adaptively the EvE balance, i.e. they are able to focus on exploitation on their own as the search evolves and solutions are closer to each other, while most of them usually put all the effort into a limited search space from scratch. This gives further evidence about the importance of adopting a high level and reward-based policy which controls rigorously the amount of quality and diversity in the population. Moreover, high level search strategies are employed to guide the search process, according to an adaptive or deterministic schedule.

Actually, the experiments carried out in the previous section back the fundamental idea of parameter control, namely it is possible to boost the algorithm performance by using multiple crossover operators to obtain better results than could be obtained from any of the constituent crossover strategies alone. In the following paragraph, an outline of an automated controller to perform this kind of process is proposed.

As a consequence, in this section we proceed by assessing the performance of the controller, i.e. in particular we evaluate the operator selection frequency and the probability of selection at different stages of search as well. Furthermore, the capability of reaching a good mix of entropy and fitness over time is also considered, in particular with respect to the high level strategy chosen.

The theoretical background of the following experiments is provided mainly by Maturana et al. (2010) with respect to the AOS framework, while di Tollo et al. (2015) provides a referring point for the implementation of high level strategies.

4.2.1 Experimental setting

First of all, recall the discussion in section 4.1 and the outline presented in Chapter 1 about adaptive operator selection: the selection of variation operators is based on a I/O interface (di Tollo et al. (2015)): the EA (solver) sends the last applied operator identifier and its performance; hence, the controller tells the EA which operator should be chosen in next iteration. We sum up now the theoretical background briefly, then we move on to practical issues, involving mainly the setting and the design of the controller.

The AOS is a generic framework to control parameters in EAs: the main idea is to implement a controller which interacts continuously with the solver, which in turn

yields useful performance measures, that is average fitness, best fitness and entropy. Hence, these quantities are sent to the AOS. The impact assessment of an operator during the search process is conveniently converted into a credit/reward measure, then stored in a credit registry, which in turn is updated according to a credit assignment scheme. Finally, a module is exclusively designated to perform operator selection, in a reinforcement learning fashion. Overall, the controller is composed of four modules, as they are proposed in di Tollo et al. (2015) (*Aggregated Criteria Computation, Reward Computation, Credit Assignment, Operator Selection*) and each of them is executed in a chronological order, as follows.

- *Aggregated Criteria Computation*: This module stores the impact of successive applications of a variation operator during the search process, i.e. the variation of the value of fitness and entropy. In particular, the values are recorded in a sliding window W_{ij} of size T, with ij denoting the operator/criterion pair. Finally, a function $F(\cdot)$ computes the final impact, which could be instantiated to *max* (to detect outliers) or *mean* if one wants to register smoother behaviours (di Tollo et al. (2015)). The input of the module are the identifier of the last applied operator and the observed variation of each criteria. The output sent to to the Reward Computation module consists in a vector contain a scalar value for each criterion (two in our case); in the following tests $F(\cdot)$ will be instantiated with $mean(\cdot)$. See Algorithm 26 for further details.

Algorithm 26: Aggregated Criteria Computation

Input : op_i: Operator i
ΔD: Observed variation of diversity criterion
ΔQ: Observed variation of quality criterion
Output: $[op_i, F(W_{i\Delta D}, T), F(W_{i\Delta Q}, T)]$

if $type = extreme$ **then**
  $F(W_{i\Delta D}, T) = max(W_{i\Delta D}, T)$
  $F(W_{i\Delta Q}, T) = max(W_{i\Delta Q}, T)$
else
  $F(W_{i\Delta D}, T) = avg(W_{i\Delta D}, T)$
  $F(W_{i\Delta Q}, T) = avg(W_{i\Delta Q}, T)$
end
if $normalize = True$ **then**
  $norm\_func = \max_{i=1,\dots,K} F(\cdot)_{i=1,\dots,K}$
  $F(\cdot) = F(\cdot)/norm\_func$
else
end

- *Reward Computation*: The reward strategy is based on the application of two modules: the first one rewards operators that can improve the fitness while simultaneously keeping a reasonable amount of diversity in the solution population (this is the so-called *Compass*, proposed by Maturana and Saubion (2008)), while the second one stores the rewards in a sliding window. The compass employs the input from the previous module (a vector) and it takes an hyperparameter $\theta \in [0, \pi/2]$, a search angle in the $\Delta D/\Delta Q$ plane. The angle could be set manually (see e.g. Maturana et al. (2009)) or it could be based on an automated dynamic strategy. Therefore, each operator is represented in this plane, according to the aggregated impact vector previously computed and associated to an additional vector, used to store rewards. The search policy is defined by θ, by which the user may favor diversity and neglect quality ($\theta = 0$), otherwise quality ($\theta = \pi/2$) may be fostered at the expense of diversity; of course, less extreme policies could be enforced. The main idea behind this is to attribute a

reward measure to each data point by computing the perpendicular distance between the performance point and the plane defined by θ. Algorithm 27 provides additional insights.

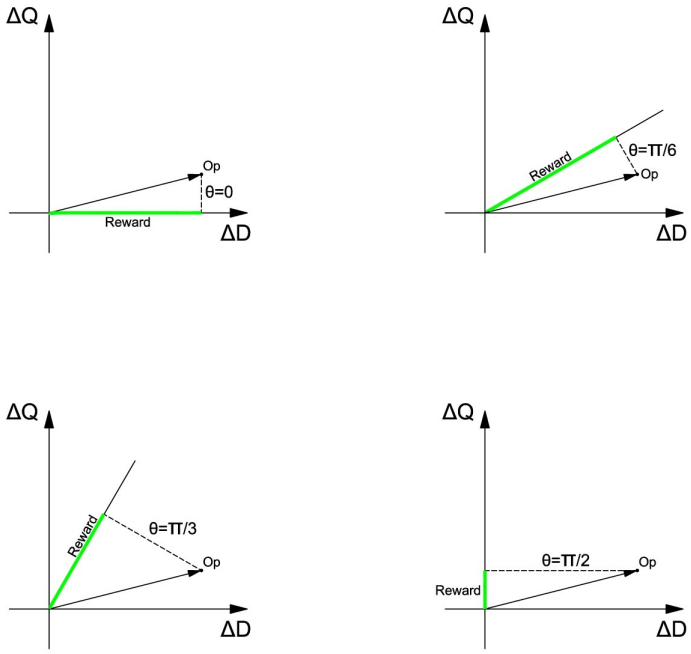


Figure 4.5: Reward computation based on a compass method, adapted from Maturana et al. (2010) and di Tollo et al. (2015)

As noted by Fialho (2010), the controller displays an interesting RL-like behaviour, as the the four main steps are iterated in a state-action-reward fashion. Furthermore, note that, before passing a credit $\mathcal{U}$ to the final module, these methods typically tend to control a balance between immediate and past reward by taking a linear combination of them, just like classic RL-techniques.

Algorithm 27: Reward Computation (Compass)

Input : $[op_i, F(W_{i\Delta D}, T), F(W_{i\Delta Q}, T)]$
Output: R_{op_i}, reward of operator op_i

$D_{op_i} = avg(diversity_{op_i})$
$Q_{op_i} = avg(quality_{op_i})$
$V_{op_i} = (D_{op_i}, Q_{op_i})$
$\alpha_{op_i} = |atan(\frac{Q_{op_i}}{D_{op_i}}) - \theta|$
$R_{op_i} = |V_{op_i} \cdot cos(\alpha_{op_i}) - min_{i=1,\dots,K}(|V_i| \cdot cos(\alpha_i)|$

- *Credit Assignment*: the credit amount summarizes the reward obtained by an operator during recent applications: it takes as input the reward of each operator op_i at time t and stores it into a sliding window of size T'. Thus, the module computes an aggregate reward by specifying an aggregation function $F(\cdot)$ (which could be instantiated to $max(\cdot)$ or $mean(\cdot)$ over the period T'. These aggregated values represent the credit, i.e. the output of the module sent to *Operator Selection*.
- *Operator Selection*: this module receives as input the credit of all operators and determines the next variation operator to be applied by the solver, according to

Algorithm 28: Credit Assignment

Input : Reward R_i
Output: Credit $\mathcal{U}_i$

if $type = extreme$ **then**
  $F'(W'_i, T', R_i) = max(W'_i, T', R_i)$
else
  $F'(W'_i, T', R_i) = avg(W'_i, T', R_i)$
end
if $normalize = True$ **then**
  $norm\_func = \max_{i=1,\dots,K} F'(\cdot)_{i=1,\dots,K}$
  $F(\cdot) = F'(\cdot)/norm\_func$
else
end
$\mathcal{U}_i = F'(W'_i, T', R_i)$

a predefined selection scheme. The input of the operator selection module is the credit estimate $\mathcal{U}_i$, while the output is the identifier op_{next}, which is sent to the EA. In the following experiment we perform selection with Probability Matching (PM), proposed in Algorithm 7, which is actually the simplest available. For a summary of selection methods, recall that in subsection 1.3.3.2 the main features of selection methods are briefly addressed. We refer the reader to the contributions of Maturana et al. (2009), Lardeux et al. (2006), Maturana et al. (2010) for a deep dive into this topic, with respect to combinatorial optimization problems.

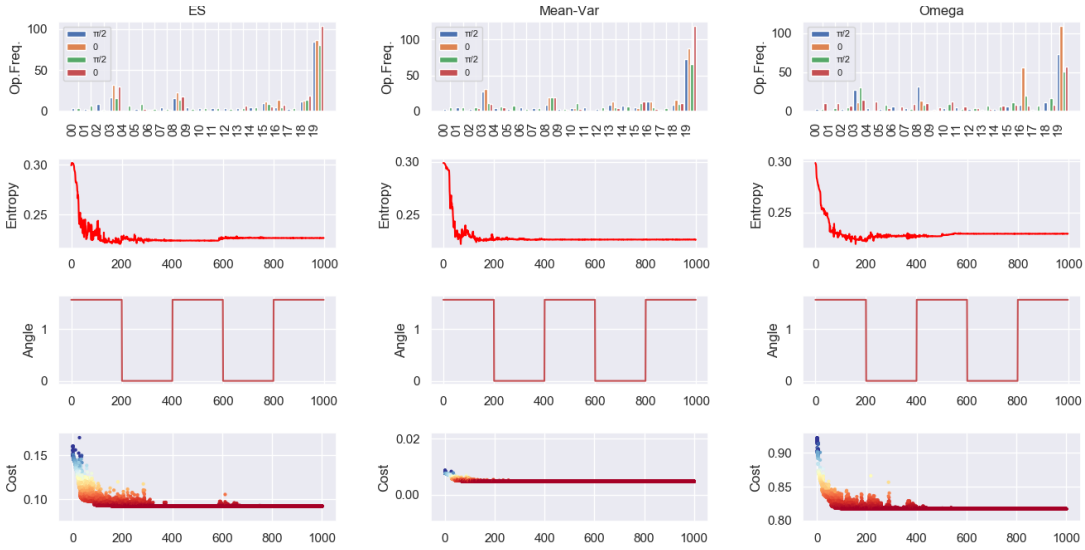
In the following experiments, we use the performance criteria described in subsection 4.1.1, namely entropy and average fitness.

As usual, the EA is stopped when 1000 iterations have been performed. The objective functions adopted in this section are ES, Mean-Var and Omega, while p_{min} in the last module is set equal to 0.01, while $K = 20$. Some other hyperparameters have to be considered. We set the rolling windows W and W' equal to 10, and as aggregation function we employ $mean(\cdot)$.

Finally, there is one more hyperparameter to be considered, namely the value of θ. To tackle this problem, we take into account the framework devised in di Tollo et al. (2015), by using four high level dynamic strategies. The schedule is either deterministic or naively self-adaptive/reactive:

- *Increasing*: Split the number of iterations into n epochs and increase the angle value in equally spaced steps in $[0, \pi/2]$;
- *Decreasing*: Split the number of iterations into n epochs and decrease the angle value in equally spaced steps in $[0, \pi/2]$;
- *Always moving*: Split the number of iterations into n epochs and alternate $\theta \in \{0, \pi/2\}$ in equally spaced steps;
- *Reactive moving*: Set $\theta = 0$ if $\frac{\Delta entropy(P,t)}{entropy(P,t-1)} < 1e-01$, else if $\frac{\Delta fitness(P,t)}{fitness(P,t-1)} < 1e-01$, set $\theta = \pi/2$. Otherwise, if neither conditions are satisfied, $\theta = 0$.

The benchmark instances used in this section are the same discussed in the previous one, for which we refer the reader to subsection 4.1.2. In particular, some preliminary tests have shown that the behaviour of the search policies is not completely homogeneous across instances: therefore, in the next subsection we will discuss the impact of high level search strategies and the pivotal role played by search policies when determining the algorithm performance. Henceforth, for space limits, we have decided not to report the results for Mean-MAD portfolios, whose behaviour is actually very similar to that of Mean-Variance portfolios.



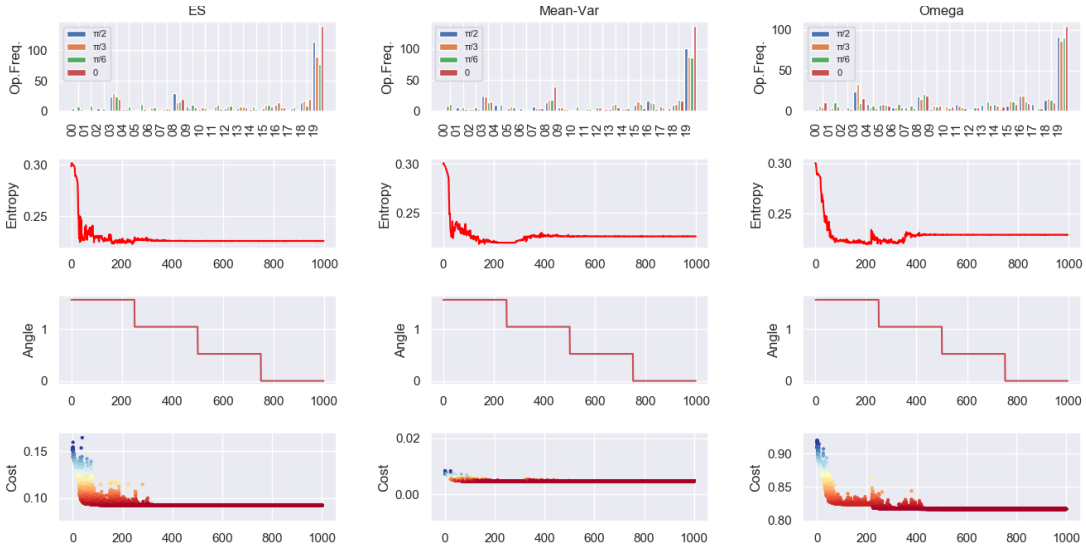
(a) Always moving strategy

Figure 4.6

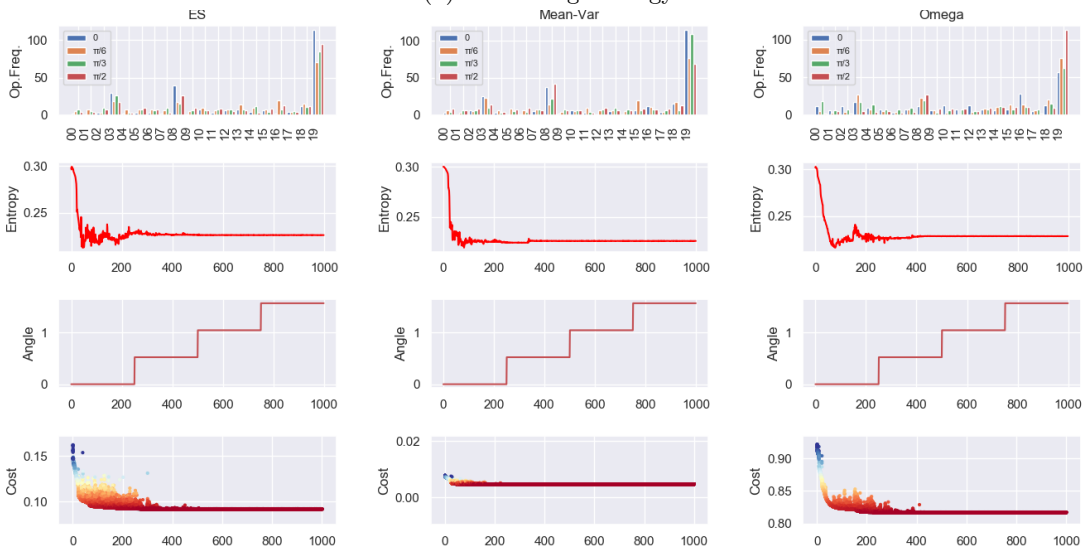
4.2.2 Testing dynamic search policies

Now we move on to consider more practical issues, as we want to evaluate the performance of dynamic search policies: the idea is to put together the results of section 4.1 and the building blocks of adaptive operator selection (AOS) introduced in Chapter 1 to improve the overall performance the algorithm. As a consequence, we look carefully at the interaction of four crucial features. First of all, recall that the goal of the controller is to strike a balance between exploration and exploitation. The extensive discussion stemming from the experimental results in section 4.1 motivate an additional test of adaptive selection of variation operators, as we have noted that certain operators display generally an explorative behaviour, while others are more tilted towards exploitation: their design of course affects deeply the performance of the algorithm. Furthermore, recall that some operators display different behaviour as a function of the state of the search, so putting them together may boost performance in terms of quality/diversity trade-off.

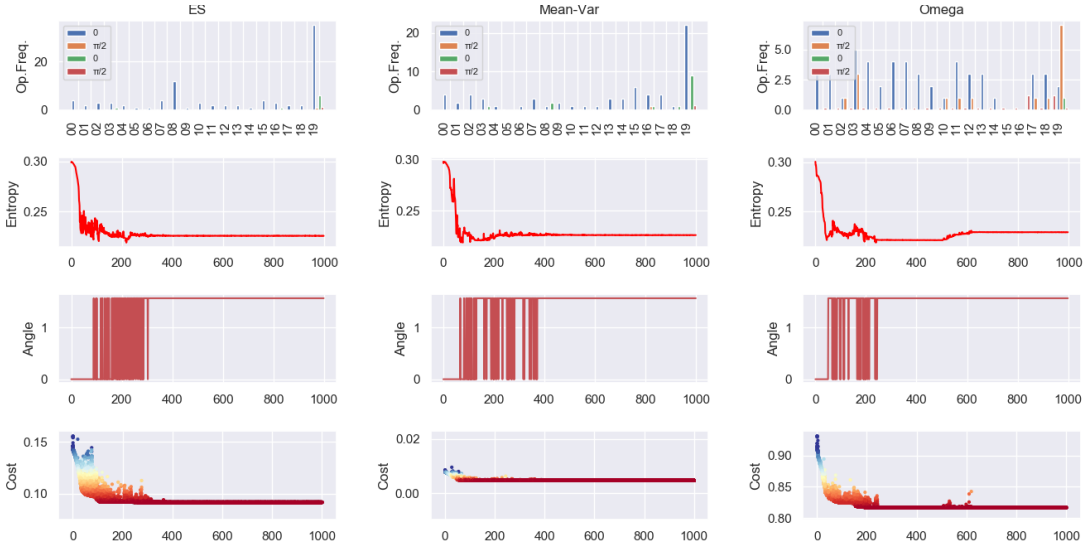
Figures 4.6a, 4.6b, 4.6c, 4.6d present the variation of individual fitness, coupled with its entropy, for changing values of $\theta \in [0, \pi/2]$: each figure reports the results with a different strategy. Moreover, the operator selection frequency histogram is meaningful, because it allows a proper evaluation of the impact of the changing angle on the operator selection process. Though the frequency is based on a highly erratic stochastic selection policy, so that less relevant operators keep being selected, actually the results are pretty much stable, so in the end we find out that the two main sources of variability in selection frequency are given by the hyperparameter θ (which fully defines a search policy) and also by the problem instances to be addressed. With respect to the FTSEMIB instance, we note that figures 4.6*b* and 4.6c display mild improvements of diversity during the search triggered by a reduction/increase in θ, especially for the Omega objective function. This also generates an immediate variation of cost for some individuals for a few generations. Overall, we note that generally exploration operators are usually selected when small or no improvements are possible, especially when they display self-adaptive features. A variation in the angle value generally leads to an important impact on selection probability and consequently on performance, though it is not always direct and straightforward; sometimes it is even tenuous and the upward trend of diversity in some cases is possibly affected by the self-adaptive nature of the operator itself. In particular, note that operators 04, 08



(b) Decreasing strategy

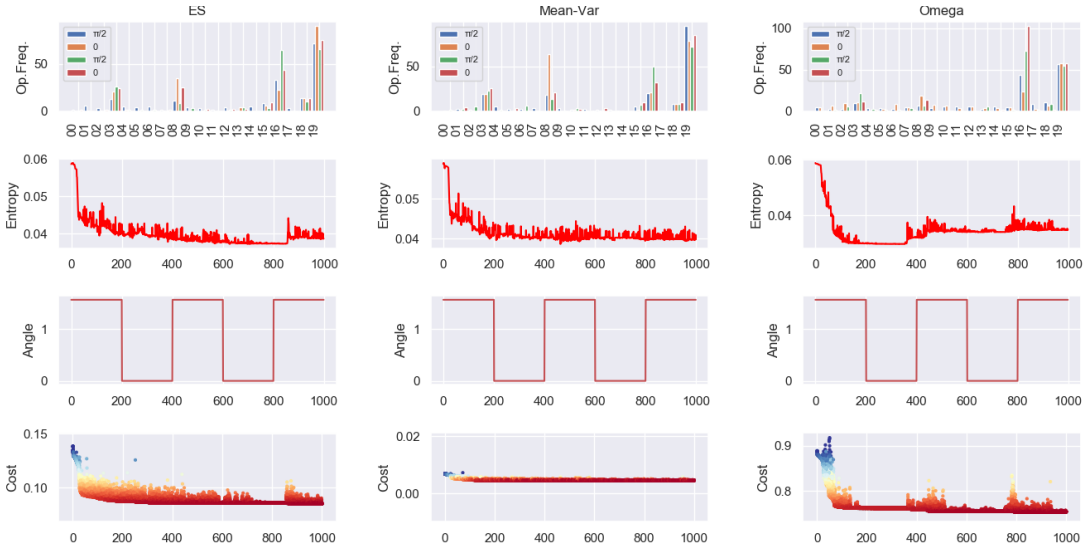


(c) Increasing strategy



(d) Reactive moving strategy

Figure 4.6: Experiments with the AOS: every portfolio selection strategy is fitted to the FTSEMIB dataset. From top to bottom: histogram of the operator selection frequency (1), the entropy convergence curve (2), the dynamic angle function driving the search process (3), the individual cost scatter plot (4).



(a) Always moving strategy

Figure 4.7

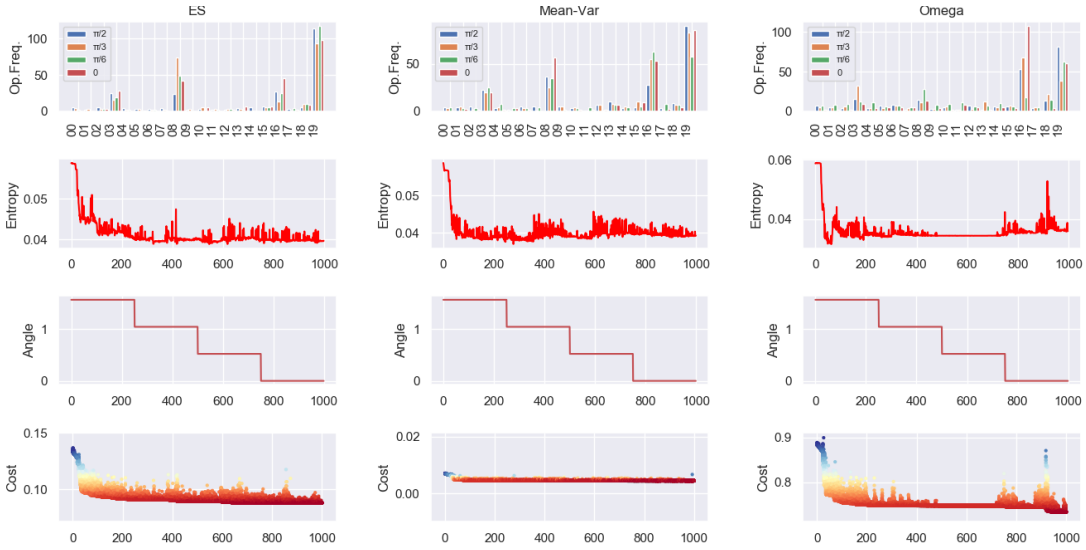
and 019 for this instance (respectively, LX, SBX and PNX) provide a good trade-off of exploration and exploitation; therefore, especially at advanced stages of search, they seem to be more effective when small or no improvements are possible. Some preliminary experiments show that the probability of selection of these exploration operators turns to be very high when the solver is stagnating, so that the controller turns to exploration, by rewarding them.

Instead, the selection probability of all mildly relevant operators tends to tumble to small values towards later stages of the process, close to the value p_{min} established before running the process, with most of the selection frequency concentrated at initial or intermediate stages of search. Furthermore, we observe in figures 4.6b and 4.6c an almost symmetrical behaviour of less relevant operators, regardless of the angle value (which still seems to have a moderate, though not regular, impact on the selection of exploration operators, whichever the stage of search); for $\theta = \pi/2$, there seems to be a more straightforward relationship between search policy and selection frequency, i.e. the controller is able to enforce the desidered direction of search.

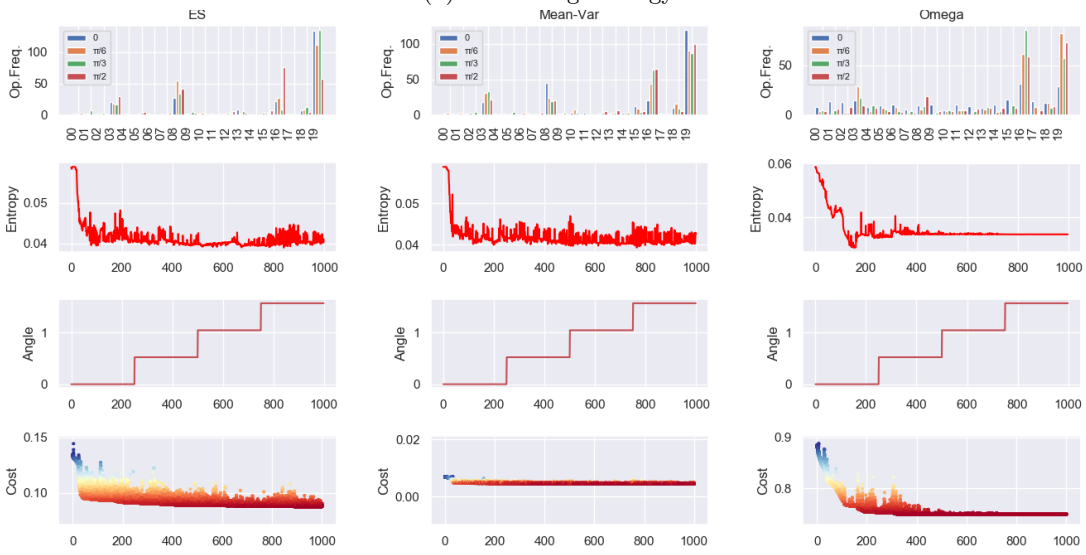
Overall, the behaviour of the controller looks quite promising, because it displays exactly an adaptive behaviour, correlated with the state of search and generally consistent with changing search policies, which seems to be able, at least in part, to steer the overall direction of search when there is small or no advancement in the optimization process, with immediate impact on quality and diversity. Unsurprisingly, the operator 16 (UNDX), which has been denoted as an explorative one (see e.g. subsection 4.1.3) has a low selection frequency. Indeed, we have previously argued that UNDX is able to strike a good balance between exploration and exploitation, though only for one instance out of five, while for other problems it has generally mimicked the exploitative behaviour of the mildly relevant variation operators.

An analysis of the results for the Nikkei 225 instance confirms that for certain problems some operators are almost neglected, while for others they stand out in a specific stage of search. Due to the role played by operator 16 (UNDX), the entropy plots take a saw-toothed shape for most of the combinations of objective functions and high level strategies. We note for instance that in figure 4.7a (see in particular the plots below 'Omega'), basically a shift to $\theta = \pi/2$ flattens both the entropy and the cost function, likely due to the fact that UNDX is not applied after generation 800.

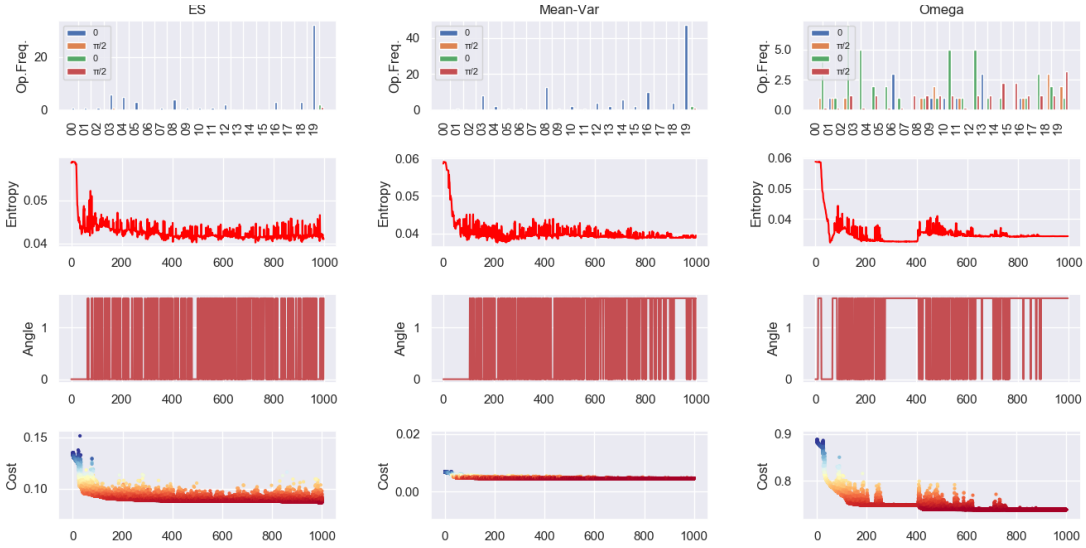
A similar behaviour can be easily detected in figure 4.7c (see in particular the



(b) Decreasing strategy

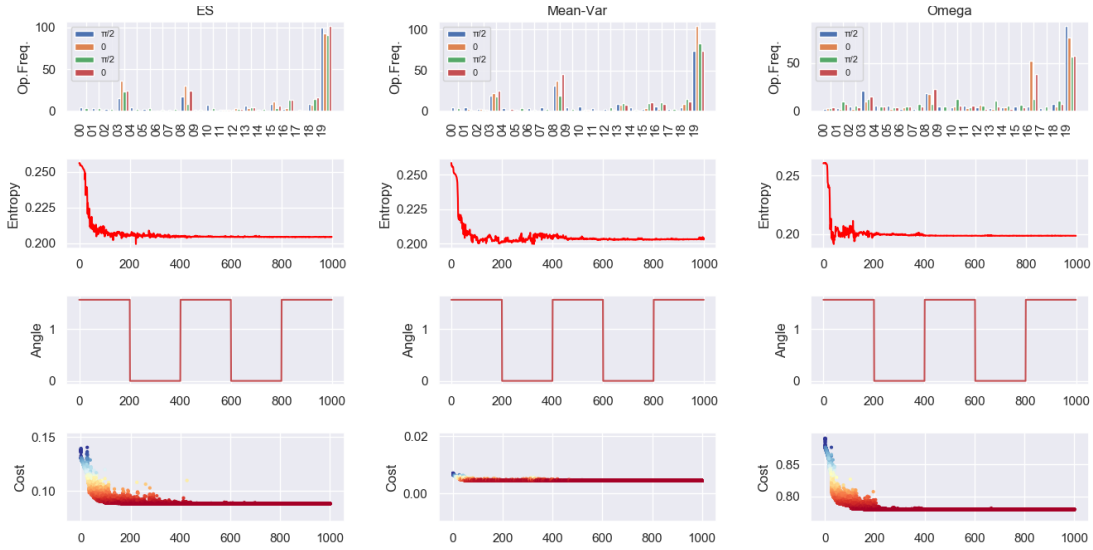


(c) Increasing strategy



(d) Reactive moving strategy

Figure 4.7: Experiments with the AOS: every portfolio selection strategy is fitted to the Nikkei 225 dataset. From top to bottom: histogram of the operator selection frequency (1), the entropy convergence curve (2), the dynamic angle function driving the search process (3), the individual cost scatter plot (4).



(a) Always moving strategy

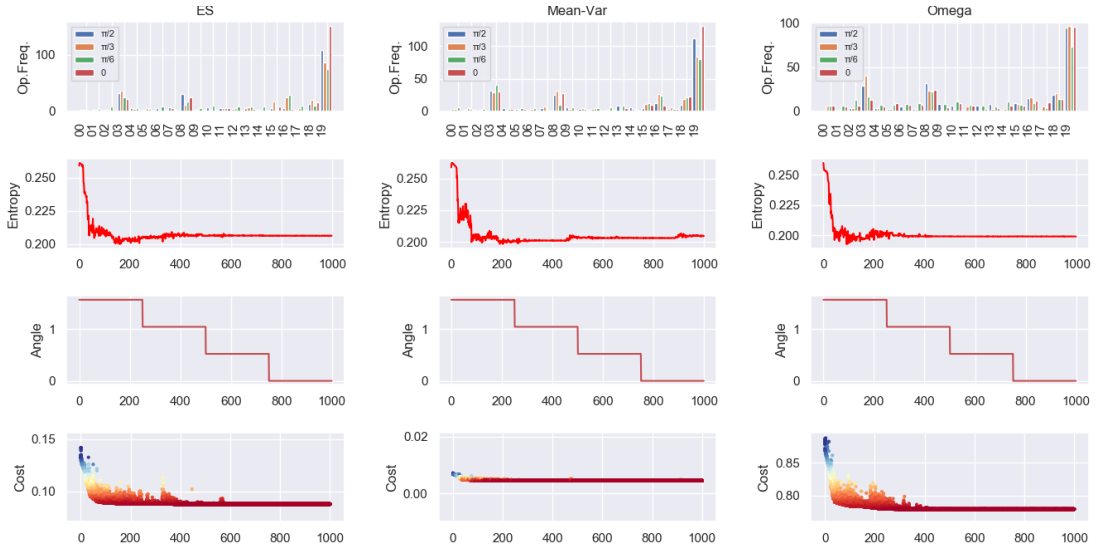
Figure 4.8

results for Omega), in which the entropy function, after tumbling to a plateau, suddenly rises to greater values at the final stage of search, leading to a small progress both in terms of best fitness and in terms of best individual cost. Looking at different risk measures, we observe in figure 4.7a and 4.7b small improvements driven either by shifts in the angle value or, more in general, by unpredictable and sudden variation in entropy, likely linked to the properties of some operators. In terms of frequency, we note again that LX, SBX and PNX seem to be among the most selected operators, due to their compelling features in terms of EvE tradeoff (as they manage to reduce cost and simultaneously they are able to retain a reasonable amount of diversity). This feature, along with swift progress at early stages of search driven by standard exploitation operators, usually leads the controller to reward the most effective operators in the set.

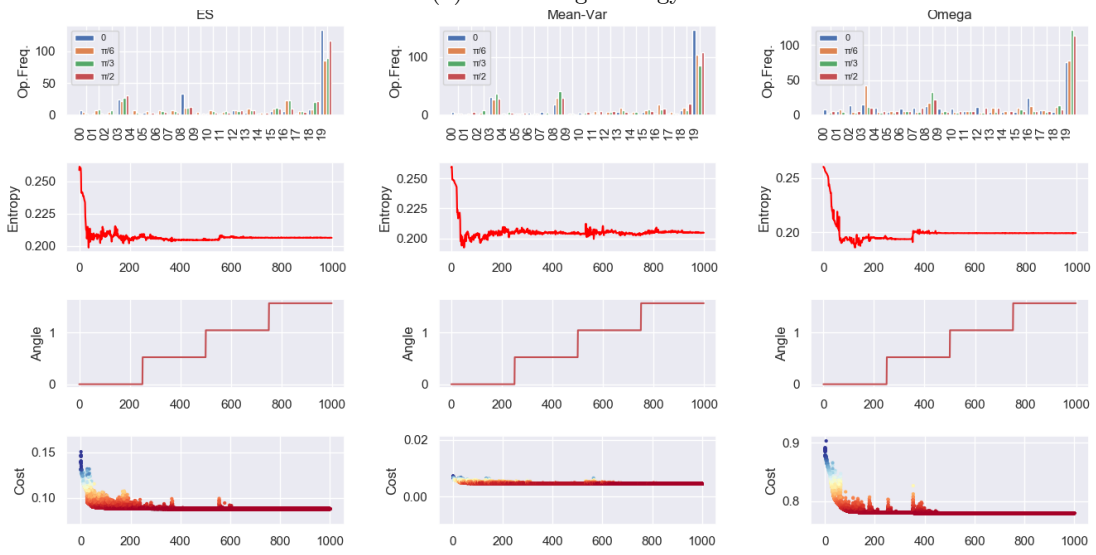
Overall, we highlight the fact that no strategy can actually able to outperform the others, so that no search policy manages to produce better results in a clear way: increasing, decreasing or always moving, which benefit from better interpretability, address the optimization problem in different ways but altogether with little impact. Moreover, the reactive policy (figure 4.7d) display a much less stable behaviour across different risk measures and consequently, as it suffers of lower predictability, it is not always that easy to discuss its performance in general, though we can comfortably state that the approach described in the previous section typically steers the controller towards an exploitative or mixed policy at the initial stage of search, then the angle tends to converge, sooner or later, to $\theta = \pi/2$.

Similar tests have been run on the remaining instances in the testbed (see appendix C), confirm that a few operators (LX, SBX, PNX) can manage optimally each problem instance at hand and typically tend to outperform the others in terms of frequency. As we have observed for the FTSEMIB dataset, for the remaining instances the operator UNDX displays a low selection frequency, which generally results in a greater amount of generations spent by the solver applying exploitation operators. Among them, we shed light on SPX (18), which shows a significant frequency of selection across different instances and angle values.

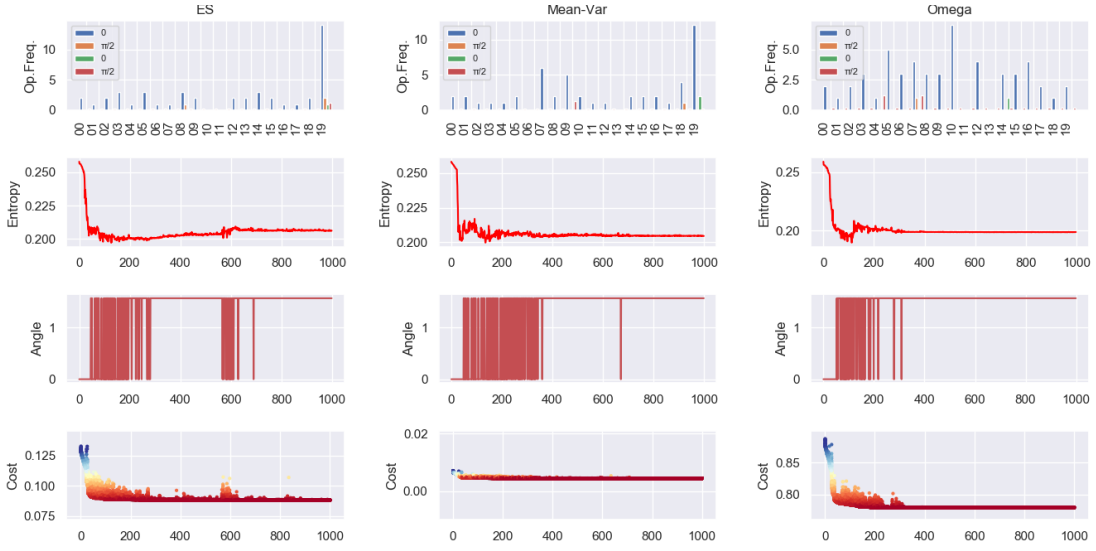
As search policies change, we do not observe any particular patterns in terms of selection frequency, as there is not an outstanding strategy in terms of performance, though in many cases it is worth mentioning a desirable behaviour stemming from a variation of the angle value (see for instance the late progress for some of the



(b) Decreasing strategy

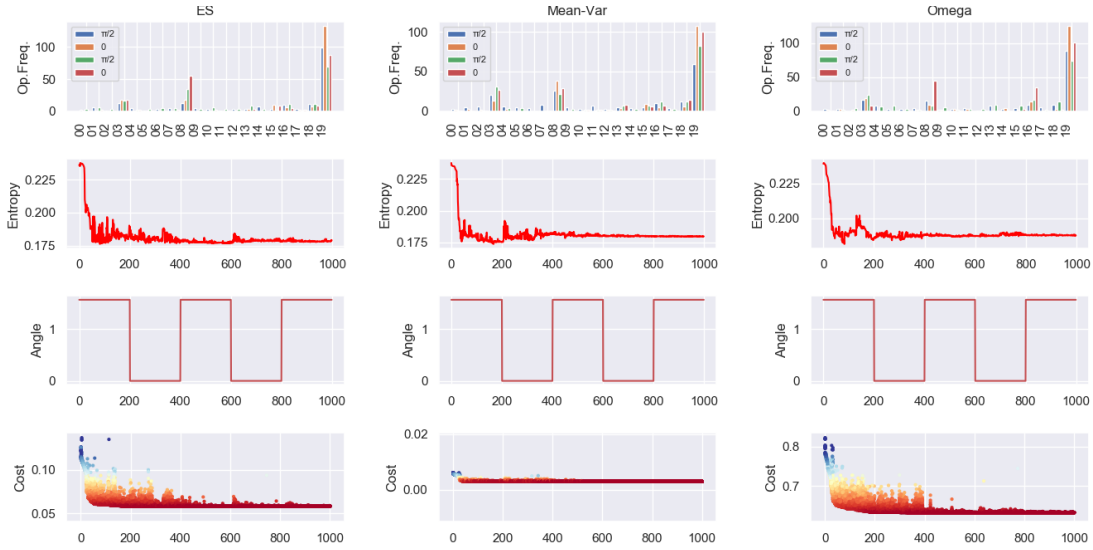


(c) Increasing strategy



(d) Reactive moving strategy

Figure 4.8: Experiments with the AOS: every portfolio selection strategy is fitted to the CAC 40 dataset. From top to bottom: histogram of the operator selection frequency (1), the entropy convergence curve (2), the dynamic angle function driving the search process (3), the individual cost scatter plot (4).



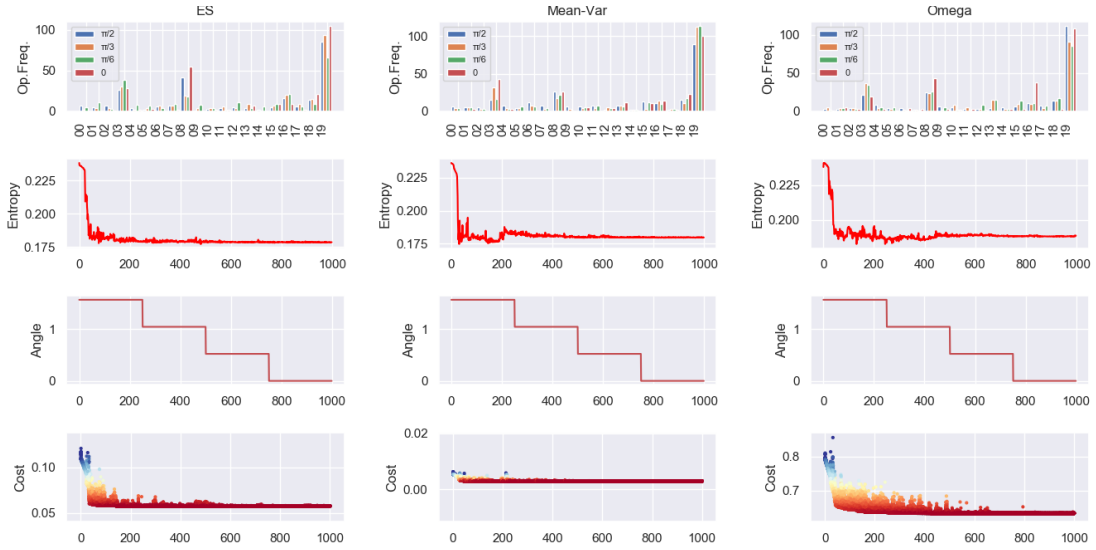
(a) Always moving strategy

Figure 4.9

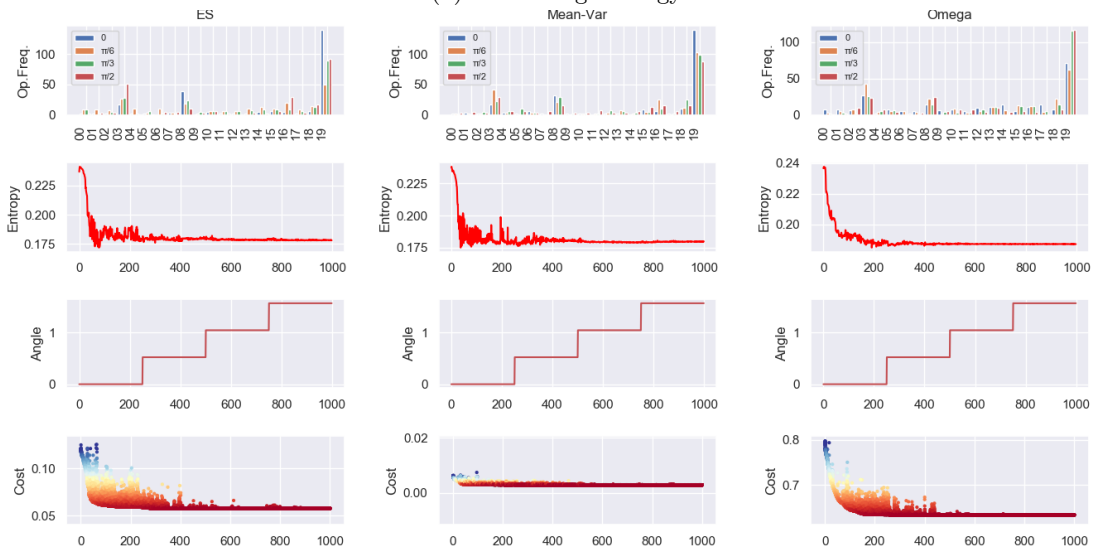
problems presented in figures 4.8a and 4.8b), which actually causes the operator selection probability to bounce and that ultimately drives a simultaneous upsurge both in quality and entropy. For what concerns the ES risk measure, the increasing strategy slightly outperforms the decreasing one in terms of best fitness in figure 4.8b and 4.8c, while the opposite happens in the case of the Hang Seng instance, where the decreasing strategy seems to achieve slightly better results towards later stages of search (see figures 4.9b and 4.9c). The behaviour of the controller looks promising when considering a reactive policy (figure 4.9d) as well, as both cost and entropy react promptly to shifting angle value. Further preliminary tests, not reported here, have been arranged to address the robustness of the results. In particular, our goal was to check if the algorithm was robust with respect to different starting solutions, especially with regards to the operator selection frequency. As a result, we have considered two main strategies. On the one hand, we have established a cardinality constraint, by imposing the condition that the portfolio is composed by no more than K assets, randomly picked from N available assets, with $K < P$. On the other hand, we have established a pre-assignment set, by which we have included P assets in the portfolio manually, with $K < P$; the residual $K - P$ assets are selected randomly. We have then run these two strategies on each instance, with $K \in \{10, 20, 30\}$. No significant differences have been found, with minimal modifications to the frequency histogram, apart from UNDX, whose performance proved to be quite sensitive to the problem instance at hand.

4.3 Test 3: evaluating the adaptive strategy on MIP problems

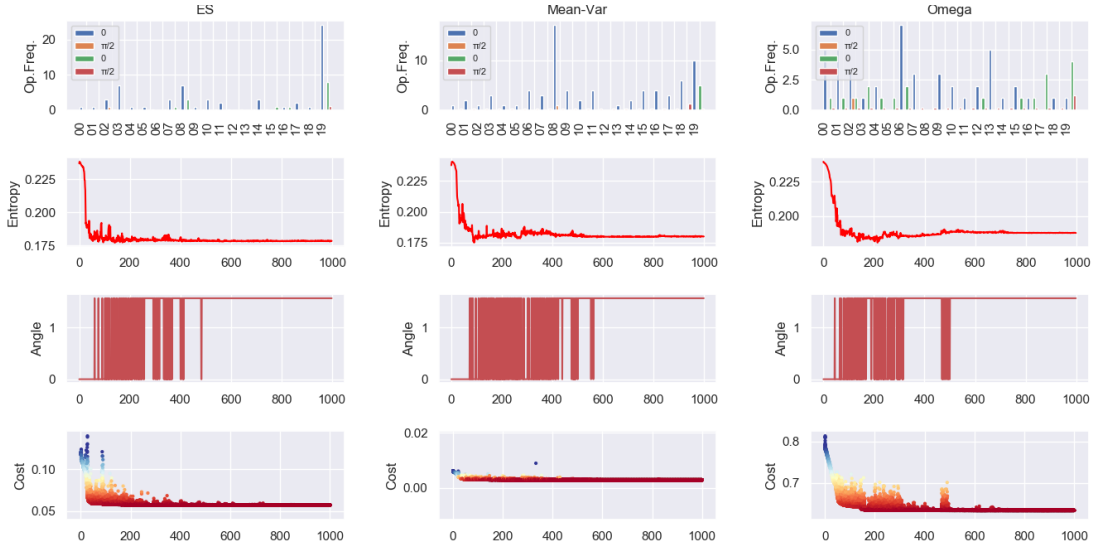
In this section we evaluate the performance of the adaptive operator selection strategy (AOS) on a variety of mixed-integer programming problems (MIPs). In particular, we consider the budget and no-short selling constraints described in subsection 4.1.2 and then we combine them with two standard integer constraints, i.e. the *cardinality constraint* and the *floor and ceiling constraints*, which we have discuss in depth in subsection 2.1.2. Recall that many of the risk measures proposed in subsection 4.1.2 deal with some recurring patterns in financial markets, such as fat-tailed and asymmetric returns. The problems described above cannot be tackled with standard optimization



(b) Decreasing strategy

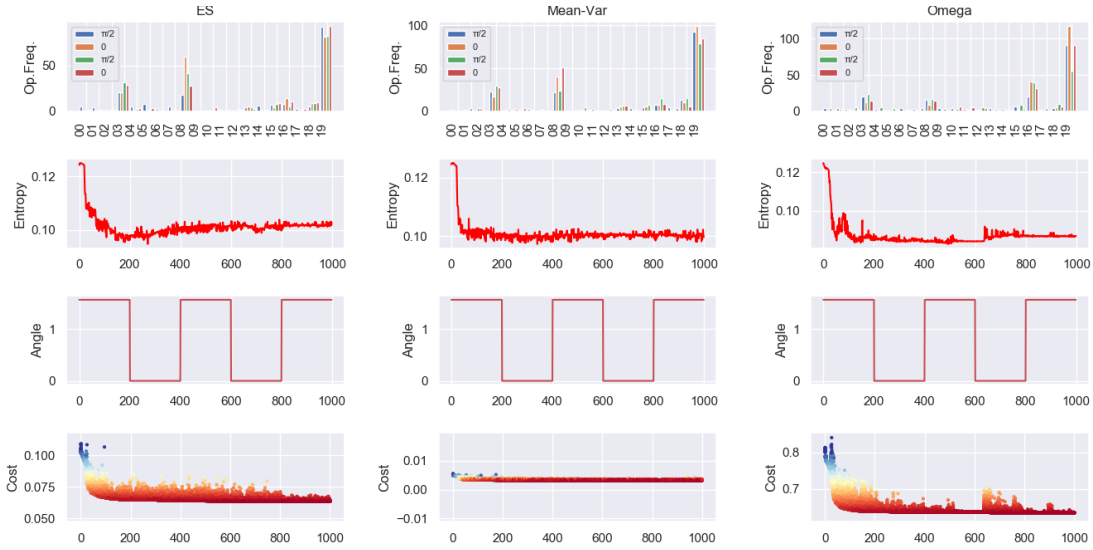


(c) Increasing strategy



(d) Reactive moving strategy

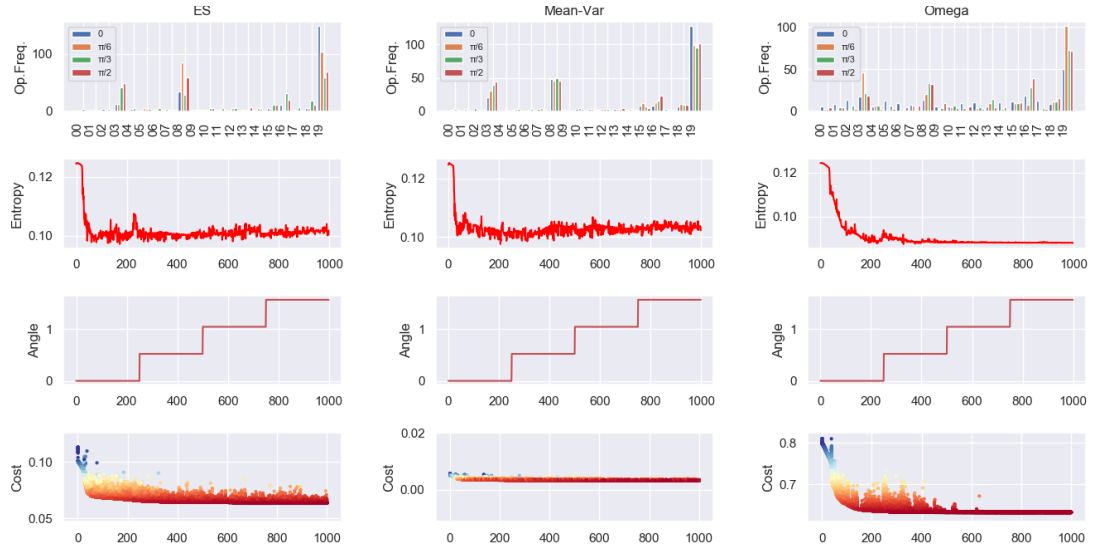
Figure 4.9: Experiments with the AOS: every portfolio selection strategy is fitted to the Hang Seng dataset. From top to bottom: histogram of the operator selection frequency (1), the entropy convergence curve (2), the dynamic angle function driving the search process (3), the individual cost scatter plot (4).



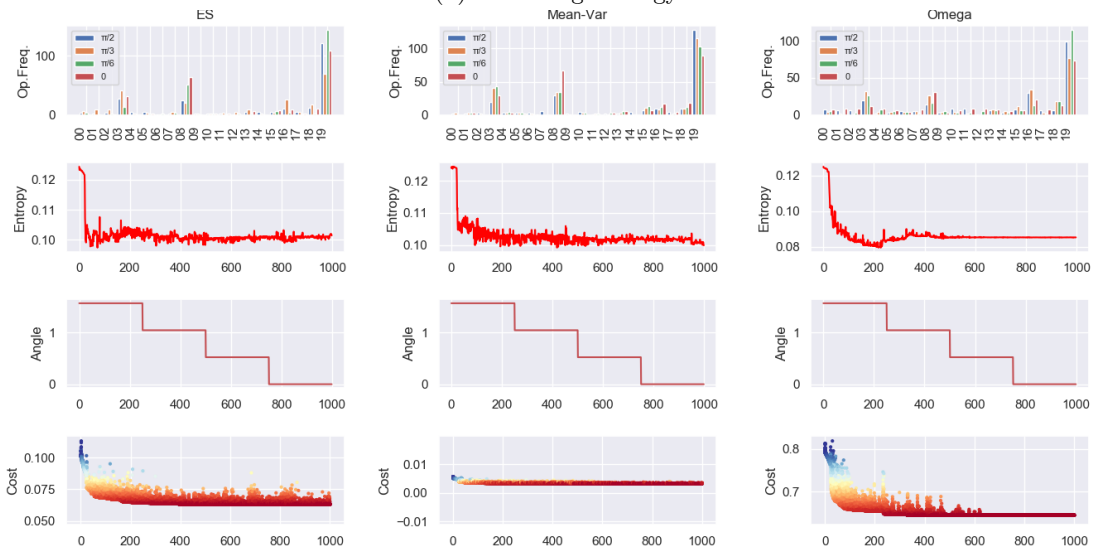
(a) Always moving strategy

Figure 4.10

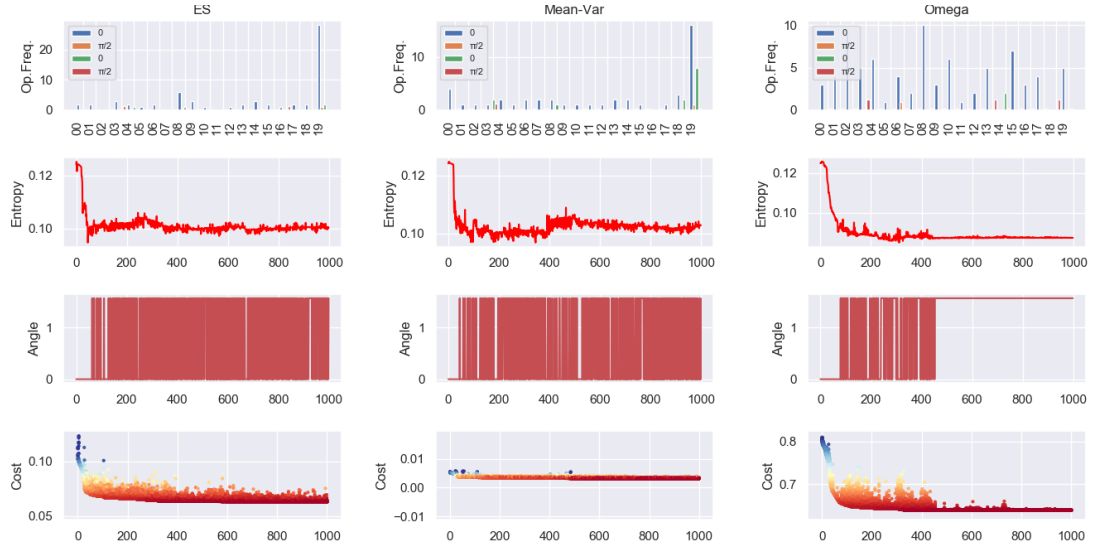
techniques and as we have stated before, a reformulation of optimization problems cannot anyway accomodate integer constraints, which make them even more complex, as integer-constrained problems typically exhibit discontinuities (Gilli et al. (2006)). Moral-Escudero et al. (2006) has proven that cardinality-constrained problems are *NP-hard*, so they cannot be solved optimally within polynomially-bounded computational time. In layman's terms this implies that for some problem instances the computational cost is so large that it is pretty much unmanageable for any conceivable amount of computational power. A discussion of the two most popular methods for solving mixed integer linear programs,, i.e. the *cutting planes* and the *branch and bound* algorithms goes well beyond the scope of this chapter; however, the intuition behind both algorithm is based on linear programming relaxation, i.e. the linear program obtained by dropping the integrality constraint. Both algorithm find a solution by solving systematically a sequence of linear programming relaxations, which are more tractable. Furthermore, recall that EAs are able to deal properly with global unconstrained optimization problems; therefore various methods have been proposed for constraint handling, i.e. penalty functions, special representations and operators, repar algorithms, separation of objectives and constraints and hybrid methods (Coello-Coello (2002)). For a thorough discussion of penalty approaches, we refer the reader to the penalty methods based on parameter control reviewed in subsection 1.3.3. Finally, in subsection 4.3.2.2 we propose an out-of-sample experiment by which we backtest the performance of the adaptive strategy and the performance of each operator on five instances. In particular, we implement a rolling-window backtest, in order to train the algorithm on new and more recent data at some point t; this strategy is generally more effective but it generally leads to higher transaction costs. As a consequence, we evaluate the impact of turnover of each approach; we take into account also standard performance measure, such as cumulative returns, standard deviation and the Sharpe Ratio. We do not compare the adaptive strategy with traditional benchmarks (such as naive $1/N$ strategies or stock market indexes), rather, we want to contrast the adaptive approach with simple genetic algorithms (SGAs); for this purpose, we also assess the quality of each solution, especially in terms of constraints violation.



(b) Increasing strategy

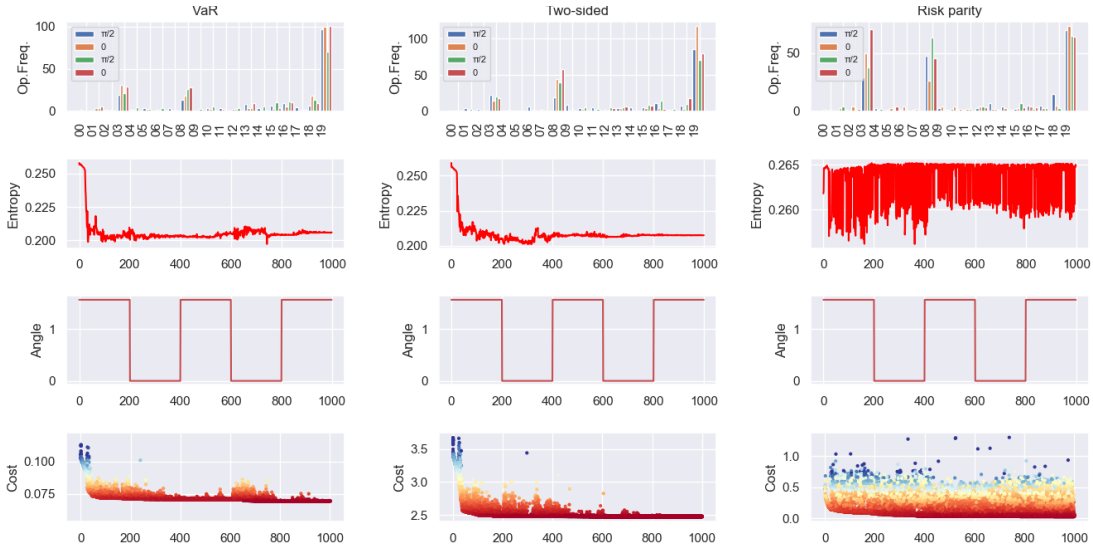


(c) Decreasing strategy



(d) Reactive moving strategy

Figure 4.10: Experiments with the AOS: every portfolio selection strategy is fitted to the FTSE 100 dataset. From top to bottom: histogram of the operator selection frequency (1), the entropy convergence curve (2), the dynamic angle function driving the search process (3), the individual cost scatter plot (4).

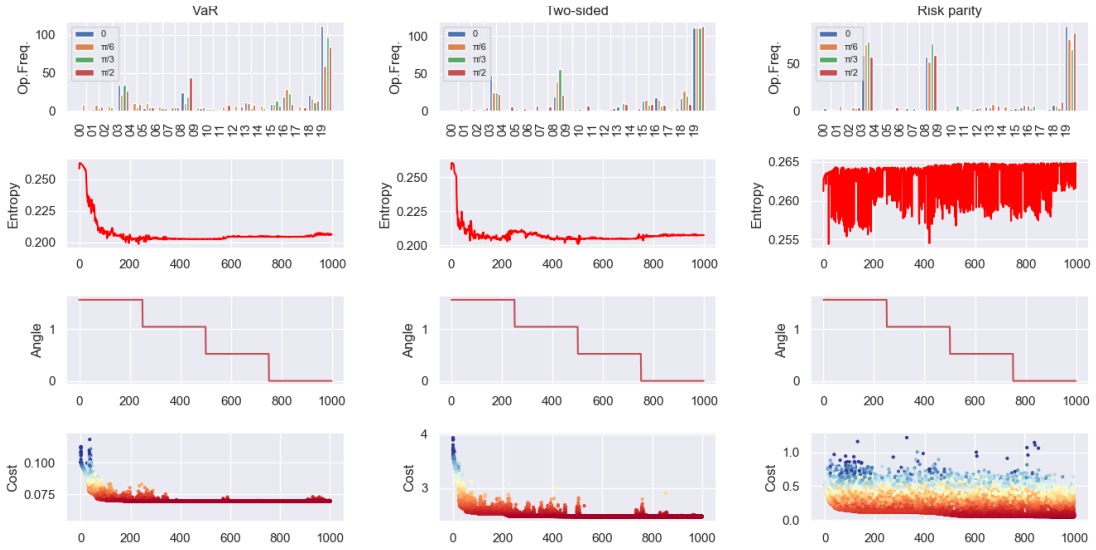


(a) Always moving strategy

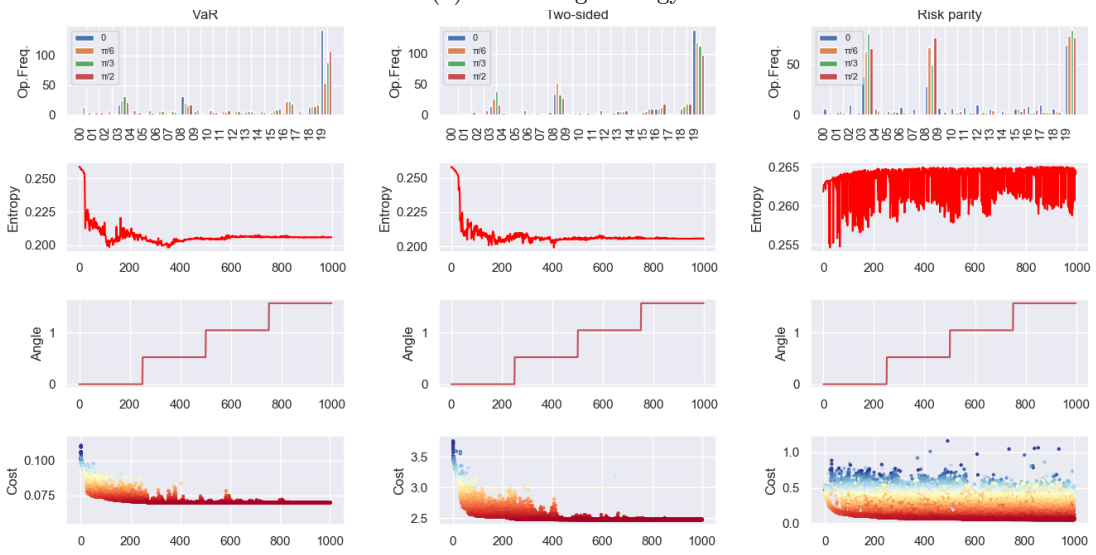
Figure 4.11

4.3.1 Experimental setting

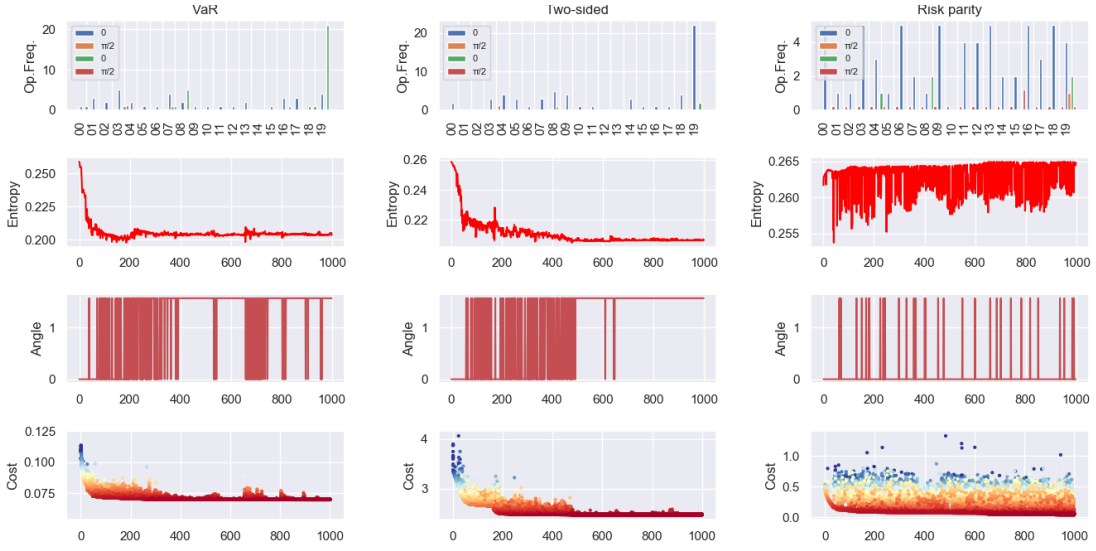
In the next subsections we perform a set of in-sample (4.3.2.1) and out-of-sample (4.3.2.2) experiments in order to evaluate the performance of the adaptive strategy on training data and then we run the model on an independent dataset repeatedly, with a rolling-window backtest. The essence of this approach is to study the robustness of a parameter control strategy applied to genetic algorithms, which we subsequently compare to simple genetic algorithms (SGAs), as defined in section 4.1, for which neither parameter tuning nor parameter control policies are considered. In other words, we want to detect the potential benefits of a procedure by which we encourage the controller to adjust the EvE balance in order to find a near-optimal solution; overall, we take into account the adaptive strategy and we compare it with twenty variation operators, which should serve as a benchmark of the parameter control approach. We construct portfolios with various combinations of cardinality constraints, lower/upper bound constraints and risk measures, so as to obtain more robust results across different portfolios. The results are reported for five different problem instances (whose properties are commented in section 4.1). We consider first the minimization capabilities of each algorithm, i.e. we report the average and the best in-sample value of the ℓ_1 penalty function 2.30, with the aim of obtaining a raw measure of in-sample performance of the algorithm in terms of cost. Furthermore, we implement a rolling-window backtest for each strategy, which is applied to our model based on parameter control and to standard genetic algorithms as well. The main reference for this rebalancing policy is Gilli et al. (2011). Essentially, we optimize the model on training data at time t_1 from $t_1 - \tau$; for this set of tests, we set the window τ equal to two years (about 500 data points). The portfolio is held for $F = 125$ days, i.e. until $t_2 = t_1 + F$. Therefore, the portfolio is trained on new data starting from $t_2 - \tau$ until $t_2 - 1$ and held for 125 more days, i.e. $t_3 = t_2 + F$. Basically, with the rolling-window procedure each time we include data for the next six month and we drop data for the earliest six months: overall, we repeat this procedure five times and this amounts to holding the portfolio for approximately 600 days. With T we denote the total number of returns in the dataset, while with $\mathcal{I}$ we denote the number of holding periods of each rebalanced portfolio. In this section we compute across five datasets the out-of-sample empirical performance using four performance metrics: the *annualized portfolio returns*, the *annualized standard deviation*, the *out-of-sample*



(b) Decreasing strategy



(c) Increasing strategy



(d) Reactive moving strategy

Figure 4.11: Experiments with the AOS: every portfolio selection strategy is fitted to the CAC 40 dataset. From top to bottom: histogram of the operator selection frequency (1), the entropy convergence curve (2), the dynamic angle function driving the search process (3), the individual cost scatter plot (4).

Table 4.3

	K	lb	up	ε	τ	F	T
Test 1	10	0.05	0.15	$1e-001$	600	125	1225
Test 2	20	0.04	0.12	$1e-001$	600	125	1225

Sharpe Ratio and *turnover*. For a given i portfolio strategy we use the time series of returns:

$$\begin{aligned}
\hat{\mu}^i &= \frac{1}{\mathcal{I}F}\sum_{j=1}^{\mathcal{I}}\sum_{k=1}^{F} x_{t_j}^{i^T} r_{t_j+k}^i && \text{average daily out-of-sample returns} \\
\hat{\sigma}^i &= \sqrt{\left(\frac{1}{\mathcal{I}F-1}\sum_{j=1}^{\mathcal{I}}\sum_{k=1}^{F} x_{t_j}^{i^T} r_{t_j+k}^i - \hat{\mu}^i\right)^2} && \text{daily out-of-sample standard deviation} \\
\hat{SR}^i &= \frac{\hat{\mu}^i}{\hat{\sigma}^i} && \text{out-of-sample Sharpe Ratio} \\
Turnover &= \frac{1}{\mathcal{I}-1}\sum_{j=1}^{\mathcal{I}-1}\sum_{k=1}^{N}\left(\left|x_{k,t_{j+1}}^i - x_{k,t_j}^i\right|\right)
\end{aligned} \tag{4.4}$$

Furthermore, we report simple graphs in which we show the out-of-sample daily cumulative returns; we compute the latter by taking the cumulative product of the daily out-of-sample returns. The indexed time series are constructed starting from an initial value $C^i = 100$.

For what concerns the impact of transaction costs, some preliminary backtests have shown that transaction costs do not influence considerably the portfolio performance, which is consistent with the results presented in Gilli et al. (2011). In particular, we have considered a wide range of transaction costs, from 10 basis points to 50 basis points for each position in the portfolio, with a small impact on portfolio performance. Likely, this is due to the fact that the portfolio is rebalanced at a quite low frequency.

Nonetheless, in order to evaluate how transaction costs could actualy affect each portfolio strategy, we compute the average turnover over the out-of-sample period, as reported in 4.4 (see, for instance, DeMiguel et al. (2009)). x_{k,t_j}^i is the i-th portfolio weight of asset k and $x_{k,t_{j+1}}^i$ is the same portfolio weight after rebalancing. The turnover measure is equal to the sum of rebalancing trades across N assets over the out-of-sample time period divided by the number of rebalancements. For our experiments, we choose first all the parameters involving the adaptive algorithm: for what concerns the general parameter setting choices, we refer the reader to subsection 4.1.1, in particular to table 4.1, in which all the strategic parameters are detailed. The only exception is that here we perform tests with the increasing strategy straight away, as it is both theoretically and empirically the most promising (see 4.2.2). As far as the portfolio selection problem is concerned, we basically perform two set of tests: the parameters setting are reported in table 4.3, in which, for the sake of readability, we also report the key parameters involved in the rolling-window backtest.

4.3.2 Results

In what follows, we evaluate the in-sample and out-of-sample performance of the adaptive strategy. In the following subsection we simply check the convergence towards a good solution for each algorithm, then we evaluate the out-of-sample performance of rebalanced portfolios with four different performance metrics.

4.3.2.1 Evaluating the in-sample performance of the adaptive policy

The first analysis is a basic in-sample test which serves as sanity check, with the aim of verifying whether each strategy can converge at least to a good solution across different problem instances and across different risk measures. The cost is computed according to the ℓ_1 exact penalty function 2.30. It is especially important to examine the cost for different combinations of risk measures and problem instances of the adaptive strategy, given that the basic genetic algorithms, which are based on a standard construction, mainly serve as a benchmark for the dynamic policy. By managing a variety of operators in an adaptive fashion, according to an optimal policy of EvE, we address a typical issue of evolutionary algorithms (for instance, some algorithms may perform better on a problem rather than another, or sometimes other an algorithm may outperform another one at a specific stage of the problem, a topic we have abundantly discussed in 1.1). Moreover, the discussion in section 3.1 has shown that only a few operators display, at least in part, self-adaptive features, by which they steer the search process on the basis of some dispersion measures, adjusting the EvE balance accordingly. Most operators, instead, are typically tilted towards exploitation and this could lead the algorithm to converge to solutions of poor quality. The basic idea behind parameter control is exactly to find a suitable parameter choice to cope with different structures of the problem at hand.

	$\Omega_{+penalty}$		$VaR_{95\%+penalty}$		$ES_{95\%+penalty}$		$MV_{0.5+penalty}$		$\rho_{0.25,2+penalty}$		$ERC_{+penalty}$	
Nikkei 225	avg.	best	avg.	best	avg.	best	avg.	best	avg.	best	avg.	best
OPX	1.0344	0.0067	0.2255	0.0066	0.2362	0.0069	0.1248	0.0069	1.7108	0.0065	0.1830	0.0079
UX	1.0518	0.0083	0.2906	0.0069	0.2995	0.0071	0.1302	0.0074	1.7289	0.0111(U)	0.2308	0.0083
HX	1.0092	0.0069	0.2161	0.0068	0.2301	0.0069	0.1422	0.0072	1.6842	0.0074	0.1779	0.0067
LX	1.0394	0.5039(U)	0.2408	0.0071	0.2104	0.0074	0.1309	0.0075	2.2413	0.0073	0.2067	0.0188(U)
QBX	1.0218	0.0071	0.2349	0.0071	0.2962	0.0071	0.1754	0.0068	1.7233	0.0068	0.1571	0.0076
TPX	0.9956	0.0077	0.2409	0.0072	0.2686	0.0071	0.1228	0.0075	1.7183	0.0098(U)	0.1949	0.0104(U)
AX	0.9986	0.0069	0.2483	0.0071	0.2686	0.0067	0.1246	0.0073	1.7031	0.0065	0.1884	0.0075
GX	0.9899	0.0072	0.2212	0.0068	0.2662	0.0069	0.1059	0.0076	1.6956	0.0065	0.2001	0.0085
SBX	1.0192	0.0073	0.2786	0.0071	0.2869	0.0069	0.1611	0.0072	1.6951	0.0065	0.1981	0.0114(U)
AVX	0.9826	0.0071	0.2488	0.0069	0.2541	0.0067	0.1038	0.0072	1.7527	0.0066	0.1891	0.0068
BLX	1.0462	0.0071	0.2625	0.0072	0.3081	0.0067	0.1266	0.0067	1.6631	0.0064	0.1817	0.0069
FX	0.9815	0.0072	0.2728	0.0068	0.2711	0.0065	0.1413	0.0072	1.6808	0.0063	0.1695	0.0067
GUX	1.0409	0.0069	0.2955	0.0073	0.2858	0.0073	0.1254	0.0074	1.7318	0.0068	0.2268	0.0078
TPX	0.9731	0.0095	0.2061	0.5041(U)	0.2401	0.0081	0.1365	0.0068	1.6795	0.0082	0.1931	0.0112(U)
LNX	1.5401	0.0067	0.2971	0.0073	0.2983	0.0082	0.1851	0.0067	1.7201	0.0076	0.1855	0.0093
DBX	1.3054	0.0077	0.4567	0.0077	0.4783	0.0078	0.2531	0.0079	1.9232	0.0151(U)	0.1959	0.0071
UNDX	1.0863	0.0073	0.3540	0.0068	0.2881	0.0068	0.1933	0.0071	1.6671	0.0071	0.2399	0.0075
FR	0.9872	0.0068	0.2251	0.0071	0.2543	0.0067	0.1251	0.0072	1.7178	0.0112(U)	0.1811	0.0067
SPX	1.0095	0.2431(U).	0.2956	0.0781	0.2654	0.0089.	0.1768	0.0079	1.6943	0.0064	0.2031	0.0109(U)
PNX	0.9991	0.0075	0.2589	0.0071	0.2411	0.0088	0.3031	0.0070	1.8555	0.0068	0.2111	0.0091
Adaptive	0.9654	0.0069	0.1935	0.0072	0.2322	0.0086	0.1412	0.0069	1.6417	0.0068	0.1637	0.0091
FTSE 100	avg.	best	avg.	best	avg.	best	avg.	best	avg.	best	avg.	best
OPX	1.0375	0.0055	0.2202	0.0052	0.2354	0.0051	0.1083	0.0051	1.2221	0.0074	0.1473	0.0081
UX	1.5515	0.4984(U)	0.2018	0.0051	0.2098	0.0055	0.1081	0.0052	1.2367	0.0062	0.1563	0.0055
HX	1.0333	0.0055	0.1889	0.0051	0.2017	0.0049	0.1005	0.0053	1.2225	0.0049	0.1482	0.0048
LX	1.0051	0.0051	0.2141	0.0059	0.2535	0.0051	0.1576	0.0071	1.5602	0.0055	0.1227	0.0081
QBX	1.0081	0.0052	0.2531	0.0051	0.2385	0.0047	0.1511	0.0053	1.2668	0.0061	0.1231	0.0061
TPX	1.0160	0.0053	0.2308	0.0054	0.2115	0.0053	0.1226	0.0053	1.2451	0.0064	0.1467	0.0051
AX	1.0172	0.0053	0.1953	0.0050	0.1986	0.0051	0.1054	0.0057	1.3151	0.0057	0.1696	0.0047
GX	1.0392	0.0053	0.1883	0.0051	0.1992	0.0050	0.1287	0.0053	1.2271	0.0048	0.1825	0.0051
SBX	1.0082	0.0054	0.1984	0.0057	0.2120	0.0052	0.1168	0.0053	1.2429	0.0047	0.2192	0.0052
AVX	0.9918	0.0053	0.1686	0.0052	0.1815	0.0051	0.1094	0.0049	1.2326	0.0048	0.1435	0.0074
BLX	1.0525	0.0053	0.2321	0.0051	0.2849	0.0051	0.1664	0.0054	1.2525	0.0045	0.1488	0.0047
FX	1.0576	0.0062	0.2115	0.0053	0.2184	0.0052	0.1396	0.0057	1.2747	0.0057	0.1713	0.0061
GUX	1.0536	0.0059	0.2097	0.0051	0.2055	0.0052	0.1170	0.0053	1.8771	0.0057	0.1645	0.0076
TPX	1.0641	0.0052	1.1893	0.9162(U)	0.2227	0.0049	0.1652	0.0051	1.8184	0.0051	0.1406	0.0078
LNX	1.1606	0.0051	0.3846	0.0053	0.3521	0.0054	0.3701	0.0059	1.3849	0.0051	0.2031	0.0095(U)
DBX	1.1811	0.0055	0.3355	0.0054	0.3039	0.0053	0.2052	0.0053	1.3401	0.0048	0.2656	0.0049
UNDX	0.9939	0.0053	0.2135	0.0061	0.2421	0.0051	0.1217	0.0057	1.2796	0.0067	0.1570	0.0046
FR	1.0721	0.0054	0.2051	0.0065	0.2329	0.0053	0.1552	0.0058	1.3178	0.0062	0.1911	0.0067
SPX	1.0923	0.0065	0.2298	0.0063	0.2237	0.0059	0.1499	0.0062	1.7944	0.0069	0.1928	0.0078
PNX	1.0201	0.0059	0.1998	0.0069	0.2772	0.0065	0.2435	0.0059	1.3002	0.0059	0.1326	0.0080
Adaptive	1.0031	0.0051	0.1936	0.0053	0.1921	0.0051	0.0776	0.0052	1.2190	0.0049	0.1198	0.0050
Hang Seng	avg.	best	avg.	best	avg.	best	avg.	best	avg.	best	avg.	best
OPX	0.9625	0.0077	0.2505	0.0078	0.2566	0.0077	0.1508	0.0078	1.8924	0.0078	0.1580	0.0099(U)

UX	0.9828	0.0081	0.2507	0.0081	0.2564	0.0077	0.1342	0.0079	2.9282	0.9510(U)	0.2122	0.0078
HX	0.9861	0.0081	0.2252	0.0077	0.2566	0.0077	0.1093	0.0078	1.9214	0.0073	0.1446	0.0075
LX	1.0766	0.0085	0.7404	0.0051	0.2555	0.0078	0.1218	0.0081	1.9183	0.0108(U)	0.1524	0.0077
QBX	0.9615	0.0079	0.2398	0.0076	0.2870	0.0076	0.1205	0.0077	1.9187	0.0075	0.2294	0.0084
TPX	0.9227	0.0079	0.2493	0.0081	0.2618	0.0082	0.3853	0.2547(U)	1.9452	0.0088	0.2455	0.0083
AX	0.9223	0.0080	0.2621	0.0077	0.2984	0.0078	0.0933	0.0078	1.9331	0.0075	0.1477	0.0011
GX	0.9022	0.0081	0.2297	0.0078	0.2637	0.0078	0.1303	0.0079	1.9213	0.0081	0.1711	0.1083(U)
SBX	0.9696	0.0082	0.2998	0.0081	0.2784	0.0079	0.1419	0.0079	1.9203	0.0079	0.1844	0.0131(U)
AVX	0.9472	0.0081	0.2516	0.0079	0.2874	0.0076	0.1193	0.0077	1.9305	0.0081	0.2223	0.0074
BLX	0.9461	0.0081	0.3191	0.0076	0.3641	0.0077	0.2399	0.0079	2.0317	0.0076	0.1846	0.0075
FX	0.9708	0.0083	0.2534	0.0076	0.2418	0.0076	0.1505	0.0078	1.8784	0.0076	0.1581	0.0074
GUX	0.9542	0.0081	0.2675	0.0078	0.2925	0.0052	0.1114	0.0078	1.8795	0.0094	0.1981	0.0084
TPX	0.9323	0.0084	0.3142	0.0077	0.3036	0.0081	0.1226	0.0078	1.9081	0.0083	0.2473	0.0078
LNX	0.9605	0.0084	0.7839	0.4824(U)	1.2759	0.0054	0.1266	0.0075	1.9357	0.0075	0.1633	0.0101(U)
DBX	1.0213	0.0079	0.2928	0.0081	0.4411	0.0081	0.4949	0.0081	2.2223	0.0081	0.1773	0.0081
UNDX	1.0691	0.0082	0.3572	0.0078	0.4119	0.0079	0.2665	0.0079	2.0507	0.0088	0.2889	0.0077
FR	0.9866	0.0078	0.2261	0.0078	0.2747	0.0077	0.1795	0.0085	1.9490	0.0081	0.2292	0.0103(U)
SPX	1.0119	0.0085	0.2678	0.0085	0.2965	0.0078	0.2908	0.0091	1.8698	0.0077	0.2301	0.0081
PNX	0.9821	0.0089	0.2133	0.0089	0.2919	0.0080	0.1554	0.0092	1.9381	0.0084	0.2431	0.0071
Adaptive	0.9002	0.0072	0.1991	0.0079	0.2111	0.0077	0.0934	0.0049	1.2190	0.0059	0.2071	0.0079
FTSE MIB	avg.	best	avg.	best	avg.	best	avg.	best	avg.	best	avg.	best
OPX	1.0225	0.0069	0.2305	0.0078	0.2342	0.0069	0.1718	0.0068	1.6921	0.0071	0.1980	0.0079
UX	0.9978	0.0071	0.2607	0.0071	0.2464	0.0075	0.1434	0.0069	1.8276	0.0110(u)	0.1922	0.0077
HX	1.0861	0.0075	0.2152	0.0067	0.2506	0.0076	0.1183	0.0073	1.7204	0.0073	0.1855	0.0077
LX	0.9966	0.0075	0.2404	0.0071	0.2425	0.0077	0.1429	0.0071	1.6883	0.0098(U)	0.1474	0.0077
QBX	0.9915	0.0079	0.2158	0.0077	0.2870	0.0072	0.1305	0.0077	1.8149	0.0091	0.1291	0.0074
TPX	1.0227	0.0078	0.1993	0.0177(U)	0.2218	0.0072	0.1853	0.0067	1.3752	0.0091	0.1495	0.0083
AX	0.9923	0.0080	0.2241	0.0078	0.2774	0.0081	0.1123	0.0059	1.8531	0.0079	0.1677	0.0071
GX	0.9922	0.0086	0.2097	0.0068	0.2257	0.0078	0.1303	0.0069	1.6918	0.0088	0.1815	0.093
SBX	0.9896	0.0072	0.1998	0.0081	0.2584	0.0079	0.1519	0.0075	1.9293	0.0077	0.2044	0.0098(U)
AVX	1.0452	0.0071	0.2116	0.0075	0.2314	0.0075	0.1853	0.0074	1.8305	0.0081	0.2153	0.0094
BLX	1.0421	0.0071	0.2191	0.0072	0.2617	0.0072	0.2145	0.0071	1.9987	0.0079	0.1646	0.0079
FX	1.0718	0.0073	0.2134	0.0072	0.2418	0.0071	0.1708	0.0078	1.7754	0.0076	0.1891	0.0078
GUX	1.0522	0.0081	0.2075	0.0071	0.2954	0.0062	0.1348	0.0079	1.8785	0.0090	0.1832	0.0084
TPX	1.0333	0.0074	0.2182	0.0079	0.2346	0.0071	0.1896	0.0071	1.8091	0.0081	0.2269	0.0077
LNX	1.0605	0.0074	0.1979	0.0080	0.2779	0.0064	0.1419	0.0065	1.6397	0.0085	0.1921	0.0076
DBX	1.0313	0.0069	0.2128	0.0079	0.2451	0.0071	0.2446	0.0081	2.0593	0.0076	0.1903	0.0088
UNDX	1.0132	0.0082	0.2572	0.0078	0.2316	0.0079	0.2165	0.0070	1.9587	0.0078	0.2109	0.0095
FR	0.9951	0.0068	0.2361	0.0083	0.2528	0.0074	0.1965	0.0084	1.8450	0.0079	0.2099	0.0093
SPX	1.0354	0.0085	0.2098	0.0085	0.2876	0.0082	0.2354	0.0089	1.6876	0.0071	0.2269	0.0081
PNX	1.0543	0.0083	0.2367	0.0082	0.2112	0.0086	0.1987	0.0076	1.6993	0.0069	0.1903	0.0075
Adaptive	0.9946	0.0079	0.1977	0.0077	0.2212	0.0079	0.1839	0.0085	1.7190	0.0069	0.1671	0.0065
CAC 40	avg.	best	avg.	best	avg.	best	avg.	best	avg.	best	avg.	best
OPX	1.0112	0.0075	0.2143	0.0078	0.2259	0.0071	0.1348	0.0071	1.2953	0.0069	0.1823	0.0089
UX	0.9928	0.0080	0.2054	0.0071	0.2167	0.0073	0.1349	0.0072	1.3369	0.0061	0.1917	0.0074
HX	1.0162	0.0079	0.2152	0.0069	0.2336	0.0077	0.1693	0.0075	1.4248	0.0063	0.2005	0.0065
LX	1.0031	0.0079	0.2044	0.0059	0.2041	0.0075	0.1755	0.0068	1.2192	0.0069	0.1994	0.0069
QBX	0.9595	0.0076	0.2195	0.0069	0.2237	0.0071	0.1921	0.0067	1.3121	0.0061	0.2012	0.0059
TPX	0.9727	0.0078	0.2233	0.0079	0.2504	0.0088	0.2249	0.0077	1.3406	0.0059	0.2032	0.0073
AX	0.8998	0.0072	0.2572	0.0074	0.2777	0.0079	0.1171	0.0079	1.2301	0.0063	0.1998	0.0069
GX	0.9029	0.0079	0.2007	0.0074	0.2653	0.0072	0.1101	0.0175(U)	1.2285	0.0071	0.1857	0.0089
SBX	0.9997	0.0085	0.1991	0.0068	0.2282	0.0073	0.1009	0.0089	1.2245	0.0069	0.1799	0.0079
AVX	0.9443	0.0086	0.1905	0.0069	0.2589	0.0069	0.1423	0.0089	1.2201	0.0051	0.2147	0.0064
BLX	1.0162	0.0072	0.2231	0.0059	0.2985	0.0067	0.1739	0.0085	1.4312	0.0066	0.1896	0.0088
FX	0.9968	0.0085	0.2137	0.0065	0.3018	0.0066	0.1293	0.0079	1.2774	0.0059	0.2052	0.0071
GUX	0.9867	0.0075	0.2479	0.0059	0.2921	0.0070	0.1019	0.0069	1.2722	0.0084	0.1901	0.0054
TPX	1.0123	0.0079	0.2772	0.0067	0.3235	0.0065	0.1226	0.0068	1.2024	0.0101(U)	0.2332	0.0095
LNX	0.9765	0.0078	0.3049	0.0058	0.2759	0.0069	0.2156	0.0059	1.2337	0.0057	0.1435	0.0067
DBX	0.9564	0.0081	0.2018	0.0061	0.3911	0.0072	0.1402	0.0071	1.2423	0.0089	0.1991	0.0092
UNDX	0.9811	0.0082	0.1892	0.0069	0.4714	0.0075	0.1768	0.0069	1.3508	0.0063	0.2453	0.0108(U)
FR	1.0232	0.0078	0.2102	0.0059	0.2601	0.0064	0.1348	0.0069	1.4460	0.0068	0.2812	0.0079
SPX	1.0349	0.0089	0.2878	0.0069	0.3562	0.0077	0.1669	0.0078	1.4987	0.0074	0.1902	0.0069
PNX	1.0341	0.0091	0.2684	0.0075	0.2954	0.0075	0.1901	0.0070	1.4903	0.0089	0.2384	0.0109(U)
Adaptive	0.9146	0.0076	0.2487	0.0081	0.2511	0.0077	0.1134	0.0081	1.8190	0.0075	0.1371	0.0075

Table 4.4: In-sample tests with cardinality and upper/lower bound constraints, $K = 10$, $lb = 0.05$ and $ub = 0.15$, $\varepsilon = 1e-001$. (U) denotes an infeasible solution.

In table 4.4 we report the results with the settings listed in the first row of table 4.3. We can draw some conclusions on the basis of the reported costs, for which we make a distinction between average and best cost; for the latter, we also specify the feasibility of the solution. The penalty approach indeed allows the optimization algorithm to visit infeasible solution during the search process: a preliminary experiment in which we compared infeasibility handling both with an embedded repair

mechanism and with the penalty function 2.30 has shown that exploration operators tend to underperform all the others when the penalty approach is applied, sometimes converging to very poor solutions and leading generally to infeasible solutions. The repair approach, instead, by forcing to satisfy the constraints at each generation, keeps the diversification in the population under control; overall, it induces the search algorithm to limit the exploration of vast areas of the search space, so that in the end this latter strategy of constraint handling usually leads to more uniform solution across operators. Consequently, some operators need to receive a larger amount of penalty for constraint violation to perform better and this can be achieved through the parameter ε, which is equal to 0.1 for this test, though it may need even lower values to encourage the convergence towards optimal solutions and preventing the search process from getting stuck in infeasible solutions.

Note that certain operators cannot manage to converge towards good solutions (table 4.4) in some cases, with a few solutions under the best columns resulting in particularly modest performance, though generally most operators are able to achieve low costs. The average costs are in general quite homogeneous for most operators: hence, the identification of an outperforming operator is not straightforward; this does not come as a surprise, given that most operators are highly specialized in solving particular problems, while others may be more successful at solving others. A similar conclusion can be drawn when various objective functions are taken into account: we note that on average only the last two operators in table 4.4 tend to slightly underperform the others, especially in terms of the best solution found; the results are reasonably consistent with the findings of section 3.1.

The adaptive parameter control policy seems to slightly outperform on average the simple genetic algorithm (SGA) approach, though not substantially. To recap, an in-sample test itself is not enough to evaluate the goodness of a model, but rather to provide a first impression of the operators behaviour. This introductory test has successfully shown the tendency of each operator to display an above-average performance only on a small subset of optimization problems, while on average we cannot pick a systematically outperforming operator. The adaptive policy has proven to be an effective approach to deal with a variety of optimization problems, leading to good results on every dataset and for most cost functions, though in general we do not find evidence of robust outperformance.

4.3.2.2 Testing the out-of-sample performance of the adaptive policy

In this out-of-sample experiment we evaluate the performance of the adaptive policy and we use simple genetic algorithms as benchmarks. In particular, we consider some basic performance metrics (see equation 4.4) to understand in depth its behaviour across a variety of problem instances and cost functions. The idea, indeed, is to verify whether a parameter control policy can produce above-average results when compared to standard heuristic optimization techniques. Moreover, it is especially important to evaluate the robustness of this policy, i.e. with some oversimplification, we want to check whether the output of the model is accurate and outperforming even for changing problems. One of the main advantages of parameter control discussed in section 1.3 is related to its flexibility, namely it allows the EA to use appropriate parameter values at different stages of the search process (e.g. Karafotias et al. (2015). A crossover operator, as Herrera et al. (2005) argues, is more effective when its search bias is adjusted to the structure of the problem at hand. In this case, we rather select each time a different operator, in order to encourage a 'dynamic approach' when solving different optimization problems.

In order to provide a first impression about the quality of the adaptive policy, we plot the cumulative daily returns of each strategy with $K = 10$ in Figures 4.12-4.16 and with $K = 20$ in Figures 4.17-4.21. Overall, the adaptive portfolio performs positively, outperforming most simple genetic algorithms for both classes of cardinality

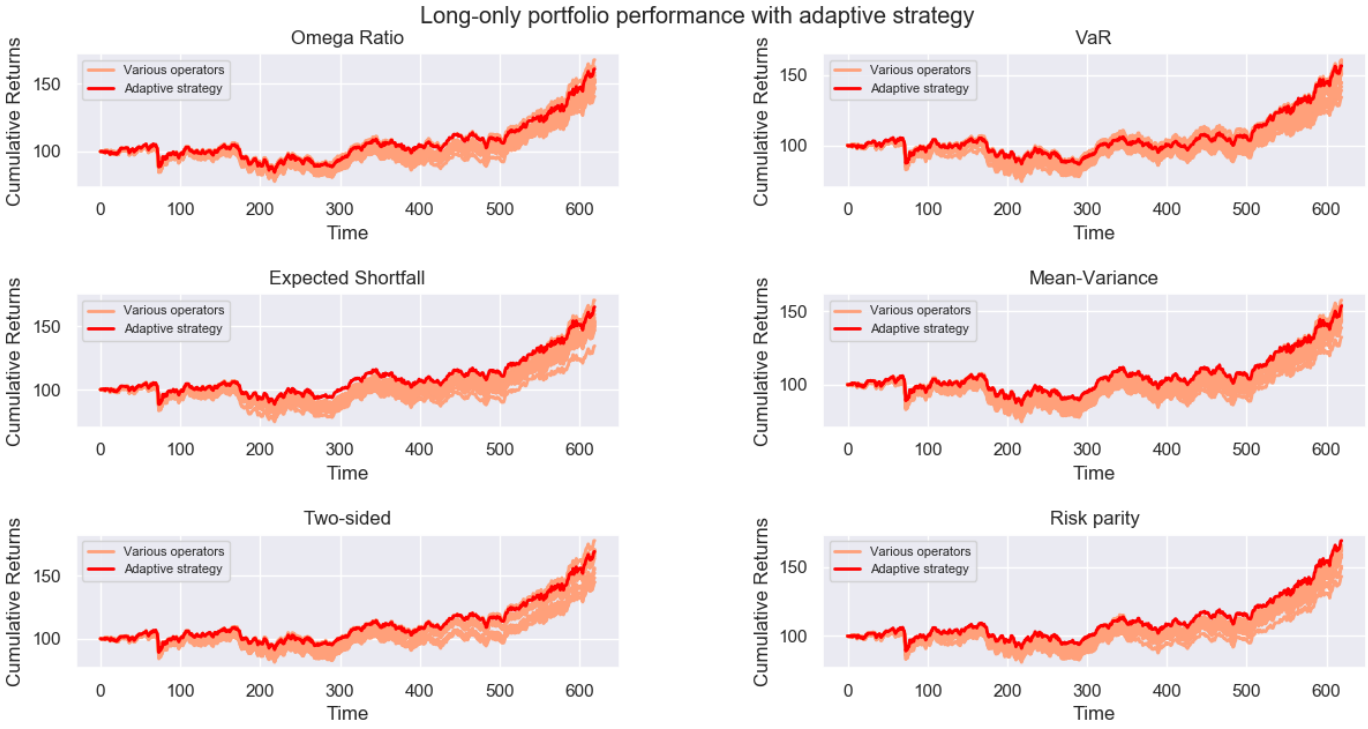


Figure 4.12: Rolling-window backtest of the adaptive strategy on Nikkei 225 index, with $K = 10$, $lb = 0.05$ and $ub = 0.15$

constrained portfolios[1] (i.e. $K = 10$ and $K = 20$); specifically, note that the adaptive approach displays a particular appealing performance when applied to the two-sided risk measure and to risk parity: for some datasets, indeed, the adaptive algorithm manages to outperform all the operators, in some cases even by a wide margin, while for other instances we note a good overall performance. As far as 'standard' risk measures are involved, the benefits of parameter control are less clear, but on average we observe the adaptive strategy is regularly among the best performing ones in terms of cumulative returns, whereas the basic strategies are less consistent across datasets.

Tables 4.5-4.10 report the performance statistics of each strategy for each risk measure, relative to biyearly rebalanced portfolios. Each table includes the results for all datasets, for a varying amount of the cardinality K. In terms of Sharpe Ratio performance, we note that the best results are achieved with $K = 10$ for three out of five instances (Nikkei 225, FTSE 100 and CAC 40), though for a few operators we observe an opposite trend. The other two datasets (Hang Seng and FTSE MIB) display a larger Sharpe Ratio for $K = 20$; therefore, we conclude that the impact of the cardinality constraint on portfolio performance is mostly related to the data cointaned in each sample and consequently we cannot generalize the results, as the relationship between portfolio performance and the value of K is not clear-cut at all: selecting a smaller subset of assets from the stock market index is not necessarily desirable.

The most profitable strategy in terms of annualized Sharpe Ratio is definitely the one based on the two-sided risk measure, though also the portfolios constructed with the Omega Ratio and the ERC approach display a fair performance. If we consider the Nikkei 225 instance, for the former the average SR is well above one, while the Sharpe

[1] Actually, for what concern risk parity, we are not constructing equally-weighted risk contribution portfolios; we rather enforce integer-constraints and at the same time we obtain 'as ERC as possible' portfolios, see Gilli and Schumann (2021).

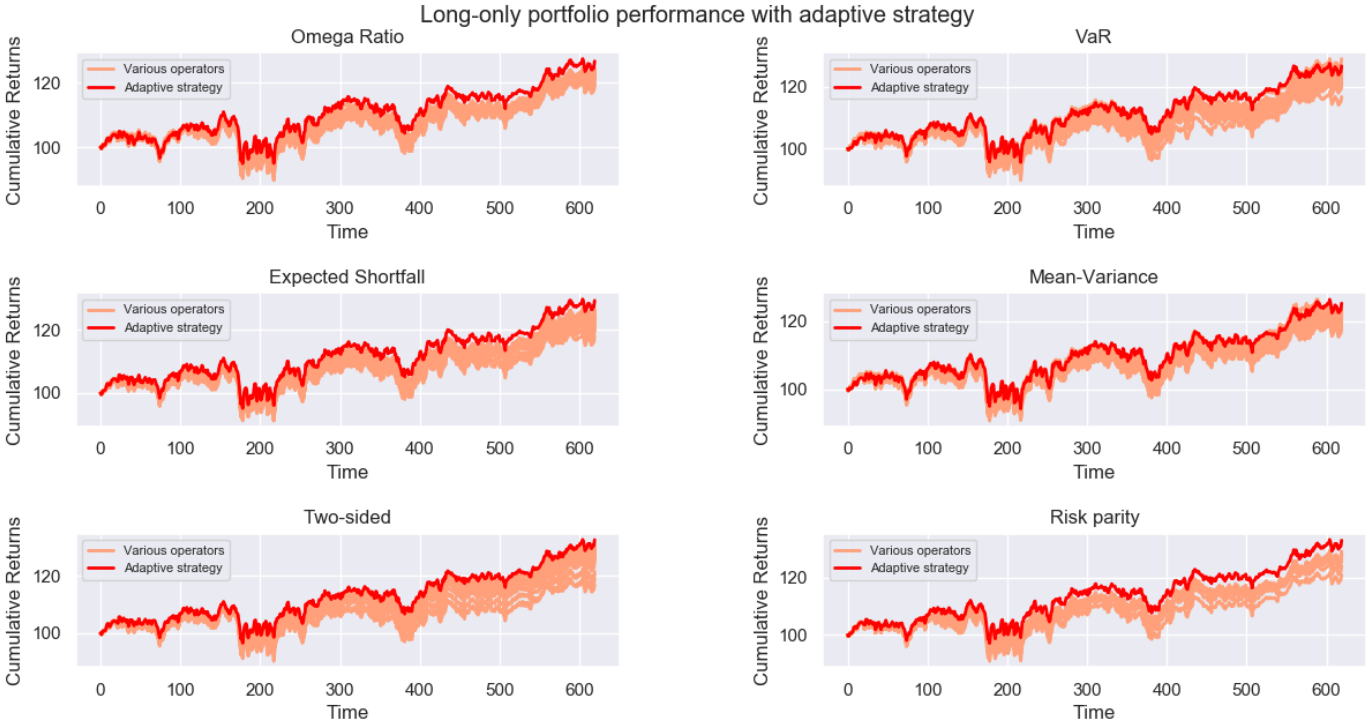


Figure 4.13: Rolling-window backtest of the adaptive strategy on FTSE100 index with $K = 10$, $lb = 0.05$ and $ub = 0.15$

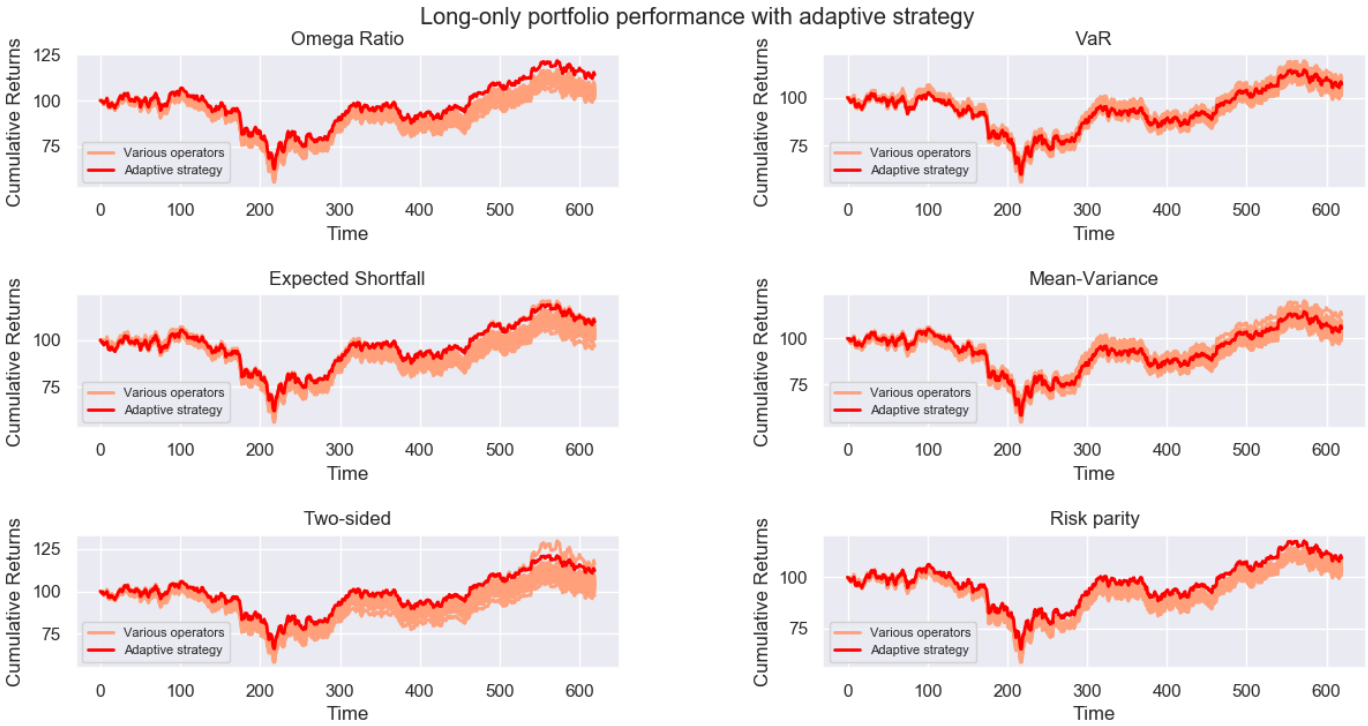


Figure 4.14: Rolling-window backtest of the adaptive strategy on Hang Seng index with $K = 10$, $lb = 0.05$ and $ub = 0.15$

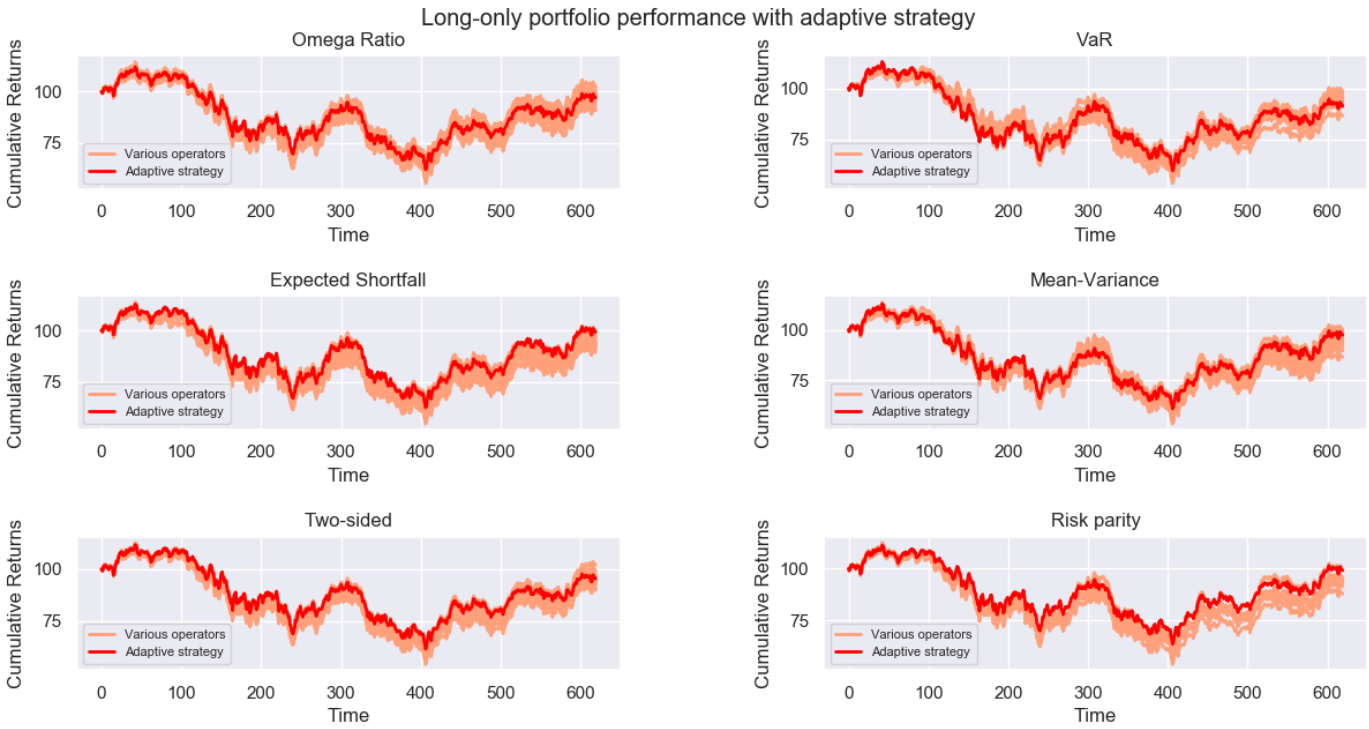


Figure 4.15: Rolling-window backtest of the adaptive strategy on FTSE MIB index with $K = 10$, $lb = 0.05$ and $ub = 0.15$

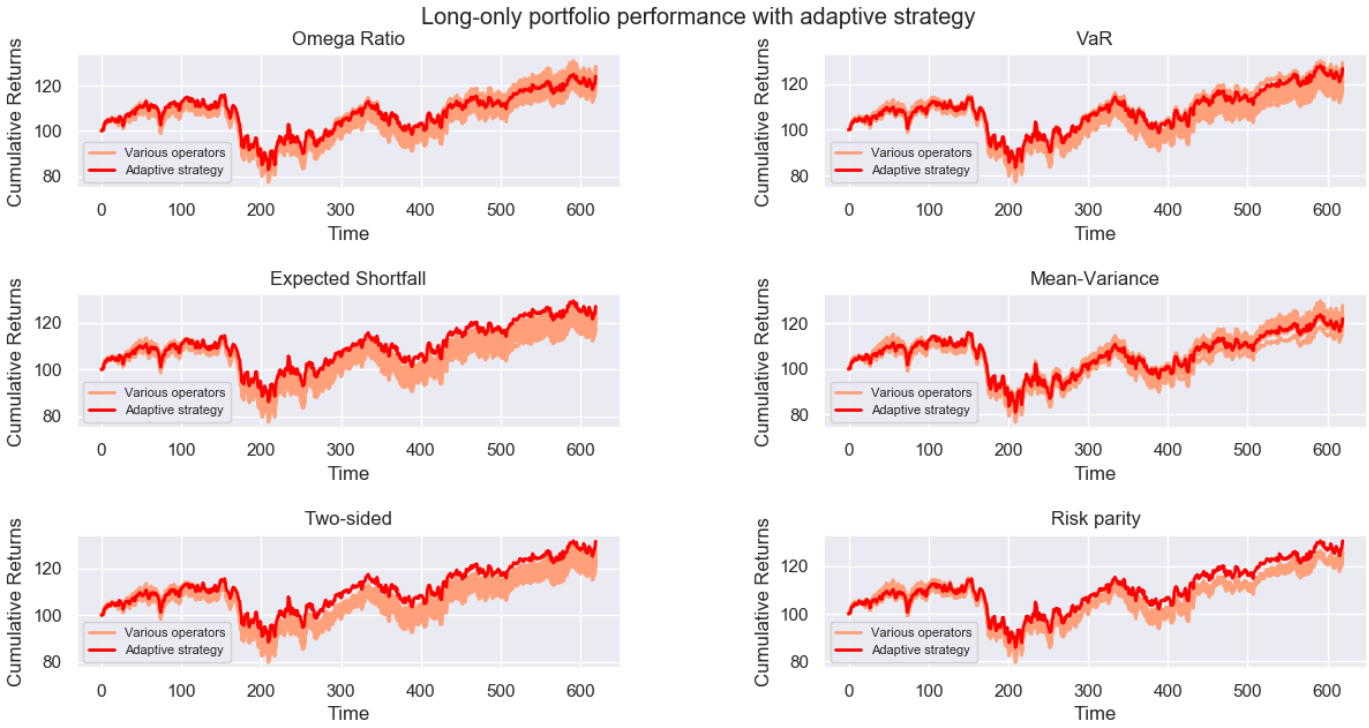


Figure 4.16: Rolling-window backtest of the adaptive strategy on CAC 40 index with $K = 10$, $lb = 0.05$ and $ub = 0.15$

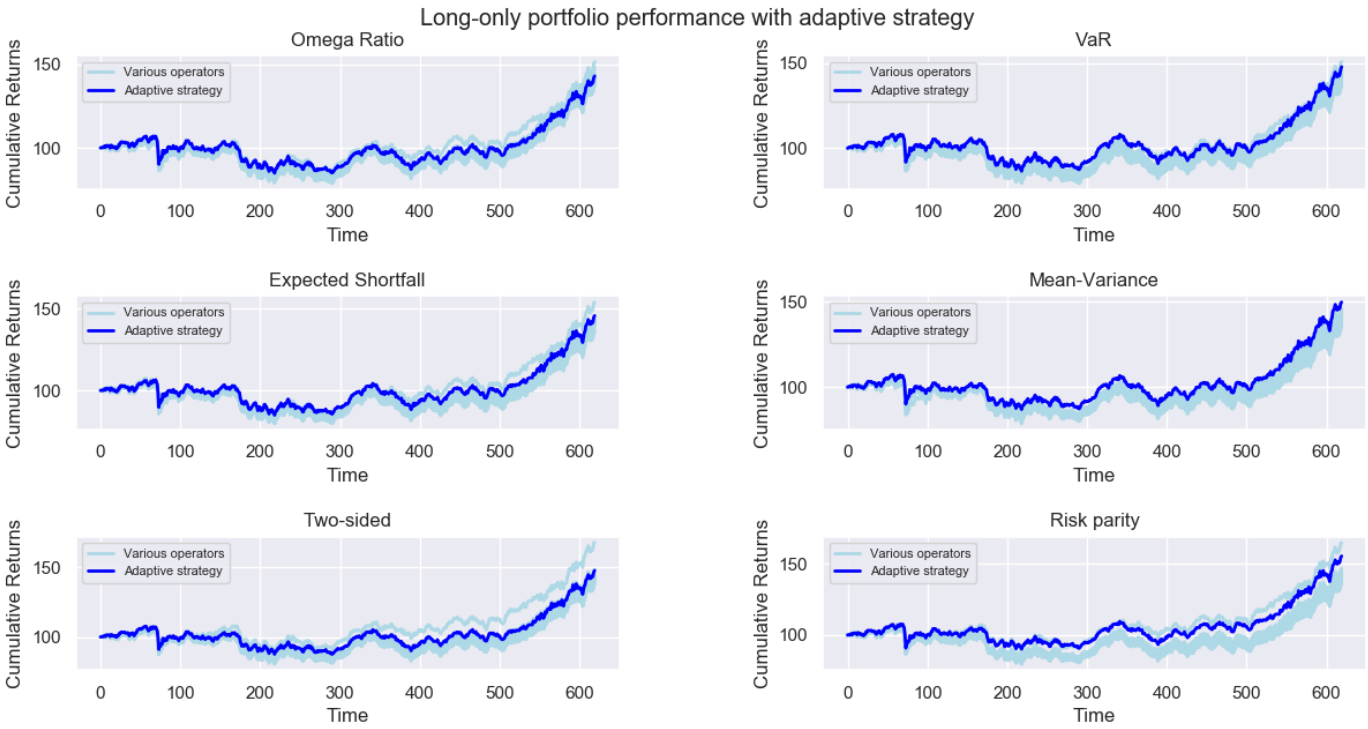


Figure 4.17: Rolling-window backtest of the adaptive strategy on Nikkei 225 index with $K = 20$, $lb = 0.04$ and $ub = 0.12$

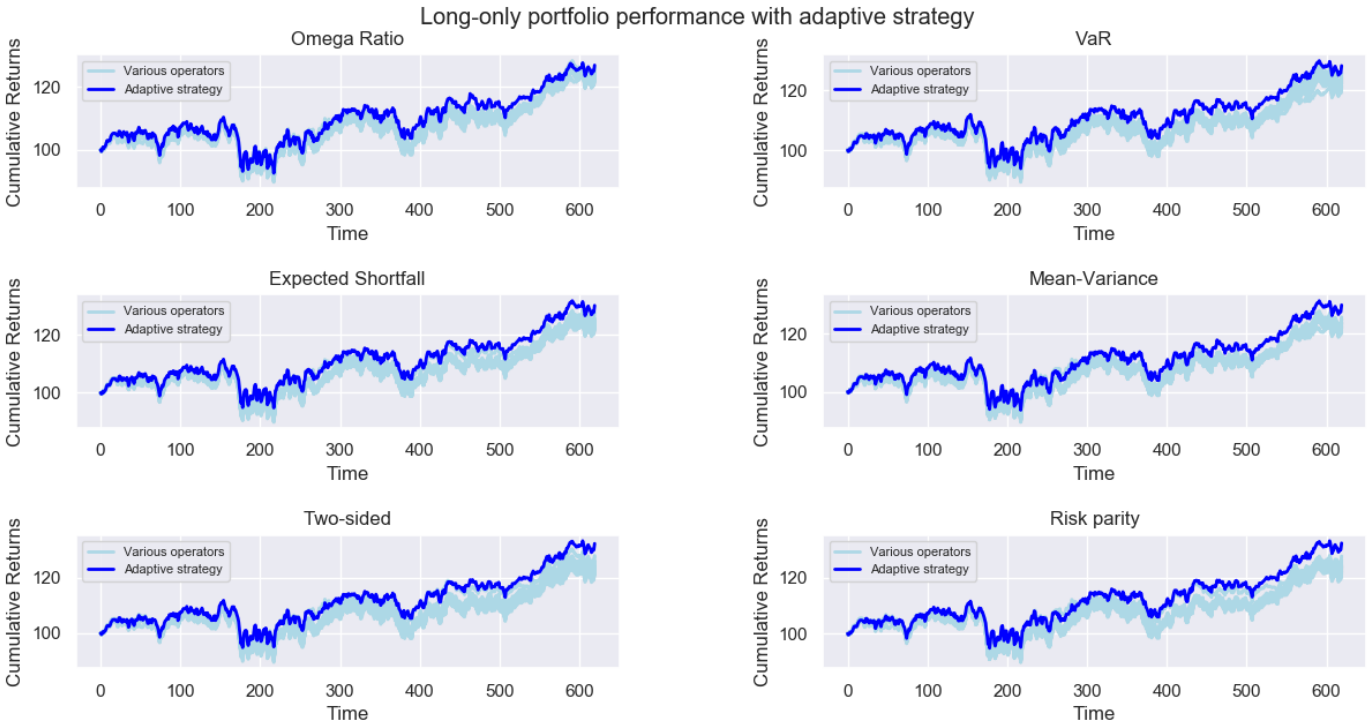


Figure 4.18: Rolling-window backtest of the adaptive strategy on FTSE 100 index with $K = 20$, $lb = 0.04$ and $ub = 0.12$

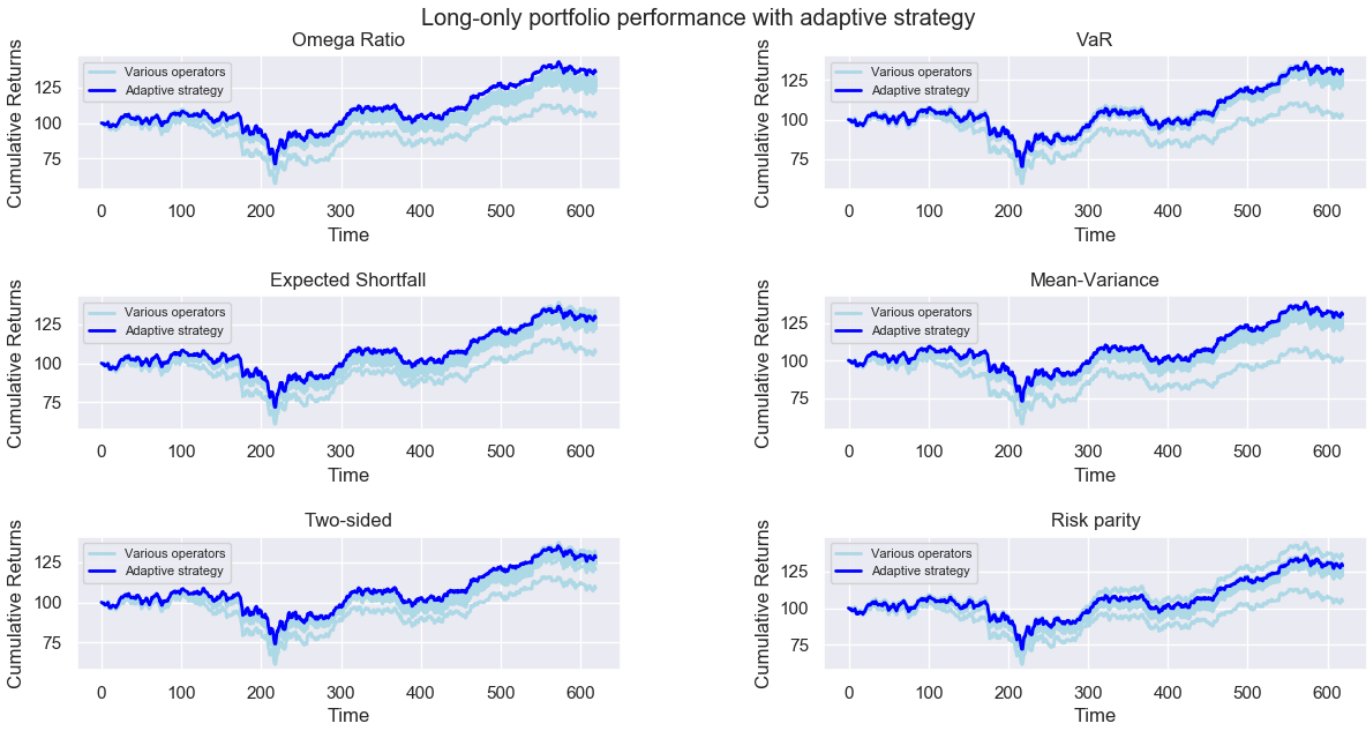


Figure 4.19: Rolling-window backtest of the adaptive strategy on Hang Seng index with $K = 20$, $lb = 0.04$ and $ub = 0.12$

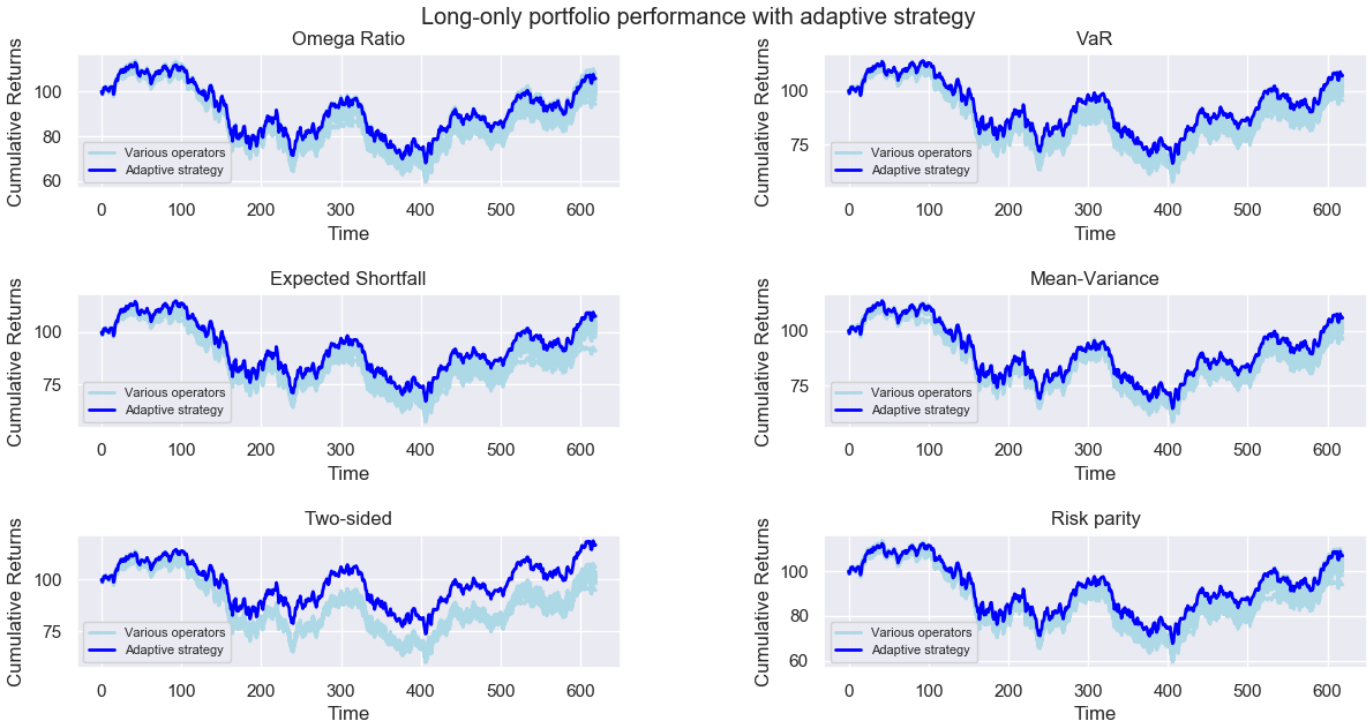


Figure 4.20: Rolling-window backtest of the adaptive strategy on FTSE MIB index with $K = 20$, $lb = 0.04$ and $ub = 0.12$

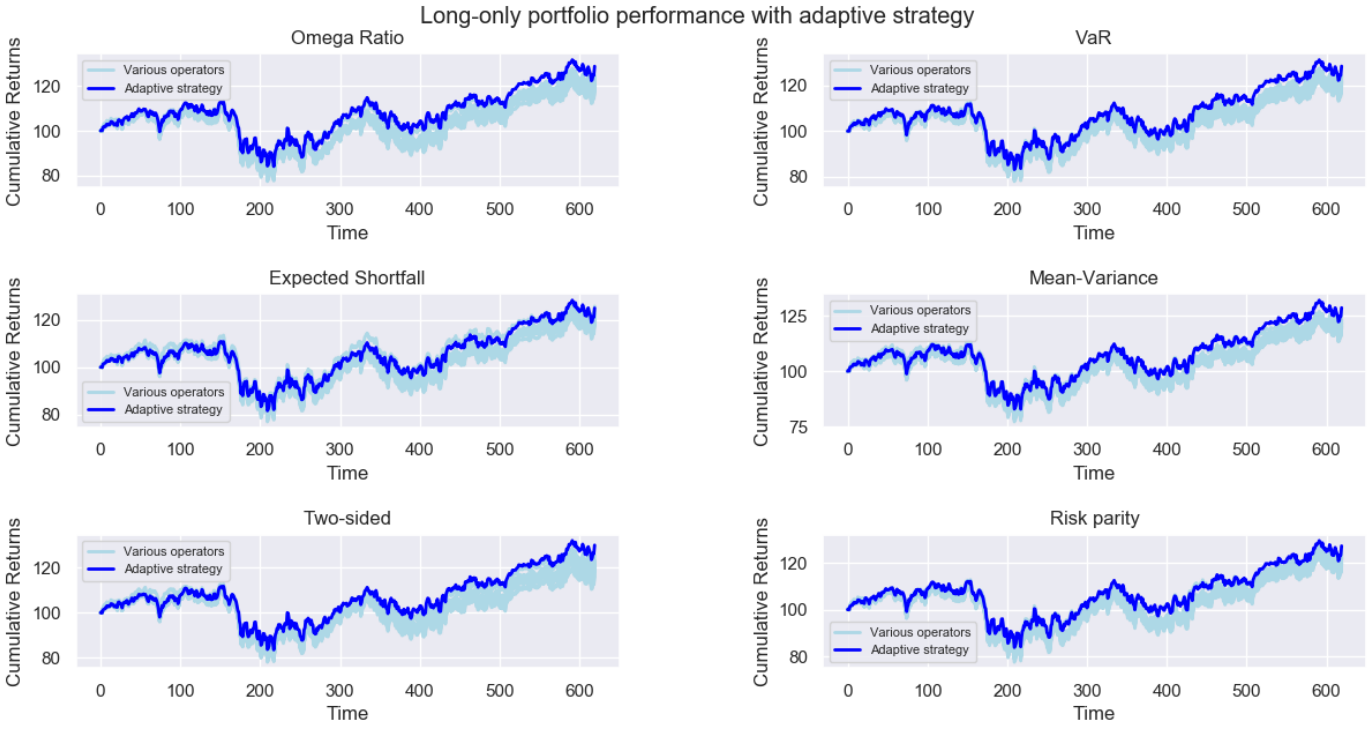


Figure 4.21: Rolling-window backtest of the adaptive strategy on CAC 40 index with $K = 20$, $lb = 0.04$ and $ub = 0.12$

Ratio of portfolios constructed with the Omega Ratio and with the ERC approach are, on average, a bit below one. The remaining risk measures display a moderately below-average performance across all datasets. Unsurprisingly, the mean-variance approach performs modestly, though also $VaR_{95\%}$ and $ES_{95\%}$ strategies present comparable performance metrics; moreover, we note that one portfolio constructed from a subset of the FTSE MIB regularly exhibits a negative Sharpe Ratio, with the exception of Omega Ratio portfolios, which always display positive values.

As far as transaction costs are involved, there is little evidence in favor of any strategy, including the adaptive one, as turnover values are not particularly consistent; actually, we do not observe particular patterns across datasets, as Tables 4.5-4.10 show pretty clearly that some strategies can be at times expensive, at others quite cheap. Similar results are observed for different risk measures, controlling for the input dataset we pass to each model.

Finally, we observe a more robust relationship between K, i.e. the subset of assets included in the portfolio and its turnover, which is substantially larger turnover for portfolios with $K = 20$. A lower turnover is an attractive feature and actually it could be also influenced by input sensitivity (which is a well-known shortcoming of mean-variance portfolios) or by lower and upper bounds constraint. By enforcing strict floor and ceiling constraints, we avoid indeed large swings in the composition of the portfolio due to rebalancing operations; as a result, small perturbations in the values of the estimates never lead to innatural and highly concentrated mean-variance portfolios. The remaining risk measures are more robust to input estimates; yet, they benefit in terms of transaction costs from constraining the amount invested in each asset. The flip side of this approach, of course, is that it may be too limiting for outperforming strategies, leading in practice to portfolios that resemble each other.

	$\hat{\mu^i}$		$\hat{\sigma^i}$		$\hat{SR^i}$		$Turnover$	
Nikkei 225	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$

OPX	0.1901	0.1574	0.2040	0.1927	0.9322	0.8165	0.0809	0.1541
UX	0.1915	0.1599	0.2053	0.1911	0.9329	0.8368	0.0936	0.1663
HX	0.2074	0.1653	0.2043	0.1928	1.0154	0.8574	0.0740	0.1541
LX	0.1874	0.1591	0.2061	0.1932	0.9093	0.8235	0.0746	0.1765
QBX	0.1931	0.1785	0.2032	0.1878	0.9502	0.9505	0.1010	0.1906
TPX	0.1896	0.1644	0.2055	0.1890	0.9228	0.8700	0.0831	0.1777
AX	0.1890	0.1747	0.2096	0.1899	0.9017	0.9202	0.0815	0.1839
GX	0.1938	0.1644	0.2020	0.1917	0.9594	0.8575	0.0739	0.1627
SBX	0.1778	0.1535	0.2086	0.1918	0.8521	0.8003	0.1038	0.1613
AVX	0.1807	0.1580	0.2075	0.1924	0.8712	0.8212	0.0629	0.1971
BLX	0.2174	0.1533	0.1987	0.1944	1.0938	0.7884	0.0928	0.1575
FX	0.1953	0.1507	0.2021	0.1925	0.9663	0.7829	0.0876	0.1688
GUX	0.2309	0.1643	0.2025	0.1895	1.1401	0.8671	0.0844	0.1640
TPX	0.2017	0.1540	0.2082	0.1928	0.9689	0.7989	0.0693	0.1672
LNX	0.1903	0.1484	0.2130	0.1971	0.8934	0.7531	0.0755	0.1464
DBX	0.1790	0.1673	0.2180	0.1897	0.8212	0.8817	0.0726	0.1532
UNDX	0.1927	0.1467	0.2076	0.1902	0.9286	0.7711	0.1081	0.1613
FR	0.2032	0.1557	0.2077	0.1942	0.9782	0.8014	0.0956	0.2163
SPX	0.1771	0.1602	0.2166	0.1908	0.8175	0.8393	0.1192	0.1873
PNX	0.1614	0.1621	0.2132	0.1909	0.7570	0.8495	0.0706	0.2308
Adaptive	0.2136	0.1505	0.2005	0.1889	1.0653	0.7966	0.1327	0.1550
FTSE 100	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.0986	0.0958	0.1506	0.1687	0.6547	0.5682	0.0780	0.1850
UX	0.0947	0.1034	0.1515	0.1650	0.6254	0.6264	0.0754	0.1636
HX	0.0985	0.0969	0.1505	0.1647	0.6547	0.5882	0.0655	0.1850
LX	0.0963	0.1005	0.1513	0.1657	0.6364	0.6067	0.0670	0.1626
QBX	0.0972	0.1019	0.1491	0.1653	0.6518	0.6166	0.1078	0.1622
TPX	0.0957	0.0977	0.1538	0.1644	0.6221	0.5944	0.0823	0.1780
AX	0.0939	0.1004	0.1492	0.1660	0.6292	0.6050	0.1072	0.2000
GX	0.0994	0.1011	0.1524	0.1639	0.6522	0.6170	0.0941	0.1721
SBX	0.0944	0.0985	0.1548	0.1627	0.6096	0.6050	0.0823	0.1835
AVX	0.0919	0.1078	0.1550	0.1634	0.5925	0.6595	0.0728	0.1795
BLX	0.0972	0.1083	0.1512	0.1626	0.6432	0.6659	0.0859	0.1523
FX	0.0987	0.1011	0.1516	0.1630	0.6507	0.6204	0.0831	0.1828
GUX	0.0853	0.0978	0.1560	0.1665	0.5469	0.5876	0.1067	0.1564
TPX	0.0918	0.1056	0.1542	0.1661	0.5955	0.6356	0.1115	0.1923
LNX	0.0927	0.0974	0.1547	0.1654	0.5993	0.5885	0.0654	0.2030
DBX	0.0867	0.1007	0.1532	0.1661	0.5657	0.6058	0.0801	0.1547
UNDX	0.0963	0.1032	0.1534	0.1648	0.6279	0.6259	0.0906	0.2130
FR	0.1003	0.1025	0.1501	0.1637	0.6678	0.6260	0.1237	0.1750
SPX	0.0929	0.1030	0.1516	0.1666	0.6129	0.6184	0.1131	0.2207
PNX	0.0844	0.0984	0.1561	0.1642	0.5406	0.5994	0.0783	0.1758
Adaptive	0.1071	0.0959	0.1478	0.1651	0.7251	0.5809	0.1043	0.1608
Hang Seng	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.0557	0.1246	0.2487	0.2082	0.2238	0.5983	0.0686	0.1351
UX	0.0411	0.1201	0.2377	0.2128	0.1729	0.5644	0.0812	0.1583
HX	0.0484	0.1297	0.2497	0.2137	0.1940	0.6068	0.0705	0.1351
LX	0.0532	0.1056	0.2450	0.2136	0.2171	0.4944	0.0909	0.1714
QBX	0.0545	0.1270	0.2512	0.2092	0.2170	0.6071	0.0712	0.1705
TPX	0.0467	0.1079	0.2504	0.2093	0.1865	0.5156	0.0806	0.1580
AX	0.0460	0.1160	0.2496	0.2094	0.1844	0.5539	0.0785	0.1530
GX	0.0644	0.1225	0.2437	0.2095	0.2641	0.5850	0.1105	0.1520
SBX	0.0573	0.1170	0.2411	0.2154	0.2375	0.5432	0.0831	0.2062
AVX	0.0507	0.1219	0.2496	0.2110	0.2033	0.5777	0.0950	0.1801
BLX	0.0567	0.1157	0.2491	0.2125	0.2277	0.5448	0.0601	0.1793
FX	0.0551	0.1273	0.2489	0.2087	0.2213	0.6098	0.0696	0.1408
GUX	0.0338	0.1197	0.2461	0.2124	0.1374	0.5634	0.0850	0.1821
TPX	0.0524	0.1246	0.2506	0.2119	0.2092	0.5881	0.1332	0.1740
LNX	0.0417	0.1205	0.2442	0.2132	0.1709	0.5654	0.0789	0.2067
DBX	0.0444	0.1111	0.2386	0.2137	0.1862	0.5200	0.0717	0.1617
UNDX	0.0470	0.1309	0.2397	0.2193	0.1962	0.5970	0.1443	0.1621
FR	0.0604	0.1366	0.2418	0.2173	0.2497	0.6287	0.0753	0.2295
SPX	0.0392	0.1319	0.2505	0.2110	0.1564	0.6253	0.0750	0.1690
PNX	0.0379	0.1228	0.2394	0.2096	0.1581	0.5859	0.0952	0.1850
Adaptive	0.0812	0.1353	0.2347	0.2153	0.3458	0.6283	0.0845	0.1981
FTSE MIB	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.0143	0.0383	0.2770	0.3021	0.0516	0.1269	0.0734	0.1647
UX	0.0228	0.0638	0.2905	0.3014	0.0785	0.2116	0.1132	0.1905
HX	0.0134	0.0587	0.2857	0.3018	0.0469	0.1945	0.0626	0.1647
LX	0.0411	0.0484	0.2930	0.3028	0.1403	0.1599	0.0739	0.1509
QBX	0.0296	0.0448	0.2786	0.2998	0.1061	0.1495	0.1036	0.1496
TPX	0.0487	0.0378	0.2862	0.3031	0.1703	0.1248	0.0975	0.1833
AX	0.0398	0.0555	0.2808	0.2999	0.1416	0.1850	0.1059	0.1669
GX	0.0474	0.0505	0.2776	0.2992	0.1708	0.1689	0.0652	0.2204
SBX	0.0360	0.0585	0.2793	0.3010	0.1288	0.1945	0.0897	0.1450
AVX	0.0302	0.0465	0.2796	0.3012	0.1080	0.1543	0.0795	0.1836
BLX	0.0302	0.0515	0.2743	0.3065	0.1100	0.1681	0.0806	0.1591
FX	0.0482	0.0435	0.2757	0.3020	0.1747	0.1440	0.0739	0.2009

GUX	0.0368	0.0541	0.2757	0.2978	0.1334	0.1817	0.0736	0.1673
TPX	0.0407	0.0663	0.2769	0.3019	0.1471	0.2195	0.0787	0.1578
LNX	0.0384	0.0426	0.3156	0.2985	0.1216	0.1428	0.0721	0.1614
DBX	0.0103	0.0536	0.3154	0.3002	0.0326	0.1787	0.0805	0.1717
UNDX	0.0389	0.0786	0.2826	0.3040	0.1375	0.2586	0.1272	0.1831
FR	0.0067	0.0390	0.2847	0.2990	0.0237	0.1304	0.0914	0.2491
SPX	0.0252	0.0302	0.2971	0.3007	0.0848	0.1005	0.0908	0.1797
PNX	0.0125	0.0532	0.3088	0.2988	0.0406	0.1779	0.0838	0.1499
Adaptive	0.0229	0.0469	0.2662	0.2889	0.0861	0.1624	0.1522	0.1721
CAC 40	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.1055	0.1014	0.2196	0.2313	0.4805	0.4385	0.0941	0.1672
UX	0.1060	0.1042	0.2204	0.2281	0.4807	0.4568	0.0755	0.1816
HX	0.1190	0.1064	0.2132	0.2309	0.5580	0.4609	0.0922	0.1672
LX	0.0955	0.1036	0.2228	0.2288	0.4285	0.4526	0.0654	0.1933
QBX	0.1234	0.0958	0.2119	0.2337	0.5824	0.4098	0.1090	0.1560
TPX	0.1101	0.1119	0.2252	0.2279	0.4888	0.4911	0.0701	0.1861
AX	0.1022	0.0942	0.2265	0.2350	0.4514	0.4006	0.0824	0.1600
GX	0.1141	0.1117	0.2124	0.2334	0.5370	0.4784	0.0658	0.1830
SBX	0.1069	0.1169	0.2261	0.2319	0.4729	0.5042	0.1016	0.1423
AVX	0.1180	0.1009	0.2200	0.2309	0.5364	0.4371	0.0853	0.2058
BLX	0.1222	0.1092	0.2155	0.2337	0.5672	0.4672	0.0804	0.1693
FX	0.1119	0.1069	0.2120	0.2307	0.5280	0.4632	0.0803	0.1657
GUX	0.0965	0.0986	0.2320	0.2340	0.4160	0.4216	0.0795	0.1789
TPX	0.1219	0.1147	0.2147	0.2291	0.5680	0.5008	0.1029	0.1791
LNX	0.1041	0.1055	0.2354	0.2311	0.4422	0.4566	0.0911	0.1566
DBX	0.1082	0.1064	0.2384	0.2328	0.4538	0.4571	0.0886	0.1896
UNDX	0.1112	0.1095	0.2219	0.2324	0.5011	0.4710	0.0926	0.1930
FR	0.1259	0.1056	0.2201	0.2322	0.5722	0.4549	0.0797	0.1947
SPX	0.1254	0.1121	0.2236	0.2301	0.5608	0.4873	0.0644	0.1573
PNX	0.1107	0.1013	0.2227	0.2300	0.4969	0.4403	0.1061	0.1772
Adaptive	0.1109	0.1140	0.2159	0.2269	0.5134	0.5025	0.0800	0.1992

Table 4.5: Out-of-sample portfolio metrics. Risk measure: *Omega ratio*. Tests with cardinality and upper/lower bound constraints, $\varepsilon = 1e-001$.

	$\hat{\mu^i}$		$\hat{\sigma^i}$		$\hat{SR^i}$		$Turnover$	
Nikkei 225	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.1893	0.1606	0.2091	0.1941	0.9283	0.8335	0.0694	0.1494
UX	0.2099	0.1781	0.2082	0.1894	1.0225	0.9320	0.0974	0.1826
HX	0.2140	0.1495	0.2051	0.1918	1.0476	0.7752	0.0956	0.1494
LX	0.1734	0.1589	0.2119	0.1943	0.8413	0.8222	0.0712	0.1737
QBX	0.2146	0.1452	0.2034	0.1945	1.0563	0.7733	0.0978	0.1545
TPX	0.2134	0.1629	0.2104	0.1888	1.0386	0.8621	0.0775	0.1799
AX	0.1890	0.1641	0.2058	0.1921	0.9017	0.8642	0.0831	0.1749
GX	0.1988	0.1626	0.2042	0.1922	0.9842	0.8480	0.0797	0.1612
SBX	0.1714	0.1571	0.2053	0.1912	0.8216	0.8187	0.0802	0.1596
AVX	0.1902	0.1699	0.2105	0.1909	0.9166	0.8828	0.0775	0.1593
BLX	0.1956	0.1715	0.2117	0.1904	0.9843	0.8819	0.0928	0.1663
FX	0.1984	0.1600	0.2086	0.1920	0.9818	0.8310	0.0897	0.1706
GUX	0.2031	0.1537	0.2116	0.1912	1.0026	0.8113	0.0933	0.1692
TPX	0.1911	0.1619	0.2122	0.1934	0.9177	0.8399	0.0600	0.1975
LNX	0.1913	0.1530	0.2140	0.1923	0.8981	0.7763	0.1009	0.1624
DBX	0.1458	0.1518	0.2259	0.1916	0.6691	0.8001	0.0849	0.1738
UNDX	0.1920	0.1540	0.2158	0.1945	0.9251	0.8094	0.1325	0.1702
FR	0.1916	0.1601	0.2104	0.1900	0.9224	0.8241	0.0909	0.2399
SPX	0.1659	0.1703	0.2166	0.1888	0.7657	0.8925	0.0796	0.1730
PNX	0.1575	0.1637	0.2139	0.1917	0.7385	0.8577	0.0720	0.1650
Adaptive	0.2033	0.1650	0.2013	0.1913	1.0143	0.8737	0.1144	0.1478
FTSE 100	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.0886	0.0967	0.1516	0.1669	0.5882	0.5730	0.0919	0.1651
UX	0.0956	0.0926	0.1491	0.1678	0.6309	0.5612	0.0602	0.1604
HX	0.1051	0.1046	0.1473	0.1621	0.6987	0.6351	0.0883	0.1651
LX	0.0980	0.0981	0.1474	0.1641	0.6473	0.5921	0.0962	0.1908
QBX	0.1141	0.1013	0.1451	0.1616	0.7657	0.6127	0.0695	0.1871
TPX	0.0995	0.0925	0.1518	0.1657	0.6472	0.5629	0.0714	0.1788
AX	0.1065	0.1066	0.1495	0.1673	0.7138	0.6424	0.0991	0.1400
GX	0.0911	0.1045	0.1488	0.1644	0.5974	0.6377	0.0826	0.2076
SBX	0.1011	0.0974	0.1511	0.1659	0.6529	0.5982	0.0873	0.1991
AVX	0.1000	0.1030	0.1488	0.1657	0.6450	0.6301	0.0711	0.1786
BLX	0.0941	0.1013	0.1478	0.1648	0.6225	0.6230	0.0675	0.1612
FX	0.1004	0.0999	0.1469	0.1640	0.6622	0.6132	0.1072	0.1602
GUX	0.0939	0.0980	0.1512	0.1640	0.6020	0.5888	0.1183	0.1818
TPX	0.0969	0.0965	0.1513	0.1669	0.6285	0.5811	0.0677	0.1890
LNX	0.1011	0.0957	0.1515	0.1656	0.6538	0.5784	0.0633	0.1573

DBX	0.0758	0.0994	0.1596	0.1655	0.4947	0.5982	0.0988	0.1489
UNDX	0.0898	0.1036	0.1524	0.1643	0.5857	0.6287	0.1056	0.1884
FR	0.0952	0.1002	0.1567	0.1659	0.6341	0.6119	0.0802	0.2439
SPX	0.0973	0.1013	0.1513	0.1640	0.6419	0.6079	0.0979	0.1570
PNX	0.0921	0.0960	0.1543	0.1645	0.5904	0.5847	0.0975	0.1750
Adaptive	0.1071	0.0997	0.1466	0.1637	0.7246	0.6042	0.0772	0.1776
Hang Seng	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.0376	0.1153	0.2377	0.2084	0.1510	0.5537	0.0740	0.1606
UX	0.0669	0.1194	0.2299	0.2156	0.2812	0.5611	0.0898	0.1931
HX	0.0585	0.1127	0.2382	0.2093	0.2342	0.5274	0.1117	0.1606
LX	0.0471	0.1100	0.2437	0.2112	0.1924	0.5150	0.1090	0.2172
QBX	0.0567	0.1130	0.2361	0.2101	0.2256	0.5402	0.0678	0.1773
TPX	0.0316	0.1228	0.2349	0.2105	0.1261	0.5865	0.0755	0.1546
AX	0.0614	0.1221	0.2401	0.2096	0.2459	0.5831	0.0859	0.1735
GX	0.0421	0.1033	0.2372	0.2178	0.1727	0.4932	0.0620	0.1710
SBX	0.0351	0.1192	0.2292	0.2158	0.1456	0.5533	0.0744	0.1745
AVX	0.0479	0.1162	0.2380	0.2168	0.1918	0.5505	0.0767	0.1327
BLX	0.0459	0.1117	0.2357	0.2047	0.1842	0.5258	0.0628	0.1587
FX	0.0435	0.1233	0.2359	0.2106	0.1749	0.5907	0.0793	0.1707
GUX	0.0401	0.1077	0.2443	0.2056	0.1629	0.5069	0.0877	0.1629
TPX	0.0527	0.1136	0.2346	0.2158	0.2103	0.5362	0.1043	0.1924
LNX	0.0374	0.1056	0.2522	0.2118	0.1533	0.4953	0.0972	0.1735
DBX	0.0547	0.1204	0.2518	0.2104	0.2292	0.5634	0.0602	0.1647
UNDX	0.0543	0.1165	0.2402	0.2080	0.2264	0.5315	0.0941	0.1710
FR	0.0593	0.1263	0.2381	0.2155	0.2454	0.5812	0.0690	0.1855
SPX	0.0356	0.1157	0.2411	0.2093	0.1422	0.5485	0.1139	0.1509
PNX	0.0534	0.1222	0.2577	0.2145	0.2229	0.5828	0.0824	0.2117
Adaptive	0.0544	0.1177	0.2328	0.2138	0.2319	0.5468	0.1146	0.1938
FTSE MIB	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.0213	0.0445	0.2845	0.3048	0.0769	0.1472	0.1060	0.1981
UX	0.0038	0.0469	0.2917	0.3022	0.0130	0.1554	0.0924	0.1563
HX	0.0270	0.0645	0.2821	0.2985	0.0946	0.2137	0.0780	0.1981
LX	0.0323	0.0579	0.2870	0.3007	0.1104	0.1911	0.0718	0.1535
QBX	0.0130	0.0601	0.2862	0.3008	0.0467	0.2006	0.0968	0.1554
TPX	0.0146	0.0515	0.2834	0.2985	0.0509	0.1698	0.0788	0.1667
AX	0.0232	0.0605	0.2864	0.3039	0.0828	0.2019	0.0655	0.1595
GX	0.0345	0.0537	0.2861	0.3014	0.1244	0.1795	0.0980	0.1793
SBX	0.0143	0.0559	0.3018	0.2985	0.0511	0.1857	0.0993	0.1842
AVX	0.0154	0.0522	0.2790	0.2997	0.0551	0.1734	0.0916	0.1832
BLX	0.0072	0.0515	0.2894	0.2999	0.0262	0.1681	0.1125	0.1863
FX	0.0334	0.0359	0.2837	0.3031	0.1211	0.1189	0.1038	0.1852
GUX	0.0199	0.0547	0.2915	0.2984	0.0722	0.1838	0.0818	0.2018
TPX	0.0337	0.0383	0.2869	0.3033	0.1219	0.1268	0.0816	0.1853
LNX	0.0266	0.0466	0.2998	0.3059	0.0843	0.1560	0.0996	0.2226
DBX	0.0250	0.0498	0.3203	0.3011	0.0793	0.1660	0.0922	0.1786
UNDX	0.0230	0.0526	0.2907	0.3034	0.0812	0.1729	0.1258	0.1778
FR	-0.0151	0.0674	0.2963	0.3031	-0.0531	0.2253	0.1229	0.2699
SPX	0.0130	0.0428	0.3012	0.2998	0.0438	0.1422	0.0806	0.1865
PNX	0.0093	0.0516	0.3051	0.3022	0.0300	0.1726	0.0640	0.1774
Adaptive	0.0043	0.0527	0.2873	0.2921	0.0160	0.1824	0.0830	0.1786
CAC 40	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.1050	0.1086	0.2188	0.2279	0.4778	0.4695	0.0646	0.1622
UX	0.1033	0.1076	0.2269	0.2308	0.4689	0.4718	0.0751	0.1606
HX	0.1109	0.1074	0.2182	0.2296	0.5199	0.4651	0.0896	0.1622
LX	0.1055	0.1135	0.2279	0.2310	0.4736	0.4962	0.0885	0.1739
QBX	0.1268	0.1075	0.2136	0.2303	0.5983	0.4598	0.0886	0.1798
TPX	0.1092	0.1042	0.2226	0.2268	0.4848	0.4573	0.0970	0.1690
AX	0.1110	0.1016	0.2153	0.2318	0.4902	0.4323	0.0895	0.1936
GX	0.1284	0.1142	0.2187	0.2289	0.6043	0.4892	0.0892	0.1687
SBX	0.0924	0.1065	0.2316	0.2321	0.4089	0.4594	0.0668	0.1726
AVX	0.1144	0.1053	0.2169	0.2310	0.5198	0.4562	0.0869	0.1741
BLX	0.1119	0.1012	0.2259	0.2314	0.5193	0.4331	0.0873	0.1691
FX	0.1051	0.1052	0.2202	0.2318	0.4957	0.4559	0.1075	0.1582
GUX	0.1082	0.0982	0.2261	0.2317	0.4665	0.4196	0.0690	0.2045
TPX	0.1172	0.1145	0.2206	0.2305	0.5458	0.4998	0.0952	0.1604
LNX	0.1027	0.1011	0.2359	0.2326	0.4365	0.4376	0.0710	0.1945
DBX	0.1066	0.1038	0.2470	0.2331	0.4472	0.4458	0.0866	0.1529
UNDX	0.1004	0.1082	0.2240	0.2303	0.4526	0.4657	0.1202	0.1551
FR	0.1177	0.1133	0.2353	0.2294	0.5346	0.4878	0.0891	0.1945
SPX	0.1066	0.1050	0.2409	0.2293	0.4768	0.4562	0.0904	0.1669
PNX	0.1114	0.1113	0.2261	0.2301	0.5001	0.4838	0.0777	0.1866
Adaptive	0.1184	0.1128	0.2114	0.2246	0.5482	0.4974	0.1151	0.1584

Table 4.6: Out-of-sample portfolio metrics. Risk measure: $VaR_{95\%}$. Tests with cardinality and upper/lower bound constraints, $\varepsilon = 1e-001$.

	$\hat{\mu^i}$		$\hat{\sigma^i}$		$\hat{SR^i}$		$Turnover$	
Nikkei 225	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.1973	0.1629	0.2069	0.1908	0.9675	0.8453	0.0781	0.1478
UX	0.1879	0.1620	0.2084	0.1900	0.9152	0.8476	0.1051	0.2007
HX	0.1966	0.1644	0.2114	0.1915	0.9623	0.8528	0.0822	0.1478
LX	0.1787	0.1679	0.2125	0.1901	0.8672	0.8690	0.0903	0.1701
QBX	0.2381	0.1469	0.2050	0.1937	1.1717	0.7823	0.1042	0.1819
TPX	0.1935	0.1637	0.2046	0.1909	0.9419	0.8659	0.0772	0.1674
AX	0.2101	0.1583	0.2044	0.1965	1.0026	0.8338	0.1064	0.1685
GX	0.1912	0.1421	0.2094	0.1959	0.9467	0.7411	0.0695	0.2163
SBX	0.2184	0.1552	0.2055	0.1947	1.0469	0.8089	0.0838	0.1398
AVX	0.1902	0.1579	0.2090	0.1932	0.9170	0.8204	0.0830	0.1769
BLX	0.2042	0.1659	0.2078	0.1923	1.0278	0.8533	0.0745	0.1813
FX	0.1983	0.1655	0.2032	0.1912	0.9813	0.8595	0.0894	0.1576
GUX	0.1956	0.1602	0.2126	0.1932	0.9655	0.8453	0.0820	0.1683
TPX	0.1943	0.1576	0.2058	0.1909	0.9332	0.8175	0.0901	0.1522
LNX	0.1777	0.1638	0.2096	0.1913	0.8341	0.8314	0.0817	0.1956
DBX	0.1462	0.1559	0.2268	0.1954	0.6706	0.8215	0.0650	0.2086
UNDX	0.1820	0.1537	0.2094	0.1924	0.8770	0.8083	0.1047	0.1539
FR	0.1870	0.1566	0.2085	0.1913	0.9003	0.8062	0.0967	0.1838
SPX	0.1872	0.1584	0.2125	0.1920	0.8640	0.8301	0.1070	0.1626
PNX	0.1858	0.1602	0.2165	0.1930	0.8713	0.8391	0.0785	0.1719
Adaptive	0.2239	0.1585	0.1978	0.1891	1.1167	0.8393	0.1049	0.1607
FTSE 100	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.0975	0.1070	0.1498	0.1652	0.6478	0.6343	0.0768	0.1505
UX	0.1006	0.0968	0.1525	0.1651	0.6643	0.5869	0.0847	0.1637
HX	0.0935	0.0972	0.1482	0.1636	0.6211	0.5899	0.0715	0.1505
LX	0.1051	0.1044	0.1506	0.1632	0.6947	0.6298	0.0932	0.1558
QBX	0.1007	0.0994	0.1475	0.1645	0.6757	0.6016	0.0771	0.1666
TPX	0.1073	0.1025	0.1471	0.1660	0.6979	0.6235	0.0746	0.1425
AX	0.0928	0.1019	0.1506	0.1622	0.6218	0.6139	0.0861	0.1518
GX	0.1004	0.1132	0.1490	0.1628	0.6584	0.6908	0.0626	0.1980
SBX	0.0901	0.1022	0.1531	0.1667	0.5820	0.6282	0.0581	0.1433
AVX	0.1015	0.1047	0.1484	0.1644	0.6549	0.6406	0.0811	0.1544
BLX	0.1010	0.1042	0.1493	0.1634	0.6681	0.6407	0.0818	0.1743
FX	0.1018	0.1041	0.1509	0.1660	0.6713	0.6387	0.0849	0.1539
GUX	0.1002	0.1006	0.1467	0.1653	0.6420	0.6045	0.0872	0.1792
TPX	0.1060	0.0987	0.1477	0.1668	0.6875	0.5939	0.0837	0.1583
LNX	0.0996	0.0998	0.1530	0.1627	0.6440	0.6034	0.0732	0.1814
DBX	0.0801	0.1061	0.1602	0.1627	0.5230	0.6387	0.0881	0.1485
UNDX	0.1013	0.0973	0.1481	0.1608	0.6608	0.5904	0.0813	0.1810
FR	0.0893	0.1006	0.1535	0.1669	0.5948	0.6143	0.1234	0.2368
SPX	0.0884	0.1072	0.1525	0.1644	0.5832	0.6431	0.0829	0.2005
PNX	0.0856	0.0995	0.1519	0.1652	0.5483	0.6057	0.0687	0.1939
Adaptive	0.1154	0.1062	0.1465	0.1607	0.7814	0.6434	0.0704	0.1357
Hang Seng	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.0553	0.1119	0.2328	0.2229	0.2222	0.5376	0.0839	0.1820
UX	0.0488	0.1154	0.2378	0.2139	0.2053	0.5423	0.1028	0.1726
HX	0.0500	0.1129	0.2375	0.2128	0.2003	0.5283	0.0771	0.1820
LX	0.0313	0.1281	0.2439	0.2072	0.1278	0.5998	0.0803	0.1546
QBX	0.0487	0.1169	0.2311	0.2096	0.1939	0.5590	0.1123	0.1573
TPX	0.0442	0.1118	0.2426	0.2140	0.1767	0.5343	0.0989	0.2185
AX	0.0425	0.1211	0.2340	0.2105	0.1705	0.5784	0.0907	0.1712
GX	0.0462	0.1051	0.2350	0.2097	0.1897	0.5016	0.0941	0.1735
SBX	0.0470	0.1384	0.2358	0.2117	0.1951	0.6422	0.0756	0.2038
AVX	0.0410	0.1132	0.2361	0.2142	0.1644	0.5366	0.0747	0.1792
BLX	0.0358	0.1202	0.2405	0.2120	0.1436	0.5656	0.0764	0.1620
FX	0.0368	0.1215	0.2335	0.2131	0.1478	0.5820	0.0910	0.1704
GUX	0.0466	0.1094	0.2440	0.2128	0.1895	0.5153	0.0759	0.1917
TPX	0.0305	0.1122	0.2399	0.2127	0.1216	0.5295	0.0862	0.1612
LNX	0.0165	0.1111	0.2451	0.2109	0.0675	0.5211	0.0841	0.1388
DBX	0.0389	0.1306	0.2534	0.2091	0.1629	0.6112	0.0640	0.1537
UNDX	0.0599	0.1196	0.2424	0.2159	0.2500	0.5456	0.0839	0.1354
FR	0.0320	0.1041	0.2411	0.2123	0.1322	0.4792	0.0977	0.1833
SPX	0.0406	0.1048	0.2408	0.2150	0.1621	0.4970	0.0780	0.2107
PNX	0.0726	0.1179	0.2588	0.2086	0.3033	0.5627	0.0884	0.1944
Adaptive	0.0640	0.1119	0.2300	0.2095	0.2729	0.5197	0.1025	0.1934
FTSE MIB	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.0026	0.0522	0.2867	0.3011	0.0094	0.1727	0.0772	0.1560
UX	0.0128	0.0568	0.2889	0.3020	0.0440	0.1884	0.0930	0.2007
HX	0.0075	0.0479	0.2833	0.2960	0.0264	0.1588	0.0862	0.1560
LX	0.0041	0.0633	0.2911	0.2992	0.0139	0.2090	0.0988	0.1753
QBX	0.0372	0.0453	0.2773	0.2984	0.1335	0.1512	0.0758	0.1776
TPX	0.0140	0.0435	0.2874	0.3015	0.0490	0.1436	0.0632	0.1534

AX	0.0119	0.0552	0.2833	0.3021	0.0422	0.1841	0.0886	0.1670
GX	-0.0051	0.0524	0.2873	0.2987	-0.0184	0.1752	0.1119	0.1793
SBX	0.0301	0.0688	0.2853	0.3012	0.1077	0.2287	0.0960	0.2134
AVX	0.0159	0.0445	0.2934	0.3033	0.0568	0.1479	0.0792	0.1919
BLX	0.0183	0.0595	0.2875	0.3016	0.0666	0.1943	0.1018	0.1505
FX	0.0170	0.0406	0.2825	0.3052	0.0615	0.1344	0.0843	0.1778
GUX	0.0120	0.0454	0.2818	0.2983	0.0434	0.1524	0.0987	0.1713
TPX	0.0104	0.0627	0.2985	0.2969	0.0376	0.2077	0.0815	0.1752
LNX	0.0137	0.0538	0.2972	0.3017	0.0435	0.1801	0.0918	0.1645
DBX	0.0367	0.0365	0.3200	0.3038	0.1164	0.1217	0.0787	0.1847
UNDX	0.0028	0.0595	0.2907	0.2965	0.0100	0.1957	0.1150	0.1647
FR	0.0115	0.0672	0.2955	0.3014	0.0405	0.2248	0.0759	0.2205
SPX	0.0252	0.0566	0.2980	0.3015	0.0847	0.1882	0.0993	0.1573
PNX	0.0289	0.0605	0.2905	0.3047	0.0936	0.2025	0.0632	0.1839
Adaptive	0.0352	0.0558	0.2787	0.2935	0.1321	0.1931	0.0884	0.1598
CAC 40	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.1190	0.0963	0.2277	0.2327	0.5417	0.4165	0.0865	0.1786
UX	0.1093	0.1057	0.2257	0.2337	0.4960	0.4633	0.0956	0.1968
HX	0.1139	0.1002	0.2140	0.2309	0.5340	0.4340	0.1045	0.1786
LX	0.1197	0.1043	0.2195	0.2310	0.5372	0.4558	0.0825	0.1816
QBX	0.1182	0.1017	0.2149	0.2337	0.5576	0.4350	0.0744	0.1595
TPX	0.1008	0.1054	0.2234	0.2277	0.4477	0.4627	0.0651	0.1653
AX	0.1151	0.1055	0.2227	0.2328	0.5084	0.4487	0.1126	0.1911
GX	0.1078	0.1071	0.2196	0.2291	0.5073	0.4589	0.0857	0.1865
SBX	0.1083	0.1019	0.2207	0.2323	0.4790	0.4396	0.0959	0.1777
AVX	0.1042	0.1026	0.2268	0.2317	0.4738	0.4441	0.0814	0.1535
BLX	0.1125	0.1075	0.2188	0.2278	0.5223	0.4601	0.0933	0.1781
FX	0.1120	0.1078	0.2138	0.2318	0.5285	0.4674	0.0815	0.1828
GUX	0.1178	0.1016	0.2149	0.2312	0.5078	0.4344	0.1113	0.1676
TPX	0.1077	0.1020	0.2123	0.2323	0.5019	0.4453	0.0921	0.1794
LNX	0.1092	0.0931	0.2288	0.2318	0.4637	0.4028	0.0912	0.1657
DBX	0.1044	0.1080	0.2435	0.2300	0.4378	0.4639	0.0799	0.1835
UNDX	0.0939	0.1018	0.2321	0.2354	0.4233	0.4380	0.1149	0.1680
FR	0.1048	0.1040	0.2233	0.2286	0.4762	0.4479	0.0787	0.2129
SPX	0.0926	0.1119	0.2361	0.2290	0.4143	0.4863	0.0698	0.1709
PNX	0.1070	0.1041	0.2337	0.2335	0.4804	0.4523	0.0836	0.1818
Adaptive	0.1180	0.1010	0.2042	0.2259	0.5464	0.4453	0.0970	0.1574

Table 4.7: Out-of-sample portfolio metrics. Risk measure: $ES_{95\%}$. Tests with cardinality and upper/lower bound constraints, $\varepsilon = 1e-001$.

	$\hat{\mu^i}$		$\hat{\sigma^i}$		$\hat{SR^i}$		$Turnover$	
Nikkei 225	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.1727	0.1617	0.2175	0.1920	0.8468	0.8389	0.0621	0.1661
UX	0.1806	0.1605	0.2118	0.1926	0.8800	0.8398	0.1120	0.1830
HX	0.1991	0.1639	0.2159	0.1924	0.9749	0.8502	0.0776	0.1661
LX	0.1689	0.1408	0.2155	0.1947	0.8199	0.7285	0.0656	0.1815
QBX	0.1896	0.1517	0.2052	0.1942	0.9330	0.8078	0.1129	0.1726
TPX	0.1570	0.1587	0.2170	0.1923	0.7640	0.8399	0.0914	0.1939
AX	0.1726	0.1719	0.2149	0.1914	0.8236	0.9055	0.0904	0.1847
GX	0.1863	0.1472	0.2091	0.1912	0.9221	0.7680	0.0711	0.2102
SBX	0.1991	0.1665	0.2104	0.1910	0.9541	0.8676	0.0856	0.1731
AVX	0.1699	0.1676	0.2135	0.1902	0.8191	0.8709	0.0749	0.1663
BLX	0.1787	0.1637	0.2169	0.1908	0.8994	0.8418	0.0859	0.1475
FX	0.1701	0.1623	0.2121	0.1943	0.8418	0.8428	0.0978	0.1702
GUX	0.1918	0.1653	0.2157	0.1882	0.9469	0.8722	0.0955	0.1903
TPX	0.1727	0.1464	0.2129	0.1931	0.8294	0.7594	0.0888	0.1620
LNX	0.2002	0.1634	0.2174	0.1906	0.9400	0.8290	0.0838	0.1833
DBX	0.1402	0.1604	0.2280	0.1926	0.6434	0.8454	0.0770	0.1565
UNDX	0.1852	0.1489	0.2194	0.1916	0.8925	0.7827	0.1515	0.1744
FR	0.1865	0.1577	0.2119	0.1925	0.8978	0.8120	0.0762	0.3012
SPX	0.2070	0.1590	0.2086	0.1910	0.9555	0.8332	0.0794	0.1372
PNX	0.1919	0.1556	0.2119	0.1900	0.9000	0.8152	0.0627	0.1692
Adaptive	0.1965	0.1691	0.2068	0.1876	0.9803	0.8952	0.1278	0.1587
FTSE 100	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.0927	0.0986	0.1501	0.1656	0.6156	0.5845	0.0860	0.1690
UX	0.0889	0.1004	0.1548	0.1635	0.5870	0.6086	0.0970	0.1693
HX	0.0961	0.1060	0.1496	0.1615	0.6387	0.6437	0.0891	0.1690
LX	0.0876	0.1047	0.1552	0.1653	0.5792	0.6320	0.0894	0.1419
QBX	0.0984	0.0980	0.1442	0.1653	0.6599	0.5926	0.0790	0.1612
TPX	0.0841	0.1001	0.1560	0.1645	0.5470	0.6086	0.1174	0.1813
AX	0.0820	0.1005	0.1521	0.1663	0.5492	0.6055	0.0909	0.2084
GX	0.0940	0.0968	0.1498	0.1663	0.6168	0.5907	0.0780	0.1734
SBX	0.0891	0.1033	0.1504	0.1626	0.5753	0.6346	0.1045	0.1566

AVX	0.0904	0.0942	0.1529	0.1670	0.5832	0.5766	0.0763	0.2118
BLX	0.0942	0.1029	0.1507	0.1642	0.6233	0.6330	0.0990	0.1738
FX	0.0933	0.0977	0.1532	0.1670	0.6151	0.5995	0.0959	0.1824
GUX	0.0979	0.1001	0.1488	0.1649	0.6273	0.6014	0.0949	0.1725
TPX	0.0966	0.0950	0.1544	0.1655	0.6268	0.5721	0.0652	0.1700
LNX	0.0937	0.0958	0.1522	0.1655	0.6060	0.5789	0.0879	0.1725
DBX	0.0805	0.0999	0.1584	0.1667	0.5255	0.6012	0.0702	0.1510
UNDX	0.0855	0.1045	0.1488	0.1626	0.5577	0.6340	0.1659	0.1637
FR	0.0941	0.1002	0.1498	0.1641	0.6270	0.6120	0.0807	0.2440
SPX	0.0905	0.0987	0.1537	0.1641	0.5970	0.5925	0.0999	0.1857
PNX	0.0884	0.1044	0.1559	0.1631	0.5662	0.6357	0.0840	0.1608
Adaptive	0.1030	0.1055	0.1497	0.1648	0.6970	0.6393	0.1335	0.1779
Hang Seng	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.0328	0.1259	0.2471	0.2093	0.1319	0.6048	0.1034	0.1970
UX	0.0261	0.1295	0.2467	0.2157	0.1098	0.6084	0.0940	0.1801
HX	0.0497	0.1272	0.2434	0.2133	0.1989	0.5953	0.0871	0.1970
LX	0.0481	0.1139	0.2383	0.2135	0.1962	0.5332	0.1028	0.1674
QBX	0.0762	0.1153	0.2315	0.2166	0.3035	0.5509	0.0916	0.2001
TPX	0.0592	0.1115	0.2428	0.2163	0.2362	0.5329	0.1089	0.1982
AX	0.0398	0.1084	0.2343	0.2119	0.1594	0.5176	0.0878	0.1855
GX	0.0408	0.1162	0.2476	0.2083	0.1674	0.5549	0.0837	0.1845
SBX	0.0377	0.1246	0.2475	0.2127	0.1564	0.5783	0.1108	0.1578
AVX	0.0439	0.1210	0.2415	0.2119	0.1760	0.5733	0.0832	0.1616
BLX	0.0507	0.1182	0.2389	0.2109	0.2034	0.5562	0.1013	0.1591
FX	0.0554	0.1059	0.2419	0.2124	0.2224	0.5075	0.0777	0.1725
GUX	0.0511	0.1107	0.2395	0.2107	0.2076	0.5210	0.0755	0.1721
TPX	0.0622	0.1244	0.2439	0.2124	0.2482	0.5870	0.0708	0.1619
LNX	0.0565	0.1225	0.2416	0.2113	0.2315	0.5747	0.0821	0.1412
DBX	0.0369	0.1177	0.2563	0.2129	0.1545	0.5510	0.0773	0.1836
UNDX	0.0479	0.1261	0.2485	0.2144	0.1999	0.5752	0.1091	0.1674
FR	0.0638	0.1163	0.2422	0.2067	0.2638	0.5353	0.1092	0.1994
SPX	0.0360	0.1022	0.2419	0.2136	0.1437	0.4843	0.0942	0.2039
PNX	0.0585	0.1120	0.2452	0.2114	0.2444	0.5341	0.0760	0.1647
Adaptive	0.0509	0.1166	0.2420	0.2112	0.2168	0.5415	0.0950	0.1700
FTSE MIB	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.0290	0.0509	0.3000	0.3036	0.1048	0.1684	0.1074	0.1783
UX	0.0384	0.0321	0.2957	0.3067	0.1323	0.1064	0.1136	0.1837
HX	0.0317	0.0527	0.2952	0.3000	0.1111	0.1747	0.1042	0.1783
LX	0.0480	0.0326	0.3086	0.3042	0.1637	0.1076	0.0845	0.1655
QBX	0.0062	0.0584	0.2904	0.3014	0.0222	0.1946	0.0814	0.1782
TPX	0.0102	0.0504	0.2930	0.2973	0.0356	0.1662	0.0942	0.1737
AX	0.0152	0.0482	0.2974	0.2978	0.0542	0.1607	0.0837	0.1641
GX	0.0352	0.0604	0.2989	0.2983	0.1267	0.2019	0.0899	0.1906
SBX	0.0339	0.0413	0.2913	0.3028	0.1215	0.1373	0.1047	0.1755
AVX	0.0367	0.0515	0.3046	0.3016	0.1314	0.1708	0.0722	0.1860
BLX	0.0422	0.0533	0.2932	0.2994	0.1537	0.1739	0.0746	0.1436
FX	0.0166	0.0435	0.2958	0.3009	0.0603	0.1442	0.0785	0.1816
GUX	0.0224	0.0586	0.3051	0.2991	0.0813	0.1968	0.0939	0.1809
TPX	0.0278	0.0525	0.3054	0.2994	0.1005	0.1739	0.0808	0.1841
LNX	0.0313	0.0633	0.3041	0.3016	0.0991	0.2121	0.1021	0.1683
DBX	0.0092	0.0487	0.3262	0.3028	0.0291	0.1621	0.0977	0.1988
UNDX	-0.0126	0.0649	0.3039	0.3006	-0.0447	0.2134	0.1023	0.1649
FR	0.0391	0.0586	0.3058	0.3008	0.1373	0.1961	0.1002	0.2012
SPX	0.0267	0.0491	0.2993	0.2998	0.0899	0.1634	0.0790	0.1589
PNX	0.0014	0.0513	0.3024	0.2997	0.0045	0.1716	0.0822	0.1571
Adaptive	0.0287	0.0506	0.2797	0.3004	0.1077	0.1752	0.1057	0.1742
CAC 40	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.1206	0.1032	0.2262	0.2309	0.5492	0.4463	0.0857	0.1808
UX	0.1134	0.1001	0.2416	0.2355	0.5143	0.4386	0.0853	0.1418
HX	0.1065	0.1063	0.2391	0.2297	0.4997	0.4603	0.1005	0.1808
LX	0.1140	0.1109	0.2367	0.2309	0.5116	0.4848	0.0710	0.1993
QBX	0.1112	0.1062	0.2191	0.2296	0.5245	0.4544	0.0714	0.1740
TPX	0.1263	0.1034	0.2289	0.2353	0.5609	0.4535	0.1044	0.1567
AX	0.1158	0.0992	0.2348	0.2291	0.5112	0.4222	0.0862	0.1892
GX	0.1093	0.1030	0.2349	0.2341	0.5148	0.4413	0.0803	0.1604
SBX	0.1097	0.1106	0.2233	0.2285	0.4852	0.4769	0.0928	0.1591
AVX	0.1094	0.1056	0.2276	0.2287	0.4971	0.4574	0.0874	0.1953
BLX	0.1055	0.1012	0.2334	0.2318	0.4897	0.4332	0.0887	0.1771
FX	0.1100	0.1086	0.2325	0.2291	0.5191	0.4706	0.0943	0.1732
GUX	0.0939	0.0996	0.2377	0.2341	0.4047	0.4258	0.0873	0.1977
TPX	0.1142	0.1097	0.2350	0.2325	0.5322	0.4790	0.0818	0.1638
LNX	0.1140	0.1064	0.2205	0.2343	0.4844	0.4603	0.0797	0.1674
DBX	0.1080	0.1052	0.2463	0.2319	0.4531	0.4519	0.0721	0.1973
UNDX	0.1093	0.1056	0.2201	0.2284	0.4925	0.4546	0.1126	0.1643
FR	0.1051	0.1025	0.2362	0.2311	0.4775	0.4414	0.0692	0.2294
SPX	0.1215	0.1024	0.2306	0.2313	0.5432	0.4452	0.0856	0.1571
PNX	0.1138	0.1109	0.2341	0.2308	0.5110	0.4821	0.0938	0.1983

Adaptive	0.1072	0.1137	0.2277	0.2279	0.4963	0.5013	0.1057	0.1946

Table 4.8: Out-of-sample portfolio metrics. Risk measure: *Mean-Variance*. Tests with cardinality and upper/lower bound constraints, $\varepsilon = 1e-001$.

	$\hat{\mu^i}$		$\hat{\sigma^i}$		$\hat{SR^i}$		$Turnover$	
Nikkei 225	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.2312	0.1570	0.1990	0.1925	1.1336	0.8146	0.0922	0.1759
UX	0.2375	0.1560	0.1979	0.1928	1.1570	0.8166	0.0758	0.1603
HX	0.2354	0.1726	0.1993	0.1898	1.1522	0.8954	0.0792	0.1759
LX	0.2331	0.1627	0.1974	0.1922	1.1315	0.8421	0.0665	0.1984
QBX	0.2335	0.1608	0.2003	0.1937	1.1490	0.8562	0.0941	0.1444
TPX	0.2253	0.1507	0.2025	0.1929	1.0964	0.7971	0.0838	0.1721
AX	0.2332	0.1679	0.1982	0.1906	1.1125	0.8842	0.0749	0.1688
GX	0.2204	0.1638	0.1988	0.1906	1.0910	0.8545	0.0840	0.1672
SBX	0.2314	0.1619	0.1982	0.1913	1.1093	0.8437	0.0811	0.1816
AVX	0.2279	0.1602	0.1971	0.1895	1.0986	0.8324	0.0843	0.1527
BLX	0.2540	0.1660	0.1954	0.1918	1.2785	0.8538	0.0787	0.1806
FX	0.2311	0.1507	0.1982	0.1916	1.1436	0.7827	0.0926	0.1856
GUX	0.2381	0.1795	0.1993	0.1882	1.1757	0.9470	0.0885	0.1771
TPX	0.2320	0.1621	0.1970	0.1914	1.1140	0.8408	0.0826	0.1467
LNX	0.1831	0.1622	0.2112	0.1914	0.8596	0.8231	0.1123	0.1632
DBX	0.1730	0.1561	0.2096	0.1911	0.7937	0.8228	0.0975	0.2000
UNDX	0.2025	0.1507	0.2043	0.1934	0.9756	0.7920	0.1152	0.1915
FR	0.1996	0.1632	0.2074	0.1915	0.9609	0.8402	0.0695	0.2243
SPX	0.1731	0.1677	0.2103	0.1932	0.7990	0.8786	0.0870	0.1628
PNX	0.1934	0.1510	0.2113	0.1892	0.9070	0.7911	0.0819	0.1660
Adaptive	0.2330	0.1635	0.1925	0.1832	1.1621	0.8658	0.0841	0.1890
FTSE 100	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.2312	0.1570	0.1990	0.1925	1.1336	0.8146	0.0922	0.1759
UX	0.2375	0.1560	0.1979	0.1928	1.1570	0.8166	0.0758	0.1603
HX	0.2354	0.1726	0.1993	0.1898	1.1522	0.8954	0.0792	0.1759
LX	0.2331	0.1627	0.1974	0.1922	1.1315	0.8421	0.0665	0.1984
QBX	0.2335	0.1608	0.2003	0.1937	1.1490	0.8562	0.0941	0.1444
TPX	0.2253	0.1507	0.2025	0.1929	1.0964	0.7971	0.0838	0.1721
AX	0.2332	0.1679	0.1982	0.1906	1.1125	0.8842	0.0749	0.1688
GX	0.2204	0.1638	0.1988	0.1906	1.0910	0.8545	0.0840	0.1672
SBX	0.2314	0.1619	0.1982	0.1913	1.1093	0.8437	0.0811	0.1816
AVX	0.2279	0.1602	0.1971	0.1895	1.0986	0.8324	0.0843	0.1527
BLX	0.2540	0.1660	0.1954	0.1918	1.2785	0.8538	0.0787	0.1806
FX	0.2311	0.1507	0.1982	0.1916	1.1436	0.7827	0.0926	0.1856
GUX	0.2381	0.1795	0.1993	0.1882	1.1757	0.9470	0.0885	0.1771
TPX	0.2320	0.1621	0.1970	0.1914	1.1140	0.8408	0.0826	0.1467
LNX	0.1831	0.1622	0.2112	0.1914	0.8596	0.8231	0.1123	0.1632
DBX	0.1730	0.1561	0.2096	0.1911	0.7937	0.8228	0.0975	0.2000
UNDX	0.2025	0.1507	0.2043	0.1934	0.9756	0.7920	0.1152	0.1915
FR	0.1996	0.1632	0.2074	0.1915	0.9609	0.8402	0.0695	0.2243
SPX	0.1731	0.1677	0.2103	0.1932	0.7990	0.8786	0.0870	0.1628
PNX	0.1934	0.1510	0.2113	0.1892	0.9070	0.7911	0.0819	0.1660
Adaptive	0.2330	0.1635	0.1925	0.1832	1.1621	0.8658	0.0841	0.1890
Hang Seng	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.0613	0.1126	0.2314	0.2140	0.2464	0.5405	0.0784	0.1544
UX	0.0467	0.1115	0.2275	0.2147	0.1966	0.5239	0.0956	0.1868
HX	0.0403	0.1147	0.2276	0.2108	0.1612	0.5365	0.0687	0.1544
LX	0.0451	0.0997	0.2278	0.2162	0.1841	0.4667	0.1064	0.1491
QBX	0.0393	0.1196	0.2302	0.2151	0.1566	0.5717	0.0730	0.2017
TPX	0.0526	0.1113	0.2278	0.2138	0.2101	0.5317	0.1055	0.1806
AX	0.0516	0.1315	0.2292	0.2105	0.2066	0.6281	0.0654	0.1695
GX	0.0429	0.1153	0.2305	0.2113	0.1760	0.5505	0.0875	0.1780
SBX	0.0359	0.1211	0.2350	0.2140	0.1490	0.5619	0.0984	0.1702
AVX	0.0375	0.1113	0.2279	0.2150	0.1503	0.5272	0.1139	0.1911
BLX	0.0324	0.1131	0.2258	0.2127	0.1303	0.5324	0.1032	0.2037
FX	0.0444	0.1121	0.2271	0.2088	0.1784	0.5371	0.0710	0.2070
GUX	0.0426	0.1217	0.2295	0.2118	0.1733	0.5728	0.1280	0.1592
TPX	0.0236	0.1215	0.2313	0.2138	0.0943	0.5734	0.0936	0.2463
LNX	0.0174	0.1246	0.2380	0.2141	0.0713	0.5847	0.0752	0.1844
DBX	0.0458	0.1168	0.2426	0.2136	0.1919	0.5467	0.0980	0.1693
UNDX	0.0312	0.1270	0.2354	0.2107	0.1301	0.5791	0.1197	0.1852
FR	0.0439	0.1273	0.2289	0.2112	0.1815	0.5858	0.0834	0.2375
SPX	0.0605	0.1156	0.2456	0.2085	0.2414	0.5482	0.1074	0.1843
PNX	0.0929	0.1223	0.2565	0.2123	0.3881	0.5833	0.0957	0.1825
Adaptive	0.0715	0.1058	0.2218	0.1989	0.3045	0.4912	0.1145	0.1961
FTSE MIB	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$

OPX	0.0147	0.0532	0.2706	0.3007	0.0530	0.1760	0.0881	0.1892
UX	0.0070	0.0494	0.2697	0.2982	0.0240	0.1640	0.1010	0.1898
HX	0.0075	0.0529	0.2714	0.3013	0.0261	0.1751	0.0985	0.1892
LX	0.0356	0.0382	0.2812	0.3027	0.1214	0.1262	0.0907	0.1837
QBX	0.0125	0.0593	0.2666	0.3027	0.0449	0.1978	0.0858	0.1902
TPX	0.0047	0.0492	0.2726	0.3037	0.0163	0.1624	0.0597	0.1588
AX	0.0170	0.0686	0.2664	0.2989	0.0604	0.2288	0.0823	0.1522
GX	0.0239	0.0484	0.2732	0.3042	0.0860	0.1619	0.0905	0.1697
SBX	-0.0013	0.0505	0.2759	0.3021	-0.0047	0.1678	0.0873	0.1658
AVX	0.0114	0.0423	0.2719	0.3006	0.0407	0.1404	0.0899	0.1533
BLX	0.0234	0.0543	0.2683	0.2987	0.0852	0.1770	0.0863	0.1985
FX	0.0086	0.0595	0.2715	0.2988	0.0313	0.1971	0.0668	0.2043
GUX	0.0148	0.0495	0.2671	0.3002	0.0536	0.1661	0.0630	0.1694
TPX	0.0466	0.0407	0.2808	0.2993	0.1682	0.1348	0.0816	0.1619
LNX	0.0031	0.0377	0.2975	0.3022	0.0100	0.1264	0.0894	0.1591
DBX	0.0384	0.0434	0.2918	0.3008	0.1217	0.1446	0.0910	0.1912
UNDX	0.0068	0.0470	0.2784	0.3044	0.0242	0.1545	0.0941	0.1641
FR	0.0101	0.0600	0.2774	0.3045	0.0354	0.2005	0.0738	0.1990
SPX	0.0007	0.0543	0.2971	0.3011	0.0024	0.1805	0.0913	0.1879
PNX	0.0236	0.0381	0.2834	0.3001	0.0765	0.1275	0.0988	0.1837
Adaptive	0.0155	0.0860	0.2651	0.2785	0.0581	0.2977	0.1113	0.1670
CAC 40	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.1081	0.0947	0.2120	0.2326	0.4921	0.4095	0.0964	0.1913
UX	0.1030	0.1060	0.2137	0.2292	0.4674	0.4648	0.0871	0.1966
HX	0.1189	0.1105	0.2062	0.2270	0.5577	0.4784	0.0925	0.1913
LX	0.1063	0.1032	0.2138	0.2321	0.4770	0.4510	0.0722	0.1788
QBX	0.1143	0.1077	0.2066	0.2315	0.5394	0.4607	0.1218	0.1624
TPX	0.1140	0.1108	0.2108	0.2307	0.5062	0.4860	0.0563	0.2012
AX	0.1141	0.1019	0.2064	0.2305	0.5037	0.4334	0.0819	0.1530
GX	0.1151	0.1031	0.2074	0.2286	0.5421	0.4415	0.0907	0.1572
SBX	0.1193	0.0875	0.2039	0.2309	0.5278	0.3774	0.0891	0.1677
AVX	0.1134	0.0985	0.2078	0.2326	0.5151	0.4266	0.1015	0.1965
BLX	0.1270	0.1010	0.2032	0.2325	0.5895	0.4322	0.0761	0.1754
FX	0.1182	0.0915	0.2040	0.2346	0.5574	0.3963	0.0762	0.1652
GUX	0.1162	0.1096	0.2126	0.2328	0.5008	0.4685	0.0948	0.1690
TPX	0.1178	0.1055	0.2073	0.2331	0.5487	0.4603	0.0641	0.1762
LNX	0.1095	0.1055	0.2313	0.2290	0.4651	0.4568	0.0780	0.1659
DBX	0.1133	0.1075	0.2267	0.2297	0.4753	0.4619	0.0535	0.1579
UNDX	0.1212	0.1028	0.2087	0.2316	0.5462	0.4423	0.0857	0.1474
FR	0.1186	0.0983	0.2093	0.2284	0.5389	0.4234	0.0879	0.1946
SPX	0.1205	0.1178	0.2298	0.2251	0.5389	0.5118	0.0929	0.1572
PNX	0.1017	0.1066	0.2258	0.2304	0.4566	0.4636	0.1055	0.1923
Adaptive	0.1316	0.1172	0.1982	0.2208	0.6097	0.5164	0.0695	0.1881

Table 4.9: Out-of-sample portfolio metrics. Risk measure: *Two-sided*. Tests with cardinality and upper/lower bound constraints, $\varepsilon = 1e-001$.

	$\hat{\mu^i}$		$\hat{\sigma^i}$		$S\hat{R}^i$		$Turnover$	
Nikkei 225	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.2241	0.1506	0.1989	0.1933	1.0987	0.7816	0.0880	0.1841
UX	0.2264	0.1589	0.1986	0.1924	1.1028	0.8314	0.0798	0.1859
HX	0.2179	0.1694	0.1994	0.1935	1.0665	0.8788	0.0796	0.1841
LX	0.2193	0.1664	0.1999	0.1894	1.0643	0.8610	0.0857	0.1668
QBX	0.2259	0.1440	0.2006	0.1939	1.1118	0.7667	0.0856	0.1639
TPX	0.2040	0.1573	0.2018	0.1943	0.9927	0.8321	0.1018	0.1863
AX	0.2198	0.1587	0.1994	0.1929	1.0485	0.8359	0.0982	0.1682
GX	0.2166	0.1638	0.2001	0.1920	1.0721	0.8546	0.0703	0.1900
SBX	0.2259	0.1451	0.2018	0.1947	1.0828	0.7565	0.1104	0.1710
AVX	0.2185	0.1613	0.2015	0.1888	1.0530	0.8385	0.1120	0.2052
BLX	0.2254	0.1592	0.2026	0.1918	1.1343	0.8189	0.0971	0.2176
FX	0.2184	0.1498	0.1995	0.1955	1.0808	0.7780	0.0943	0.1846
GUX	0.2078	0.1562	0.2017	0.1928	1.0261	0.8244	0.0924	0.1701
TPX	0.2215	0.1581	0.1996	0.1888	1.0638	0.8200	0.0615	0.1528
LNX	0.1853	0.1748	0.2112	0.1895	0.8699	0.8871	0.0861	0.1285
DBX	0.2107	0.1554	0.2025	0.1918	0.9668	0.8187	0.0817	0.1628
UNDX	0.2205	0.1570	0.1993	0.1960	1.0624	0.8256	0.1256	0.2152
FR	0.2241	0.1672	0.1986	0.1916	1.0789	0.8610	0.0949	0.2448
SPX	0.1686	0.1619	0.2098	0.1886	0.7781	0.8487	0.0720	0.1708
PNX	0.1893	0.1597	0.2107	0.1917	0.8876	0.8368	0.0790	0.1568
Adaptive	0.2333	0.1848	0.1931	0.1823	1.1635	0.9782	0.0583	0.1728
FTSE 100	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.1099	0.1037	0.1445	0.1630	0.7296	0.6145	0.0791	0.1660
UX	0.1135	0.1007	0.1442	0.1642	0.7490	0.6103	0.0847	0.1921
HX	0.1111	0.0990	0.1444	0.1673	0.7386	0.6010	0.0656	0.1660

LX	0.1126	0.1020	0.1440	0.1649	0.7441	0.6155	0.0732	0.1661
QBX	0.1104	0.1031	0.1443	0.1639	0.7407	0.6236	0.0812	0.1605
TPX	0.1099	0.0930	0.1447	0.1651	0.7144	0.5657	0.0783	0.1654
AX	0.1113	0.0955	0.1444	0.1658	0.7455	0.5757	0.0818	0.1713
GX	0.1106	0.0953	0.1440	0.1631	0.7256	0.5814	0.0952	0.1493
SBX	0.1100	0.1079	0.1446	0.1629	0.7105	0.6633	0.1017	0.1834
AVX	0.1113	0.1051	0.1442	0.1624	0.7179	0.6431	0.0883	0.1798
BLX	0.1106	0.0943	0.1442	0.1655	0.7315	0.5803	0.0797	0.1905
FX	0.1118	0.1033	0.1441	0.1654	0.7374	0.6338	0.0852	0.1589
GUX	0.1143	0.0986	0.1437	0.1645	0.7326	0.5921	0.0531	0.1794
TPX	0.1113	0.0974	0.1448	0.1654	0.7221	0.5862	0.0781	0.1669
LNX	0.0893	0.1007	0.1548	0.1665	0.5776	0.6086	0.0911	0.1637
DBX	0.1091	0.0998	0.1456	0.1631	0.7119	0.6009	0.0977	0.1571
UNDX	0.1102	0.0988	0.1443	0.1649	0.7184	0.5993	0.1036	0.1841
FR	0.1101	0.1000	0.1450	0.1634	0.7334	0.6106	0.0856	0.1918
SPX	0.0994	0.0953	0.1553	0.1638	0.6558	0.5720	0.0700	0.1732
PNX	0.1066	0.1018	0.1492	0.1645	0.6832	0.6197	0.0818	0.1616
Adaptive	0.1257	0.1121	0.1393	0.1563	0.8506	0.6789	0.1386	0.1720
Hang Seng	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.0456	0.1199	0.2305	0.2121	0.1835	0.5758	0.0570	0.1371
UX	0.0419	0.1151	0.2320	0.2166	0.1761	0.5411	0.0911	0.1986
HX	0.0426	0.1223	0.2330	0.2109	0.1708	0.5722	0.0897	0.1371
LX	0.0446	0.1223	0.2303	0.2100	0.1820	0.5726	0.0830	0.1646
QBX	0.0455	0.1219	0.2312	0.2103	0.1812	0.5826	0.0550	0.1720
TPX	0.0466	0.1139	0.2318	0.2165	0.1860	0.5440	0.1437	0.1657
AX	0.0427	0.1103	0.2319	0.2145	0.1712	0.5269	0.1172	0.2158
GX	0.0373	0.1063	0.2300	0.2132	0.1529	0.5076	0.1205	0.2295
SBX	0.0451	0.1037	0.2305	0.2153	0.1871	0.4815	0.0786	0.1932
AVX	0.0454	0.1071	0.2325	0.2130	0.1817	0.5074	0.0789	0.1509
BLX	0.0439	0.1082	0.2308	0.2117	0.1762	0.5092	0.0617	0.1533
FX	0.0453	0.1246	0.2316	0.2126	0.1818	0.5969	0.0957	0.1363
GUX	0.0320	0.1124	0.2374	0.2131	0.1299	0.5291	0.0766	0.1725
TPX	0.0504	0.1068	0.2319	0.2133	0.2012	0.5040	0.1017	0.1453
LNX	0.0526	0.1117	0.2432	0.2111	0.2155	0.5239	0.0903	0.1801
DBX	0.0499	0.1283	0.2333	0.2104	0.2092	0.6006	0.0645	0.1766
UNDX	0.0449	0.1468	0.2313	0.2156	0.1874	0.6693	0.0971	0.1556
FR	0.0440	0.1196	0.2310	0.2136	0.1818	0.5504	0.0859	0.1979
SPX	0.0607	0.1200	0.2489	0.2129	0.2425	0.5690	0.1025	0.1821
PNX	0.0389	0.1097	0.2337	0.2108	0.1623	0.5234	0.0643	0.2039
Adaptive	0.0612	0.1096	0.2230	0.2041	0.2609	0.5089	0.0434	0.1786
FTSE MIB	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.0122	0.0488	0.2740	0.3037	0.0440	0.1616	0.0662	0.1435
UX	0.0142	0.0482	0.2731	0.3051	0.0488	0.1599	0.0888	0.1740
HX	0.0120	0.0575	0.2751	0.3024	0.0420	0.1906	0.0746	0.1435
LX	0.0157	0.0522	0.2740	0.3063	0.0534	0.1722	0.0782	0.1543
QBX	0.0137	0.0529	0.2742	0.2951	0.0493	0.1764	0.0984	0.1938
TPX	0.0113	0.0703	0.2747	0.3009	0.0396	0.2318	0.0807	0.1782
AX	0.0111	0.0526	0.2748	0.3017	0.0396	0.1755	0.0818	0.1719
GX	0.0132	0.0429	0.2758	0.2966	0.0476	0.1434	0.0615	0.1626
SBX	0.0074	0.0738	0.2744	0.3098	0.0266	0.2453	0.0811	0.1395
AVX	0.0153	0.0636	0.2754	0.3043	0.0546	0.2112	0.1056	0.1470
BLX	0.0137	0.0529	0.2749	0.3003	0.0499	0.1727	0.0990	0.1944
FX	0.0115	0.0467	0.2751	0.2981	0.0416	0.1547	0.1132	0.1642
GUX	0.0017	0.0654	0.2771	0.2980	0.0061	0.2197	0.0881	0.2134
TPX	0.0131	0.0495	0.2737	0.3010	0.0475	0.1639	0.0907	0.1612
LNX	-0.0063	0.0459	0.3056	0.3000	-0.0200	0.1538	0.0742	0.1572
DBX	0.0126	0.0766	0.2807	0.2984	0.0398	0.2551	0.0941	0.1485
UNDX	0.0139	0.0634	0.2745	0.3023	0.0493	0.2086	0.1007	0.1924
FR	0.0193	0.0610	0.2755	0.3036	0.0678	0.2040	0.0911	0.2436
SPX	0.0450	0.0377	0.3020	0.3024	0.1514	0.1253	0.1010	0.1722
PNX	0.0180	0.0516	0.2871	0.2986	0.0584	0.1726	0.1004	0.1788
Adaptive	0.0310	0.0503	0.2626	0.2844	0.1166	0.1742	0.0757	0.2206
CAC 40	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$	$K = 10$	$K = 20$
OPX	0.1131	0.1013	0.2117	0.2305	0.5149	0.4380	0.0734	0.1724
UX	0.1156	0.1112	0.2102	0.2297	0.5246	0.4873	0.0688	0.1516
HX	0.1136	0.1150	0.2098	0.2323	0.5329	0.4979	0.0872	0.1724
LX	0.1164	0.1066	0.2094	0.2318	0.5224	0.4661	0.1083	0.1628
QBX	0.1142	0.1007	0.2103	0.2345	0.5389	0.4310	0.0964	0.1777
TPX	0.1141	0.1010	0.2096	0.2332	0.5067	0.4434	0.0936	0.1887
AX	0.1147	0.1081	0.2103	0.2300	0.5067	0.4600	0.0991	0.1513
GX	0.1170	0.1020	0.2106	0.2288	0.5510	0.4370	0.0874	0.1702
SBX	0.1163	0.1097	0.2102	0.2332	0.5145	0.4731	0.0665	0.1717
AVX	0.1162	0.0990	0.2108	0.2306	0.5280	0.4286	0.0750	0.1661
BLX	0.1151	0.1094	0.2096	0.2324	0.5343	0.4680	0.0952	0.1685
FX	0.1140	0.1096	0.2101	0.2311	0.5379	0.4748	0.0613	0.1612
GUX	0.1155	0.1062	0.2098	0.2295	0.4976	0.4538	0.0786	0.1620
TPX	0.1152	0.1141	0.2099	0.2315	0.5368	0.4978	0.0704	0.1698
LNX	0.1188	0.1090	0.2306	0.2312	0.5049	0.4718	0.0937	0.1621

DBX	0.1112	0.1008	0.2121	0.2290	0.4664	0.4331	0.1113	0.1807
UNDX	0.1153	0.0991	0.2094	0.2287	0.5196	0.4262	0.0804	0.1881
FR	0.1165	0.1023	0.2093	0.2294	0.5291	0.4404	0.0904	0.1805
SPX	0.1131	0.1083	0.2350	0.2303	0.5057	0.4709	0.1086	0.1932
PNX	0.1130	0.1055	0.2268	0.2283	0.5072	0.4585	0.0745	0.2080
Adaptive	0.1294	0.1078	0.2023	0.2207	0.5993	0.4751	0.0943	0.1651

Table 4.10: Out-of-sample portfolio metrics. Risk measure: *Risk parity*. Tests with cardinality and upper/lower bound constraints, $\varepsilon = 1e - 001$.

Conclusions

Before discussing the main conclusions of our work, we would like to point out some useful insights into the analysis and the experiments we have proposed before:

- Although we have used a combination of exploration and exploitation operators, actually most real-valued crossover operators behave similarly, with a strong focus on quality improvement, whereas only a few are actually able to enforce a tradeoff between quality and diversity. This is particularly relevant for the algorithm performance, as a good exploration of the search space is necessary to prevent the algorithm from getting stuck in local optima;
- The incorporation of a set of crossover into the algorithm has not been as straightforward as we expected: in order to implement effectively the adaptive operator selection, we first had to select them thoroughly, in order to remove the nonconverging ones;
- Despite the fact that the high-level search strategies have robustly influenced the selection probability and ultimately the EvE balance, we have found modest evidence of a strategy outperforming the others. A larger and more diversified set of operators may lead to different results.

Now, let us recap the key findings of our work. First, we have taken into account the general framework outlined by Maturana et al. (2010) in order to detect the potential benefits of a parameter control strategy for genetic algorithms; then, we have considered an extension of this approach proposed by di Tollo et al. (2015). We have implemented a controller -which allows to perform adaptive operator selection- and then we have included a set of high level search strategies, in order to achieve a dynamic EvE balance. This approach has several benefit:

- At each iteration, the controller chooses an optimal operator, according to several performance criteria and to a given search strategy;
- At each iteration, the EvE balance is managed in a dynamic fashion, in order to address robustly several optimization problems for varying problem instances.

We have proposed an extension of standard metaheuristic solvers, in the context of portfolio selection problems with mixed-integer constraints, where a real-valued population of portfolios is managed by an EA connected to a controller with an I/O interface, by which the EA transmits the identifier of the last applied operator and its performance; then, the controller yields the identifier of the operator to be applied at the next iteration (di Tollo et al. (2015)).

In order to assess the benefits of this approach, we have considered a range of constrained portfolio selection problems; then, each constrained model has been reformulated as an unconstrained one by means of an ℓ_1 exact penalty method, a widespread approach for handling nonlinear programs.

The computational analysis on a set of real-world test problems shows that generic EAs do not perform homogeneously across problem instances, whereas using an optimal operator at different stages of the search process leads to slightly improved solutions. Furthermore, we note that the adaptive policy behaves exactly as expected,

turning to exploration when improvements become harder, in order to escape from local optima, and opting for exploitation when the average quality of the population is poor. Finally, the probability of selection of each operator, the entropy and the fitness levels display high sensitivity to the search policy, which is a desirable property. An out-of-sample test with periodically-rebalanced portfolios has then confirmed the effectiveness of the adaptive strategy, which tends to outperform most standard EAs.

Appendix A

KKT Conditions

Consider again the general minimization problem 2.25 with equality and inequality constraints:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \phi(\mathbf{x}) \\ \text{s.t.} \quad & g_i(\mathbf{x}) = 0 \quad i = 1, \dots, m \\ & k_i(\mathbf{x}) \leq 0 \quad i = m+1, \dots, p \end{aligned} \tag{A.1}$$

Let x^* be a local minimum and $\mathcal{I} = \{i : k_i(\mathbf{x}) = 0\}$, i.e. the set of binding constraints for which $k_i(\mathbf{x}) \leq 0$ is satisfied with equality. Furthermore, we say that $\mathbf{x}^*$ is a regular point of the constraints of problem A.1 if the gradient vectors $\nabla k_i(\mathbf{x}^*)$ for $i \in \mathcal{I}$ and $\nabla g_i(\mathbf{x}^*)$ for $g_i = 1, \dots, m$ are linearly independent (equivalently, we also say that the Linear Independent Constraint Qualification is satisfied). The first order necessary conditions are the following:

Theorem A.1 (*KKT First Order Necessary Conditions*)**.** *Given a point* $\mathbf{x}^*$ *denoting a local minimizer of problem A.1, assume that* $\mathbf{x}^*$ *is a regular point. Then there is a unique Lagrange multiplier vector* (u_i^*, λ_i^*) *such that:*

$$\begin{aligned} & \nabla\phi(\mathbf{x}^*) + \sum_{i=1}^{m} u_i \nabla g_i(\mathbf{x}^*) + \sum_{i \in \mathcal{I}} \lambda_i \nabla k_i(\mathbf{x}^*) = 0 \\ & g_i(\mathbf{x}^*) = 0 \quad \textit{for } i = 1, \dots, m \\ & \lambda_i^* k_i(\mathbf{x}^*) = 0 \quad \textit{for } i \in \mathcal{I} \\ & k_i(\mathbf{x}^*) \leq 0 \quad \textit{for } i \in \mathcal{I} \\ & \lambda_i \geq 0 \quad \textit{for } i \in \mathcal{I} \end{aligned} \tag{A.2}$$

The second order necessary and sufficient conditions could be stated as follows:

Theorem A.2 (*KKT Second Order Necessary Conditions*)**.** *Given a point* $\mathbf{x}^*$ *denoting a local minimizer of problem A.1, assume that* $\mathbf{x}^*$ *is a regular point. Furthermore, suppose that* f *is twice continuously differentiable; then there exist* u_i *for* $i = 1, \dots, m$ *and* λ_i *for* $i \in \mathcal{I}$*, such that:*

$$\nabla^2\phi(\mathbf{x}^*) + \sum_{i=1}^{m} u_i \nabla^2 g_i(\mathbf{x}^*) + \sum_{i \in \mathcal{I}} \lambda_i \nabla^2 k_i(\mathbf{x}^*) \tag{A.3}$$

is positive semidefinite on the tangent subspace $T(\mathbf{x}^*)$ *of active constraints, where* $T(\mathbf{x}^*) = \{\mathbf{d} \in \mathbb{R}^n : \nabla g_i(\mathbf{x}^*)^T \mathbf{d} = 0, i = 1, \dots, m \text{ and } \nabla k_i(\mathbf{x}^*)^T \mathbf{d} = 0, i \in \mathcal{I}\}$

Theorem A.3 (*KKT Second Order Sufficient Conditions*)**.** *Given a point* $\mathbf{x}^*$ *denoting a local minimizer of problem A.1, assume that* $\mathbf{x}^*$ *is a regular point. Furthermore, suppose that* f *is twice continuously differentiable; then there exist* u_i *for* $i = 1, \dots, m$

and λ_i *for* $i \in \mathcal{I}$*, such that:*

$$\nabla^2\phi(\mathbf{x}^*) + \sum_{i=1}^{m} u_i \nabla^2 g_i(\mathbf{x}^*) + \sum_{i\in\mathcal{I}} \lambda_i \nabla^2 k_i(\mathbf{x}^*) \tag{A.4}$$

is positive definite on the tangent subspace $T(\mathbf{x}^*)$ *of active constraints.*

Appendix B

Source code

```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
import scipy.interpolate as sci
import random
import seaborn as sns
import scipy.optimize as sco
import time
import scipy.stats as scis

##Cleaning data and computing historical returns/covariance matrix

data=pd.read_csv('C:\\Users\\nonloso\\OneDrive\\Desktop\\codici_ga\\FTSE_MIB1.CSV
    ',header=None)
data.sort_index(inplace=True)
n_stocks=len(data.columns)
returns_set_1=data.pct_change().dropna()
average_returns_set_1=returns_set_1.mean()
covariance_matrix_1=returns_set_1.cov()

##Risk measures and standard portfolio statistics

def returns(pop):
    portfolio_returns=np.dot(average_returns_set_1,pop.T)
    weights_init=pop
    portfolio_returns=np.zeros(len(weights_init))
    for i in range(len(weights_init)):
        portfolio_returns[i] = np.sum(average_returns_set_1 * weights_init[i])
    return portfolio_returns


def vol(pop):
    covariance_matrix_1=returns_set_1.cov()
    weights_init=pop
    std_dev_portfolio=np.zeros(len(weights_init))
    for i in range(len(pop)):
        std_dev_portfolio[i]=np.sqrt(np.dot(weights_init[i].T,np.dot(
            covariance_matrix_1,weights_init[i])))
    return std_dev_portfolio

def risk_parity(pop):
    weights_init=pop
    fRP=np.zeros(np.shape(weights_init))
    portvar=vol(weights_init)**2
    Cx=(np.dot(covariance_matrix_1,weights_init.T))
    for j in range(len(weights_init.T)):
            #fRP[:,j]=(((weights_init[:,j]*Cx[j,:])/portvar)-(1/genes))**2  #
                second option with squared cost
            fRP[:,j]=np.abs(((weights_init[:,j]*Cx[j,:])/portvar)-(1/genes)) #
                first option with absolute cost
    rp=-np.sum(fRP,axis=1)
    return rp

def mean_absolute_deviation(pop):
    weights_init=pop
    portfolio_returns_mm=returns_set_1@weights_init.T
    mad=np.mean(np.abs(portfolio_returns_mm-np.mean(portfolio_returns_mm)))
    return mad

def omega_ratio(pop):
```

```
    weights_init=pop
    portfolio_returns_mm=returns_set_1@weights_init.T
    omega=(np.sum(np.minimum(portfolio_returns_mm,0),axis=0)/np.sum(np.maximum(
        portfolio_returns_mm,0),axis=0))
    return omega

def twosided(pop):
    a=0.25
    weights_init=pop
    portfolio_returns_mm=returns_set_1@weights_init.T
    twoside=np.zeros(len(portfolio_returns_mm.T))
    upside=np.maximum(portfolio_returns_mm-portfolio_returns_mm.mean(),0)
    downside=np.maximum(portfolio_returns_mm.mean()-portfolio_returns_mm,0)
    for z in range(len(portfolio_returns_mm.T)):
       twoside[z]=-a*np.linalg.norm(upside.iloc[:,z],ord=1)-(1-a)*np.linalg.norm(
           downside.iloc[:,z],ord=2)+portfolio_returns_mm.mean()[z]
    #twosided=a*np.linalg.norm(upside,ord=1)+(1-a)*np.linalg(downside,ord=2)-
        portfolio_returns_mm
    return twoside

def mean_variance(pop):
    lambda_1=0.5
    delta=0.1
    return -lambda_1*vol(pop)+(1-lambda_1)*returns(pop)

def mean_mad(pop):
    lambda_1=0.5
    return -lambda_1*mean_absolute_deviation(pop)+(1-lambda_1)*returns(pop)

def value_at_risk(pop):
    alpha=0.05
    rend=returns(pop)
    stdev=vol(pop)
    var=norm.ppf(alpha,rend,stdev)*np.sqrt(21) #monthly VaR
    return var

def expected_shortfall(pop):
    alpha=0.05
    rend=returns(pop)
    std=vol(pop)
    es=-(alpha**-1*norm.pdf(norm.ppf(alpha))*std - rend)*np.sqrt(21) #monthly
        CVaR
    return es

def compute_entropy(pop):
    aux=np.zeros((chromosomes,genes))
    for i in range(len(pop)):
        for j in range(genes):
            aux[i,j]=-(pop[i,j]/chromosomes)*(np.log(pop[i,j]/chromosomes))/(np.
                log(2)*genes)
    aux1=np.sum(aux,axis=1)
    aux2=np.sum(aux1,axis=0)
    return aux2


def compute_pop_fitness(pop):
    cost=omega_ratio(pop)
##    cost=risk_parity(pop)
##    cost=twosided(pop)
##    cost=mean_variance(pop)
##    cost=mean_mad(pop)
##    cost=value_at_risk(pop)
##    cost=expected_shortfall(pop)
    return cost

##Selection

def elitist_selection(pop, fitness, num_parents):
    ##find max fitness solution, then select without replacement
    ## set a very low fitness to rule out replacement
    parents=np.zeros((num_parents, pop.shape[1]))
    for i in range(num_parents):
        pos_idx_max_fitness = np.where(fitness == np.max(fitness)) ##pos idx
        pos_idx_max_fitness = pos_idx_max_fitness[0][0]
        parents[i,:] = pop[pos_idx_max_fitness, :]
        fitness[pos_idx_max_fitness] = -10000
    return parents

##Set of functions implementing crossover. Each operator is sent to the
    controller to perform AOS

def crossover(parents, offspring_size): #Goldberg (1975)
    offspring=np.zeros(offspring_size)
    crossover_point=int(offspring_size[1]/2) ##len columns/2
```

```
    for i in range(offspring_size[0]): ##loop by rows
        parent1_pos = i%parents.shape[0]
        parent2_pos = (i+1)%parents.shape[0]
        offspring[i, :crossover_point] = parents[parent1_pos, :crossover_point]
        offspring[i, crossover_point:] = parents[parent2_pos, crossover_point:]
    return offspring


def two_point_crossover(parents,offspring_size): #Goldberg (1975), Muehlenberger
    (1993)
    offspring=np.zeros(offspring_size)
    for i in range(offspring_size[0]):
        parent1_pos=i%parents.shape[0]
        parent2_pos=(i+1)%parents.shape[0]
        crossover_point_1=np.random.randint(1,genes-1)
        crossover_point_2=np.random.randint(crossover_point_1,genes-1)
        if i%2==0:
            offspring[i,:crossover_point_1]=parents[parent1_pos,:
                crossover_point_1]
            offspring[i,crossover_point_1:crossover_point_2]=parents[parent2_pos,
                crossover_point_1:crossover_point_2]
            offspring[i,crossover_point_2:]=parents[parent1_pos,crossover_point_2
                :]
        else:
            offspring[i,:crossover_point_1]=parents[parent2_pos,:
                crossover_point_1]
            offspring[i,crossover_point_1:crossover_point_2]=parents[parent1_pos,
                crossover_point_1:crossover_point_2]
            offspring[i,crossover_point_2:]=parents[parent2_pos,crossover_point_2
                :]
    return offspring

def three_point_crossover(parents,offspring_size): #Muehlenberger (1993)
    offspring=np.zeros(offspring_size)
    for i in range(offspring_size[0]):
        parent1_pos=i%parents.shape[0]
        parent2_pos=(i+1)%parents.shape[0]
        crossover_point_1=int((offspring_size[1]-1)/3)
        crossover_point_2=int((2*offspring_size[1]-1)/3)
        crossover_point_3=int((3*offspring_size[1]-1)/3)
        if i%2==0:
            offspring[i,:crossover_point_1]=parents[parent1_pos,:
                crossover_point_1]
            offspring[i,crossover_point_1:crossover_point_2]=parents[parent2_pos,
                crossover_point_1:crossover_point_2]
            offspring[i,crossover_point_2:crossover_point_3]=parents[parent1_pos,
                crossover_point_2:crossover_point_3]
            offspring[i,crossover_point_3:]=parents[parent2_pos,crossover_point_3
                :]
        else:
            offspring[i,:crossover_point_1]=parents[parent2_pos,:
                crossover_point_1]
            offspring[i,crossover_point_1:crossover_point_2]=parents[parent1_pos,
                crossover_point_1:crossover_point_2]
            offspring[i,crossover_point_2:crossover_point_3]=parents[parent2_pos,
                crossover_point_2:crossover_point_3]
            offspring[i,crossover_point_3:]=parents[parent1_pos,crossover_point_3
                :]
    return offspring

def uniform_crossover(parents, offspring_size): #Spears, De Jong (1991)
    offspring=np.zeros(offspring_size)
    for i in range(offspring_size[0]):
        parent1_pos = i%parents.shape[0]
        parent2_pos = (i+1)%parents.shape[0]
        for j in range(genes):
            r=np.random.uniform(0,1)
            if r>0.5:
                offspring[i,j] = parents[parent1_pos,j]
            else:
                offspring[i,j] = parents[parent2_pos,j]
    return offspring


def global_uniform_crossover(parents, offspring_size): #Simon (2013)
    offspring=np.zeros(offspring_size)
    for i in range(offspring_size[0]):
        for j in range(genes):
            offspring[i,j]=random.choice(parents[:,j])
    return offspring

def flat_crossover(parents, offspring_size): #Herrera (1998)
    offspring=np.zeros(offspring_size)
    for i in range(offspring_size[0]):
```

```
        parent1_pos = i%parents.shape[0]
        parent2_pos = (i+1)%parents.shape[0]
        for j in range(genes):
                offspring[i,j] =np.random.uniform(min(parents[parent1_pos,j],
                    parents[parent2_pos,j]),max(parents[parent1_pos,j],parents[
                    parent2_pos,j]))
    return offspring

def blend_crossover(parents, offspring_size): #Houst (1995) & Herrera (1998)
    #alpha is a tunable parameter, which can be used to control the EvE balance
    alpha=0.5
    offspring=np.zeros(offspring_size)
    for i in range(offspring_size[0]):
        parent1_pos = i%parents.shape[0]
        parent2_pos = (i+1)%parents.shape[0]
        for j in range(genes):
                xmin=min(parents[parent1_pos,j],parents[parent2_pos,j])
                xmax=max(parents[parent1_pos,j],parents[parent2_pos,j])
                deltax=xmax-xmin
                offspring[i,j]=np.abs(np.random.uniform(xmin-(alpha*deltax),xmax
                    +(alpha*deltax)))
    return offspring

def average_crossover(parents, offspring_size): #Nomura (1997)
    offspring=np.zeros(offspring_size)
    for i in range(offspring_size[0]):
        parent1_pos = i%parents.shape[0]
        parent2_pos = (i+1)%parents.shape[0]
        for j in range(genes):
                offspring[i,j] =(parents[parent1_pos,j]+parents[parent2_pos,j])/2
    return offspring

def multi_parent_average_crossover(parents, offspring_size): #Nomura (1997)
    offspring=np.zeros(offspring_size)
    for i in range(offspring_size[0]):
        parent1_pos = i%parents.shape[0]
        parent2_pos = (i+1)%parents.shape[0]
        parent3_pos = (i+2)%parents.shape[0]
        for j in range(genes):
                offspring[i,j] =(parents[parent1_pos,j]+parents[parent2_pos,j]+
                    parents[parent3_pos,j])/3
    return offspring


def heuristic_crossover(parents, offspring_size): #Wright (1990)
    offspring=np.zeros(offspring_size)
    for i in range(offspring_size[0]):
        parent1_pos = i%parents.shape[0]
        parent2_pos = (i+1)%parents.shape[0]
        for j in range(genes):
            w=np.random.uniform(0,1)
            if parents[parent1_pos,j]>parents[parent2_pos,j]:
                offspring[i,j] =parents[parent2_pos,j]+w*(parents[parent1_pos,j]-
                    parents[parent2_pos,j])
            else:
                offspring[i,j] =parents[parent1_pos,j]+w*(parents[parent2_pos,j]-
                    parents[parent1_pos,j])
    return offspring

def arithmetic_crossover(parents, offspring_size): #Michalewicz (1996)
    beta=0.7
    offspring=np.zeros(offspring_size)
    for i in range(offspring_size[0]):
        parent1_pos = i%parents.shape[0]
        parent2_pos = (i+1)%parents.shape[0]
        for j in range(genes):
            if j%2==0:
                offspring[i,j]=beta*parents[parent1_pos,j]+(1-beta)*parents[
                    parent2_pos,j]
            else:
                offspring[i,j]=beta*parents[parent2_pos,j]+(1-beta)*parents[
                    parent1_pos,j]
    return offspring

def linear_crossover(parents, offspring_size): #Wright (1990)
    offspring=np.zeros(offspring_size)
    for i in range(offspring_size[0]):
        parent1_pos=i%parents.shape[0]
        parent2_pos=(i+1)%parents.shape[0]
        offspring1=np.zeros(genes)
        offspring2=np.zeros(genes)
        offspring3=np.zeros(genes)
        for j in range(genes):
```

```
            offspring1[j]=np.abs(0.5*parents[parent1_pos,j]+0.5*parents[
                parent2_pos,j])
            offspring2[j]=np.abs(1.5*parents[parent1_pos,j]-0.5*parents[
                parent2_pos,j])
            offspring3[j]=np.abs(-0.5*parents[parent1_pos,j]+1.5*parents[
                parent2_pos,j])
        offspring_matrix=np.stack((offspring1,offspring2,offspring3))
        offspring_fitness=compute_pop_fitness(offspring_matrix)
        pos_offspring=np.where(offspring_fitness==max(offspring_fitness))[0][0]
        offspring[i]=offspring_matrix[pos_offspring]
    return offspring

def simulated_binary_crossover(parents,offspring_size): #Deb and Agrawal (1995)
    #beta=1 --> 'stationary crossover'
    mu=0.05
    offspring=np.zeros(offspring_size)
    for i in range(offspring_size[0]):
        parent1_pos = i%parents.shape[0]
        parent2_pos = (i+1)%parents.shape[0]
        r=np.random.uniform(0,1)
        if r<0.5:
            beta=(2*r)**(1/(mu+1))
        else:
            beta=(2-2*r)**(-1/(mu+1))
        if i%2==0:
            offspring[i]=np.abs(0.5*((1-beta)*parents[parent1_pos]+(1+beta)*
                parents[parent2_pos]))
        else:
            offspring[i]=np.abs(0.5*((1+beta)*parents[parent1_pos]+(1-beta)*
                parents[parent2_pos]))
    return offspring

def queen_bee_crossover(parents,offspring_size):  ##Karc (2004)
    offspring=np.zeros(offspring_size)
    queen_bee=np.where(compute_pop_fitness(parents)==max((compute_pop_fitness(
        parents))))[0][0]
    parent1_pos=queen_bee
    for i in range(offspring_size[0]):
        parent2_pos=(parent1_pos+(i+1))%parents.shape[0]
        crossover_point_1=np.random.randint(1,genes-1)
        crossover_point_2=np.random.randint(crossover_point_1,genes-1)
        if i%2==0:
            offspring[i,:crossover_point_1]=parents[parent1_pos,:
                crossover_point_1]
            offspring[i,crossover_point_1:crossover_point_2]=parents[parent2_pos,
                crossover_point_1:crossover_point_2]
            offspring[i,crossover_point_2:]=parents[parent1_pos,crossover_point_2
                :]
        else:
            offspring[i,:crossover_point_1]=parents[parent2_pos,:
                crossover_point_1]
            offspring[i,crossover_point_1:crossover_point_2]=parents[parent1_pos,
                crossover_point_1:crossover_point_2]
            offspring[i,crossover_point_2:]=parents[parent2_pos,crossover_point_2
                :]
    return offspring

def laplace_crossover(parents,offspring_size): #Deep and Thakur (2007a)
    offspring=np.zeros(offspring_size)
    a=0
    b=5.0
    for i in range(offspring_size[0]):
        parent1_pos = i%parents.shape[0]
        parent2_pos = (i+1)%parents.shape[0]
        alpha=np.random.uniform(0,1)
        if alpha>0.5:
            beta=a-b*np.log(alpha)
        else:
            beta=a+b*np.log(alpha)
        if i%2==0:
            offspring[i]=np.abs(parents[parent1_pos]+beta*np.abs(parents[
                parent1_pos]-parents[parent2_pos]))
        else:
            offspring[i]=np.abs(parents[parent2_pos]+beta*np.abs(parents[
                parent1_pos]-parents[parent1_pos]))
    return offspring


def direction_based_crossover(parents, offspring_size): #Arumugam et al (2005)
    offspring=np.zeros(offspring_size)
    for i in range(offspring_size[0]): ##loop per riga
        r=np.random.uniform(0,1)
        parent1_pos = i%parents.shape[0]
        parent2_pos = (i+1)%parents.shape[0]
```

```
        vstack=np.vstack((parents[parent1_pos],parents[parent2_pos]))
        if compute_pop_fitness(vstack)[0]>=compute_pop_fitness(vstack)[1]:
            offspring[i]=r*(np.abs(parents[parent1_pos]-parents[parent2_pos]))+
                parents[parent2_pos]
        else:
            offspring[i]=r*(np.abs(parents[parent2_pos]-parents[parent1_pos]))+
                parents[parent1_pos]
    return offspring

def geometrical_crossover(parents, offspring_size): #Michalewicz et al.(1996)
    offspring=np.zeros(offspring_size)
    for i in range(offspring_size[0]):
        parent1_pos = i%parents.shape[0]
        parent2_pos = (i+1)%parents.shape[0]
        for j in range(genes):
            offspring[i,j] =np.sqrt(parents[parent1_pos,j]*parents[parent2_pos,j
                ])
    return offspring


def simplex_crossover(parents,offspring_size):
    offspring=np.zeros(offspring_size)
    for i in range(offspring_size[0]):
        parent1_pos=i%parents.shape[0]
        parent2_pos=(i+1)%parents.shape[0]
        parent3_pos=(i+2)%parents.shape[0]
        vstack=np.vstack((parents[parent1_pos],parents[parent2_pos],parents[
            parent3_pos]))
        idx_worst_fitness=np.where(compute_pop_fitness(vstack)==min(
            compute_pop_fitness(vstack)))[0][0]
        idx_best_fitness=np.where(compute_pop_fitness(vstack)==max(
            compute_pop_fitness(vstack)))[0][0]
        best_parents=np.delete(vstack,(idx_worst_fitness),axis=0)
        centroid=np.sum(best_parents,axis=0)/(len(vstack)-1)
        offspring[i]=centroid+(np.abs(centroid-vstack[idx_worst_fitness]))
    return offspring

def fuzzy_crossover(parents,offspring_size): #Voigt 1995
    offspring=np.zeros(offspring_size)
    d=0.5
    for i in range(offspring_size[0]):
        parent1_pos = i%parents.shape[0]
        parent2_pos = (i+1)%parents.shape[0]
        for j in range(genes):
            if parents[parent1_pos,j]<parents[parent2_pos,j]:
                phi_1=random.triangular(parents[parent1_pos,j]-d*np.abs(parents[
                    parent2_pos,j]-parents[parent1_pos,j]),parents[parent1_pos,j
                    ]+d*np.abs(parents[parent2_pos,j]-parents[parent1_pos,j]),
                    parents[parent1_pos,j])
                phi_2=random.triangular(parents[parent2_pos,j]-d*np.abs(parents[
                    parent2_pos,j]-parents[parent1_pos,j]),parents[parent2_pos,j
                    ]+d*np.abs(parents[parent2_pos,j]-parents[parent1_pos,j]),
                    parents[parent2_pos,j])
            else:
                phi_2=random.triangular(parents[parent1_pos,j]-d*np.abs(parents[
                    parent2_pos,j]-parents[parent1_pos,j]),parents[parent1_pos,j
                    ]+d*np.abs(parents[parent2_pos,j]-parents[parent1_pos,j]),
                    parents[parent1_pos,j])
                phi_1=random.triangular(parents[parent2_pos,j]-d*np.abs(parents[
                    parent2_pos,j]-parents[parent1_pos,j]),parents[parent2_pos,j
                    ]+d*np.abs(parents[parent2_pos,j]-parents[parent1_pos,j]),
                    parents[parent2_pos,j])
            offspring[i,j]=random.choice([np.abs(phi_1),np.abs(phi_2)])
    return offspring

def unimodal_crossover(parents,offspring_size): #Ono 1997
    offspring=np.zeros(offspring_size)
    std1=0.25
    std2=0.05
    for i in range(offspring_size[0]):
        parent1_pos = i%parents.shape[0]
        parent2_pos = (i+1)%parents.shape[0]
        parent3_pos = (i+2)%parents.shape[0]
        #x_p=0.5*(parents[parent1_pos]+parents[parent2_pos])
        g=0.5*(parents[parent1_pos]+parents[parent2_pos])
        d1=parents[parent1_pos]-0.95*g
        d2=parents[parent2_pos]-0.90*g
        d3=parents[parent3_pos]-0.85*g
        e1=d1/np.abs(d1)
        e2=d2/np.abs(d2)
        e3=d3/np.abs(d3)
        d=parents[parent2_pos]-parents[parent1_pos]
        aux=(np.random.normal(0,std1)*e1*np.abs(d1))+(np.random.normal(0,std1)*e2
            *np.abs(d2))
```

```
        D=(1-(np.dot(parents[parent3_pos]-parents[parent1_pos].T,parents[
            parent2_pos]-parents[parent1_pos])/np.dot(np.abs(parents[parent3_pos
            ]-parents[parent1_pos]),np.abs(parents[parent2_pos]-parents[
            parent1_pos])))**2)**0.5
        D=np.cross(np.abs(parents[parent3_pos]-parents[parent1_pos]),D)
        offspring[i,:]=np.abs(g+aux+np.random.normal(0,std2)*D*e3)
    return offspring

def parent_centric_normal_crossover(parents,offspring_size):
    offspring=np.zeros(offspring_size)
    eta=0.25
    for i in range(offspring_size[0]):
        parent1_pos = i%parents.shape[0]
        parent2_pos = (i+1)%parents.shape[0]
        for j in range(genes):
            w=np.random.uniform(0,1)
            if w<0.5:
                offspring[i,j]=np.abs(np.random.normal(parents[parent1_pos,j],np.
                    abs(parents[parent2_pos,j]-parents[parent1_pos,j])/eta))
            else:
                offspring[i,j]=np.abs(np.random.normal(parents[parent2_pos,j],np.
                    abs(parents[parent2_pos,j]-parents[parent1_pos,j])/eta))
    return offspring


list_variation_ops=[crossover,crossover_uniforme,heuristic_crossover,
              laplace_crossover,queen_bee_crossover,two_point_crossover,
                arithmetic_crossover,geometrical_crossover,
                simulated_binary_crossover,average_crossover,
                blend_crossover,flat_crossover,
                crossover_uniforme_globale,three_point_crossover,
                linear_crossover,direction_based_crossover,unimodal_crossover,
              fuzzy_crossover,simplex_crossover,parent_centric_crossover]


###AOS

#some parameters
genes=len(data.columns)
chromosomes=30 #number of solutions
num_parents=5 #number of parents for elitism
sim_ga=1
num_generations=1000
#population size
pop_size=(chromosomes,genes)
#offspring_size=(pop_size[0]-parents.shape[0]
#initialize some counters
counter_fitness_max=[]
counter_fitness_avg=[]
counter_entropy=[]
window=10
num_crossover=20

#initialize vectors and matrices to store operator
fitness_module1=np.zeros((num_generations,num_crossover))
entropy_module1=np.zeros((num_generations,num_crossover))
deltafitness_module1=np.zeros((num_generations-window,num_crossover))
deltaentropy_module1=np.zeros((num_generations-window,num_crossover))
credit_reward_list=np.zeros((num_generations-window,num_crossover))
credit_reward_aggregation=np.zeros((num_generations-2*window,num_crossover))


#mod1

def aggregated_criteria_computation(fitness_module1,entropy_module1):
    if i>=window:
        delta_avg_fitness_list=pd.DataFrame(fitness_module1[i-window:i,:]).diff()
        delta_entropy=pd.DataFrame(entropy_module1[i-window:i,:]).diff()
        Fwin1=delta_avg_fitness_list.mean()
        Fwin2=delta_entropy.mean()
        deltafitness_module1[i-window,:]=Fwin1
        deltaentropy_module1[i-window,:]=Fwin2
        return Fwin1, Fwin2, deltafitness_module1

#mod2

def reward_computation(theta):
    store_reward=np.zeros(num_crossover)
    if i>=window:
        for s in range(len(store_reward)):
            origin=[0,0]
            Fwin1=aggregated_criteria_computation(fitness_module1,entropy_module1
                )[0][s]
```

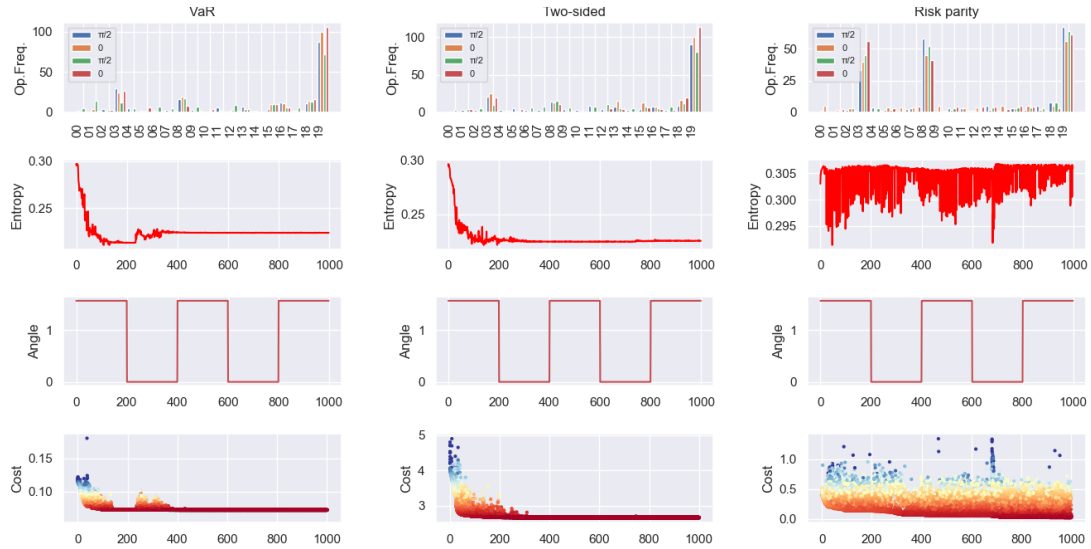
```
            Fwin2=aggregated_criteria_computation(fitness_module1,entropy_module1
                )[1][s]
            x=np.linspace(0,np.max(1.5*Fwin1),100)
            m=np.tan(theta) #slope
            y=m*x
            dp=abs((Fwin2-(m*Fwin1)))/(np.sqrt(1+m**2)) #perpendicular distance
            dpp=np.sqrt((Fwin1-origin[0])**2+(Fwin2-origin[1])**2) #distance
                between two points
            reward=np.sqrt(dpp**2-dp**2)
            store_reward[s]=reward
        return store_reward

##dynamic strategies for module 3

def increasing_strategy():
    angle=0
    if i>=window:
        if i<=num_generations/4:
            angle=0
        elif i>=num_generations/4 and i<=num_generations/2:
            angle=np.pi/6
        elif i>=num_generations/2 and i<=num_generations*(3/4):
            angle=np.pi/3
        elif i>=num_generations*(3/4):
            angle=np.pi/2
    return angle

def decreasing_strategy():
    angle=np.pi/2
    if i>=window:
        if i<=num_generations/4:
            angle=np.pi/2
        elif i>=num_generations/4 and i<=num_generations/2:
            angle=np.pi/3
        elif i>=num_generations/2 and i<=num_generations*(3/4):
            angle=np.pi/6
        elif i>=num_generations*(3/4):
            angle=0
    return angle

def always_moving_strategy():
    angle=np.pi/2
    if i>=window:
        if i<=num_generations/5:
            angle=np.pi/2
        elif i>=num_generations*(1/5) and i<=num_generations*(2/5):
            angle=0
        elif i>=num_generations*(2/5) and i<=num_generations*(3/5):
            angle=np.pi/2
        elif i>=num_generations*(3/5) and i<=num_generations*(4/5):
            angle=0
        elif i>=num_generations*(4/5):
            angle=np.pi/2
    return angle


def reactive_moving_strategy():
    angle=0
    if i>=window:
        if  ((entropy_list[i-1]-entropy_list[i-window])/(entropy_list[i-window]))
            <(-1/100):
            angle=0
        elif np.abs((avg_fitness_list[i-1]-avg_fitness_list[i-window])/(
            avg_fitness_list[i-window]))<(1/100):
            angle=np.pi/2
        else:
            angle=0
    return angle


#mod3

def credit_assignment(strategy):
    if i>=window:
        angle=strategy
        store_reward=reward_computation(angle)
        prova[i]=store_reward[19]
        credit_reward_list[i-window,:]=store_reward
    if i>=2*window:
        credit_reward_aggregation[i-2*window,:]=np.mean(pd.DataFrame(
            credit_reward_list[i-2*window:i-window,:]).dropna())
    return credit_reward_aggregation
```

```
#mod4

def operator_selection(credit_function): #Probability Matching (PM)
        p_min=0.01
        K=num_crossover
        idx=np.random.randint(0,19)
        credito=credit_function
        if i>=2*window:
            #credito=credit_assignment()
            wheel_selection=p_min+(1-K*p_min)*(credito[i-2*window,:]/(np.sum(
                credito[i-2*window,:])))
            #print(wheel_selection)
            wheel_selection=np.cumsum(wheel_selection)
            u=np.random.uniform(0,1)
            for c in range(len(wheel_selection)):
                if u<wheel_selection[c]:
                    idx=c
                    #print(idx)
                    break
    return idx

###END AOS


#some string variables and initialization of int constraints
selection='elitist'
strategy='always'
epsilon=1.0e-001
app1=np.zeros((chromosomes,genes))
app2=np.zeros((chromosomes,genes))
app3=np.zeros((chromosomes,genes))
vinc1=np.zeros(chromosomes)
vinc2=np.zeros(chromosomes)
vinc3=np.zeros(chromosomes)
vinc4=np.zeros(chromosomes)
vinc5=np.zeros(chromosomes)
K_u=10
perc_min=np.ones(genes)*0.05
perc_max=np.ones(genes)*0.15


start=time.time()

for w in range(1):

    max_fitness_outer=[]
    avg_fitness_outer=[]
    entropy_list_outer=[]
    idx=0
    store_operators_matrix=np.zeros((num_crossover,4))
    store_angle=np.zeros(num_generations)
    individual_fitness=np.zeros((num_generations,chromosomes))
    counter1=0


    for k in range(sim_ga):

        new_pop=np.random.uniform(low=0, high=1, size=pop_size)
        fitness_max=[]
        avg_fitness_list=[]
        entropy_list=[]

        for i in range(num_generations):
            for a in range(chromosomes):
                for b in range(genes):
                    app1[a,b]=max(0,perc_min[b]-new_pop[a,b])
                    app2[a,b]=max(0,new_pop[a,b]-perc_max[b])
                    app3[a,b]=abs(zeta[a,b]*(1-zeta[a,b]))
                vinc1[a]=abs(np.sum(new_pop[a,:])-1)
                vinc2[a]=max(0,np.sum(zeta[a,:])-K_u)
                vinc3[a]=np.sum(app1[a,:])
                vinc4[a]=np.sum(app2[a,:])
                vinc5[a]=np.sum(app3[a,:])
            fitness=compute_pop_fitness(new_pop)-((1/epsilon)*(vinc1+vinc2+vinc3+
                vinc4+vinc5))
            individual_fitness[i,:]=-fitness
            entropy=calcola_entropy(new_pop)
            average_fitness=np.sum(fitness)/chromosomes
            if selection=='elitist':
                parents=elitist_selection(new_pop,fitness,num_parents)
                offspring_crossover=list_variation_ops[idx](parents,
                    offspring_size=(pop_size[0]-parents.shape[0], genes))
                crossover_sols=np.zeros((num_crossover,chromosomes,data.shape[1])
                    )
```

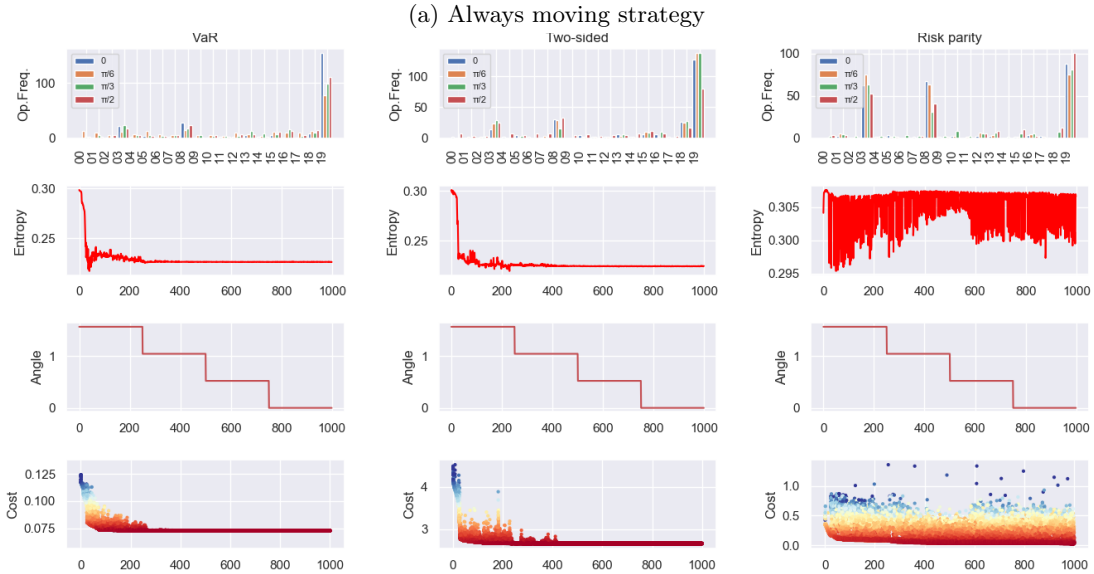
```
                for opti in range(num_crossover):
                    crossover_sols[opti,:,:]=list_variation_ops[opti](parents,
                        offspring_size=(pop_size[0], genes))
            avg_fitness_20=np.array([np.sum(compute_pop_fitness(crossover_sols[u
                ]))/chromosomes for u in range(num_crossover)])
            entropy_20=np.array([calcola_entropy(crossover_sols[u]) for u in
                range(num_crossover)])
            fitness_module1[i,:]=avg_fitness_20
            entropy_module1[i,:]=entropy_20
            if strategy=='reactive':
                current_angle=reactive_moving_strategy()
                store_angle[i]=current_angle
                idx=operator_selection(credit_assignment(reactive_moving_strategy
                    ()))
            elif strategy=='always':
                current_angle=always_moving_strategy()
                store_angle[i]=current_angle
                idx=operator_selection(credit_assignment(always_moving_strategy()
                    ))
            elif strategy=='decreasing':
                current_angle=decreasing_strategy()
                store_angle[i]=current_angle
                idx=operator_selection(credit_assignment(decreasing_strategy()))
            elif strategy=='increasing':
                current_angle=increasing_strategy()
                store_angle[i]=current_angle
                idx=operator_selection(credit_assignment(increasing_strategy()))
            print(idx)
            if i>0:
                if counter1==0:
                    store_operators_matrix[idx,counter1]=store_operators_matrix[
                        idx,counter1]+1
                if store_angle[i]!=store_angle[i-1]:
                    counter1=counter1+1
                if counter1==1:
                    store_operators_matrix[idx,counter1]=store_operators_matrix[
                        idx,counter1]+1
                if counter1==2:
                    store_operators_matrix[idx,counter1]=store_operators_matrix[
                        idx,counter1]+1
                if counter1==3:
                    store_operators_matrix[idx,counter1]=store_operators_matrix[
                        idx,counter1]+1
            if selection=='elitist':
                new_pop[:parents.shape[0], :] = parents
                new_pop[parents.shape[0]:, :] = offspring_crossover
            idx_best_fitness=np.where(fitness == np.max(fitness))
            idx_best_fitness=idx_best_fitness[0][0]
            avg_fitness_list.append(average_fitness)
            entropy_list.append(entropy)
        avg_fitness_outer.append(avg_fitness_list)
        entropy_list_outer.append(entropy_list)


    df_avg_fitness_list=pd.DataFrame(np.transpose(avg_fitness_outer))
    avg_sim_fitness_list=np.mean(df_avg_fitness_list,axis=1)
    df_entropy=pd.DataFrame(np.transpose(entropy_list_outer))
    avg_sim_entropy_list=np.mean(df_entropy,axis=1)
    counter_fitness_avg.append(avg_sim_fitness_list)
    counter_entropy.append(avg_sim_entropy_list)
    print(avg_sim_fitness_list)
    end=time.time()
    print(end-start)
```

Appendix C

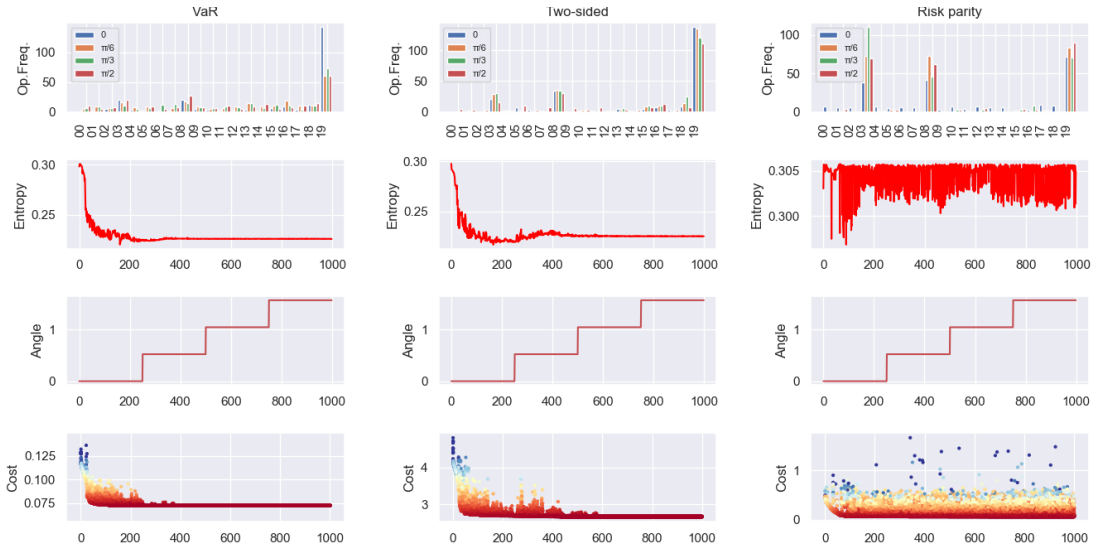
Figures



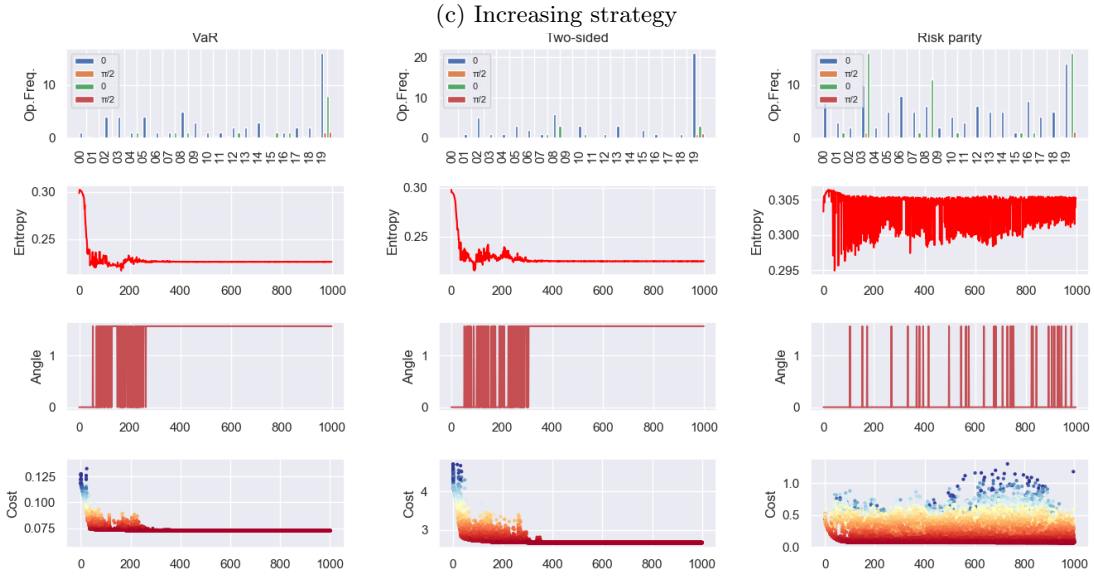
(a) Always moving strategy



(b) Decreasing strategy

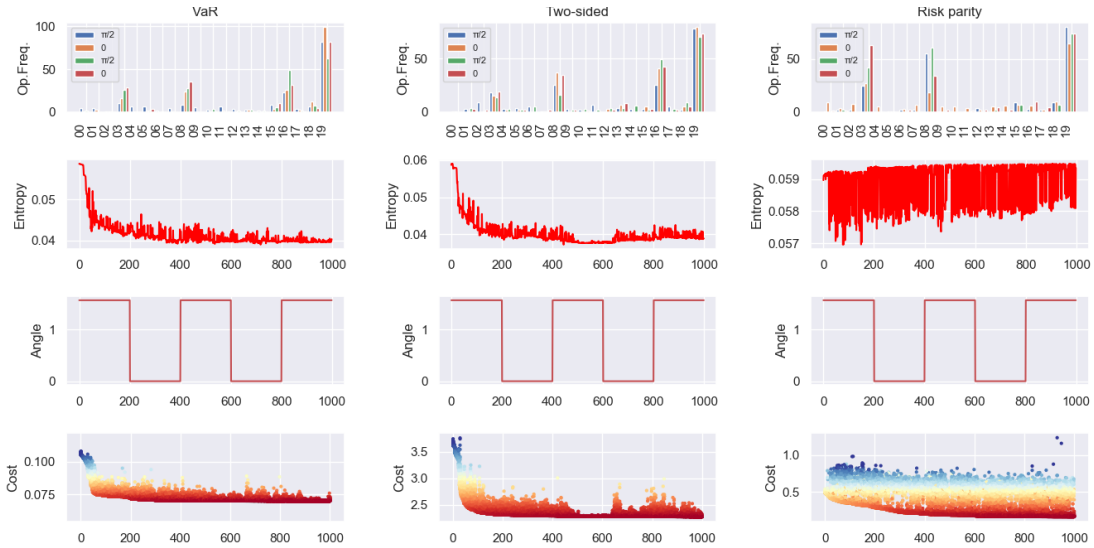


(c) Increasing strategy

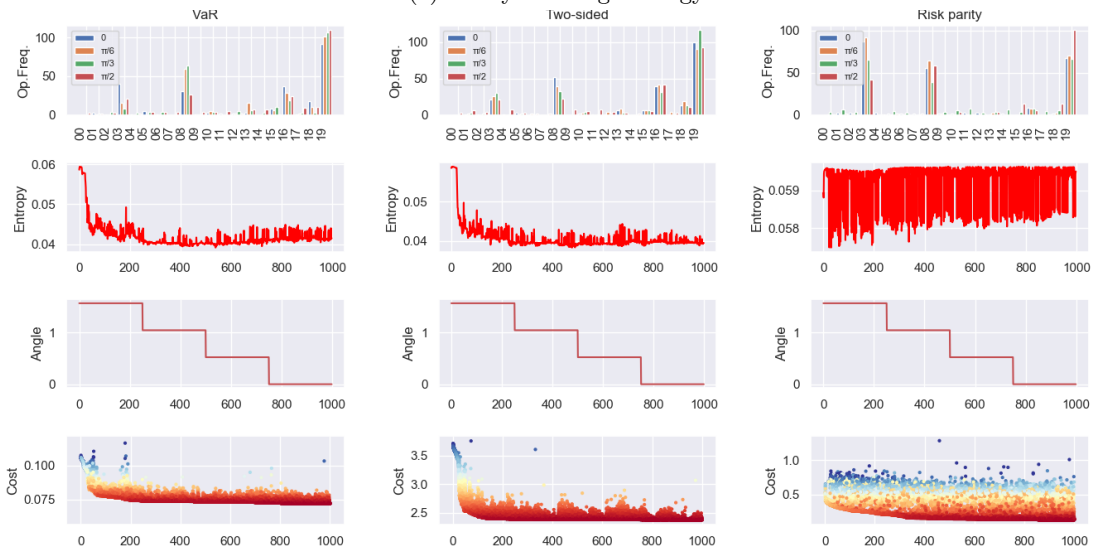


(d) Reactive moving strategy

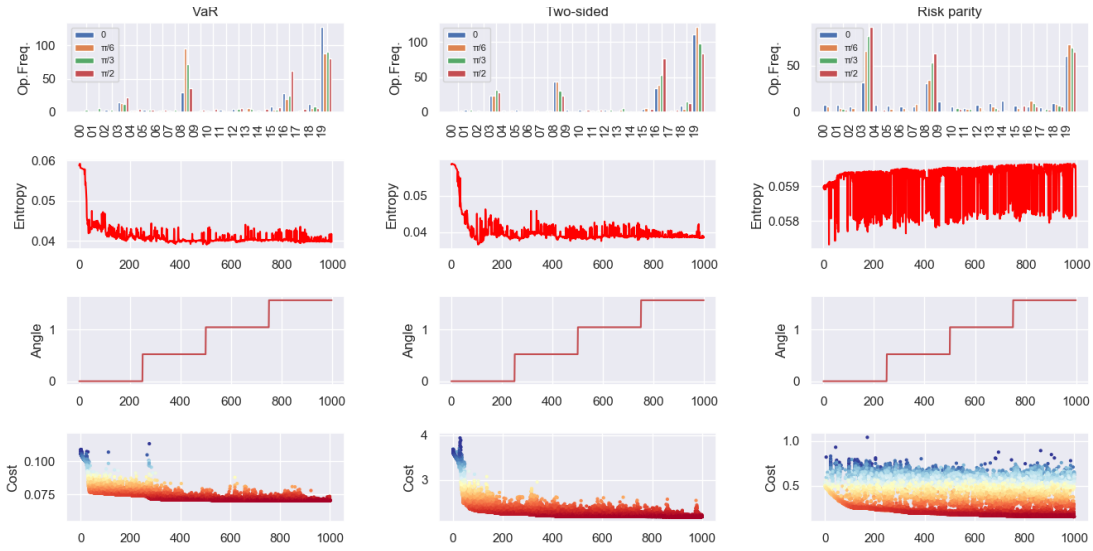
Figure C.1: Experiments with the AOS: every portfolio selection strategy is fitted to the FTSE MIB dataset. From top to bottom: histogram of the operator selection frequency (1), the entropy convergence curve (2), the dynamic angle function driving the search process (3), the individual cost scatter plot (4).



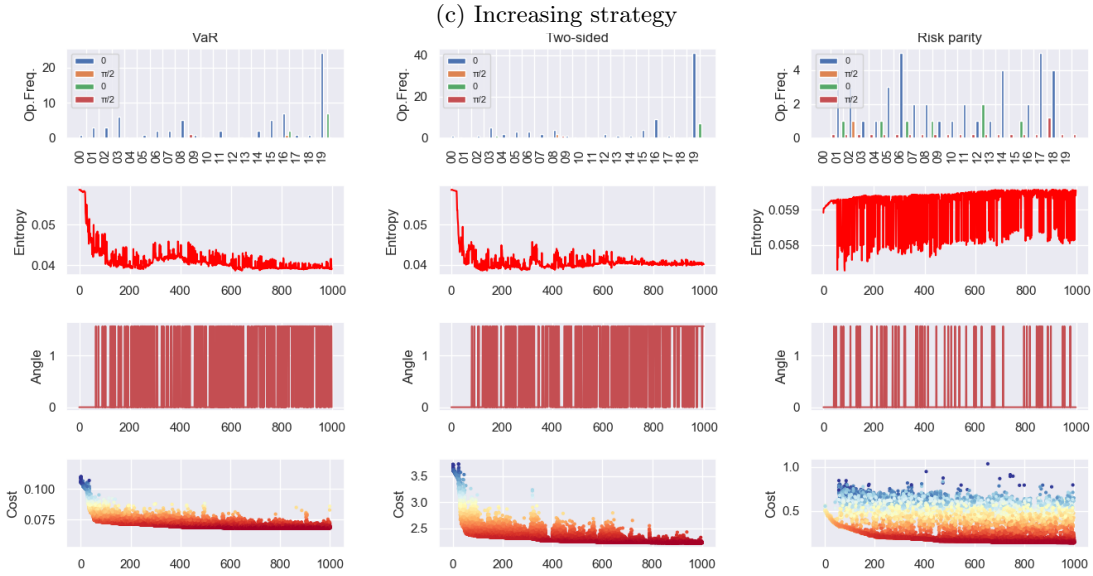
(a) Always moving strategy



(b) Decreasing strategy

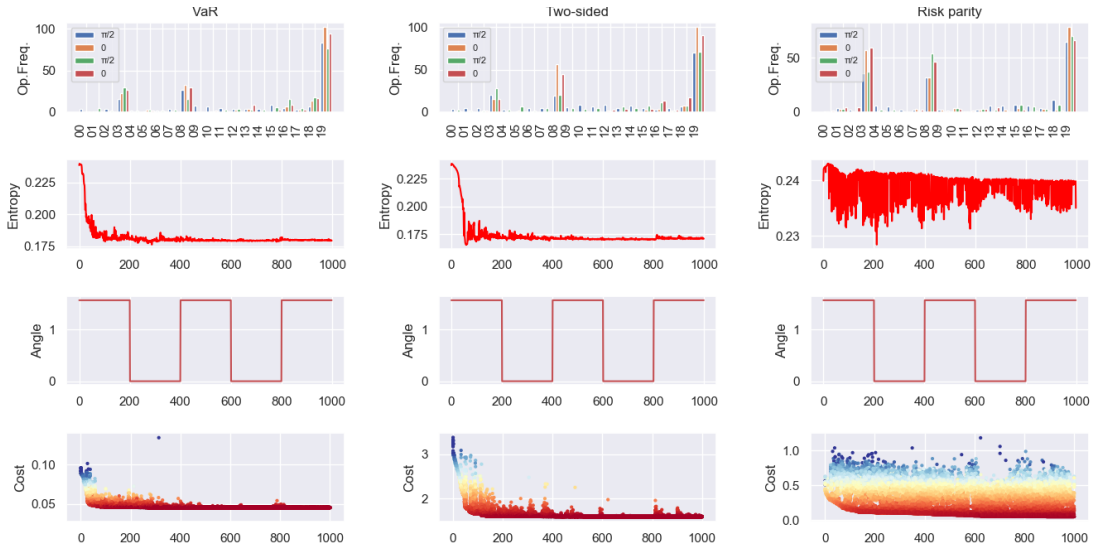


(c) Increasing strategy

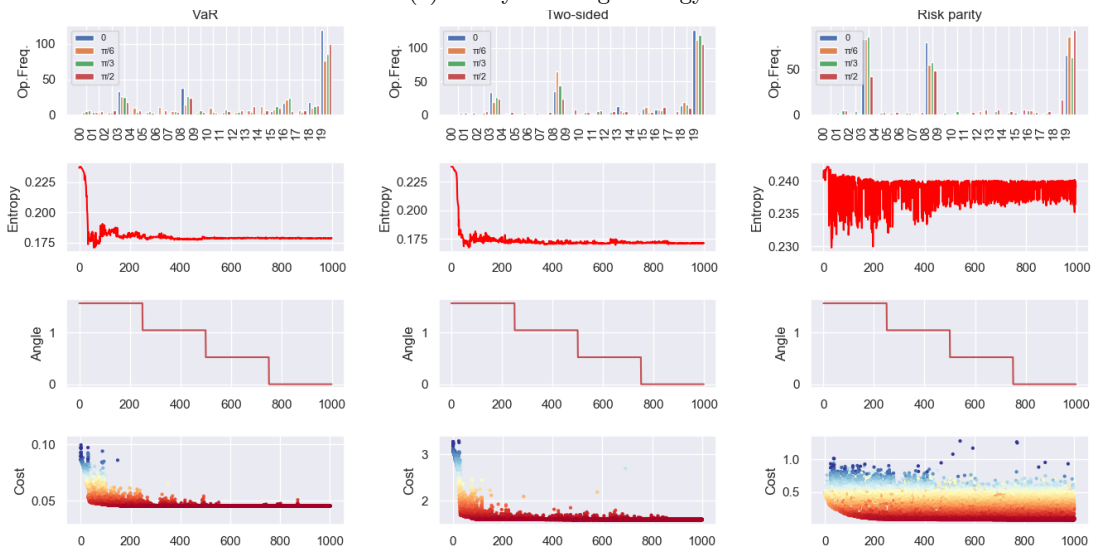


(d) Reactive moving strategy

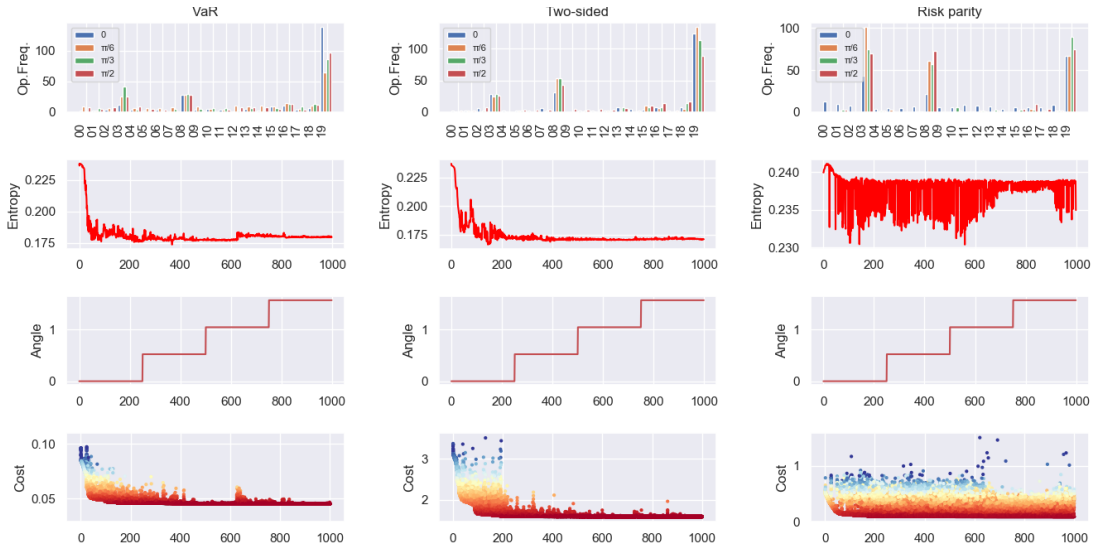
Figure C.2: Experiments with the AOS: every portfolio selection strategy is fitted to the Nikkei 225 dataset. From top to bottom: histogram of the operator selection frequency (1), the entropy convergence curve (2), the dynamic angle function driving the search process (3), the individual cost scatter plot (4).



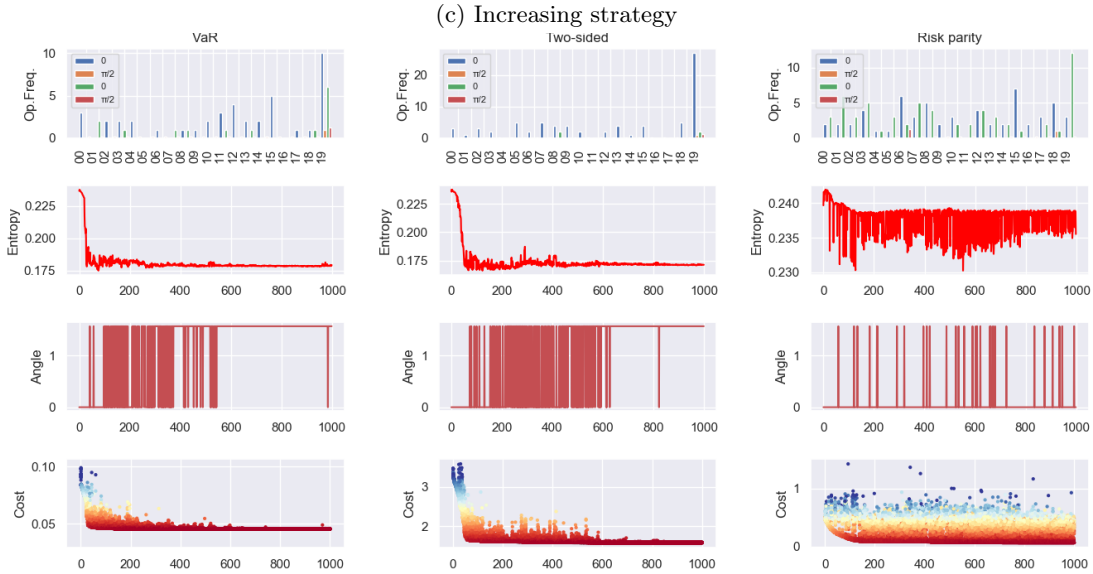
(a) Always moving strategy



(b) Decreasing strategy

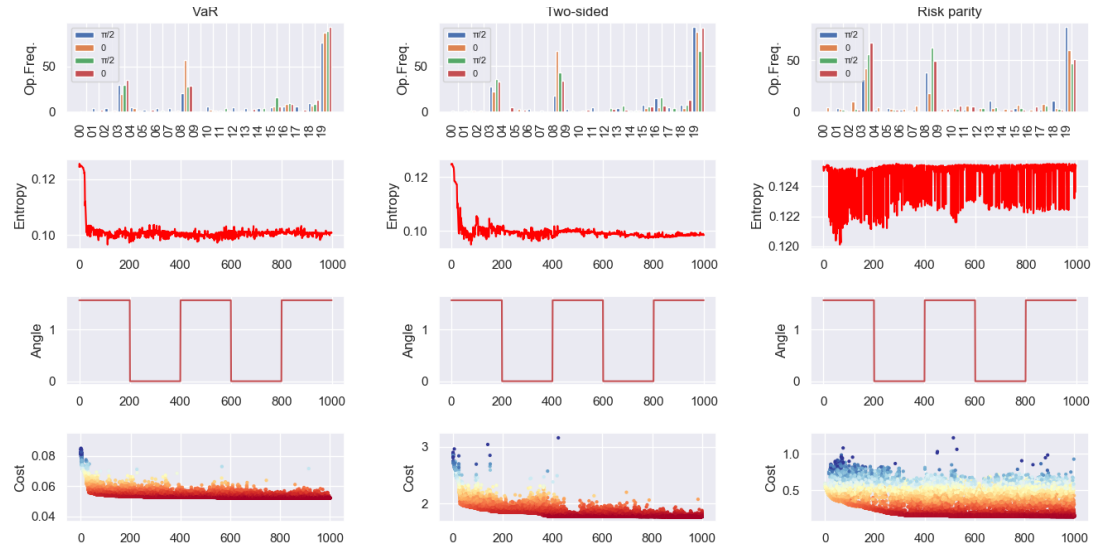


(c) Increasing strategy

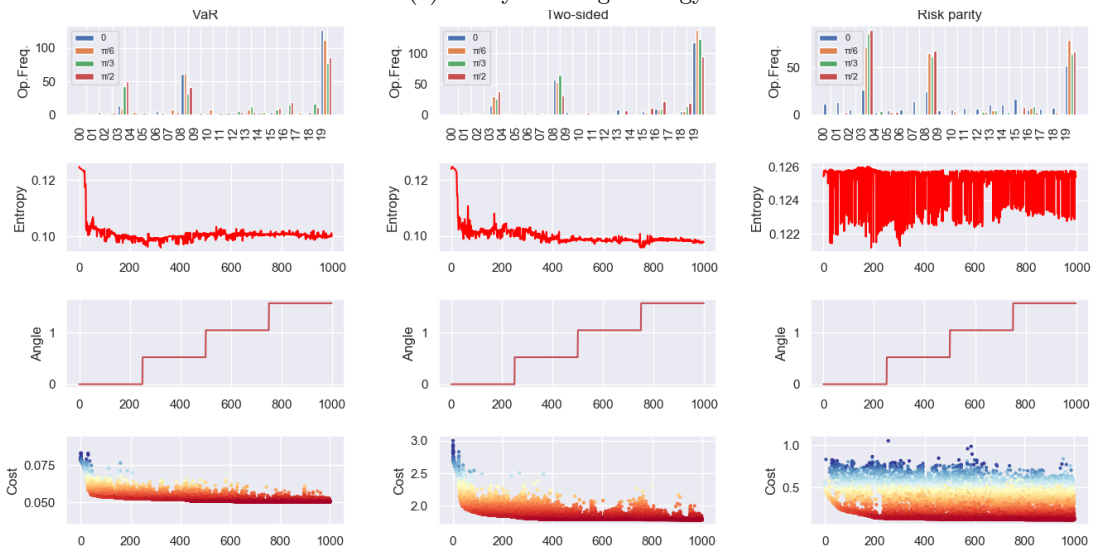


(d) Reactive moving strategy

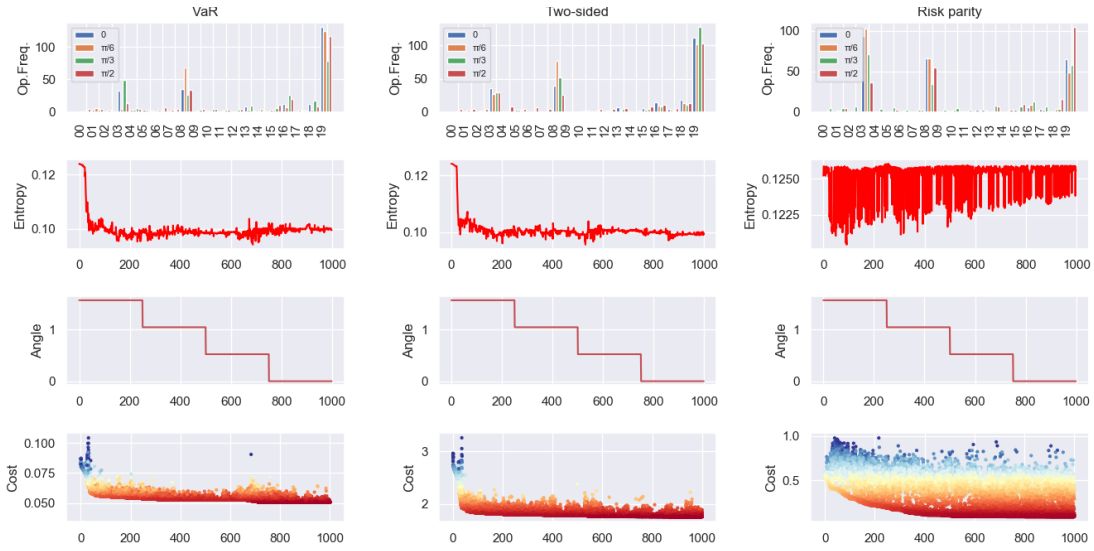
Figure C.3: Experiments with the AOS: every portfolio selection strategy is fitted to the Hang Seng dataset. From top to bottom: histogram of the operator selection frequency (1), the entropy convergence curve (2), the dynamic angle function driving the search process (3), the individual cost scatter plot (4).



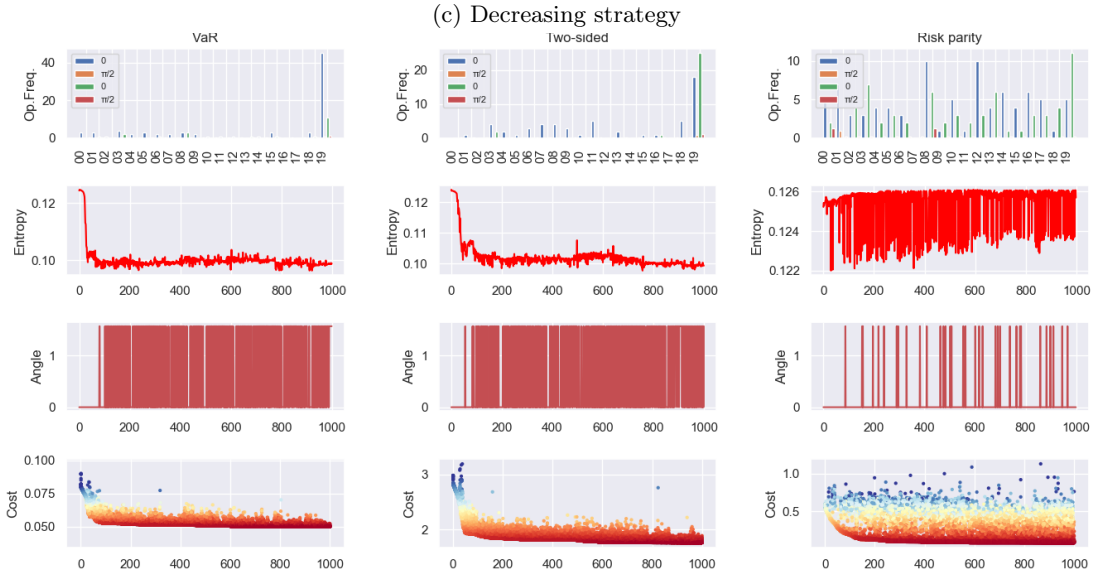
(a) Always moving strategy



(b) Increasing strategy



(c) Decreasing strategy



(d) Reactive moving strategy

Figure C.4: Experiments with the AOS: every portfolio selection strategy is fitted to the FTSE 100 dataset. From top to bottom: histogram of the operator selection frequency (1), the entropy convergence curve (2), the dynamic angle function driving the search process (3), the individual cost scatter plot (4).

Bibliography

Adenso-Díaz, B. and M. Laguna (2006). "Fine-tuning of algorithms using fractional experimental designs and local search". In: *Operations Research* 54(1), pp. 99–114.

Aleti, A. and I. Moser (2011). "Predictive parameter control". In: *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*. Ed. by N. Krasnogor.

— (2016). "A systematic literature review of adaptive parameter control methods for evolutionary algorithms". In: *ACM Computing Surveys* 49, pp. 1–35.

Alpaydin, E. (2019). *Introduction to Machine Learning*. MIT Press.

Anagnostopoulos, K. and G. Mamanis (2011). "A portfolio optimization model with three objectives and discrete variables". In: *Computers & Operations Research* 37.7, pp. 1285–1297.

Angeline, P.J. (1995). "Adaptive and Self-Adaptive Evolutionary Computations". In: *Computational Intelligence: A Dynamic Systems Perspective*. IEEE Press, pp. 152–163.

Arabas, J., Z. Michalewicz, and J. K. Mulawka (1994). "GAVaPS - A Genetic Algorithm with Varying Population Size". In: *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, Orlando, Florida, USA, June 27-29, 1994*. IEEE, pp. 73–78.

Artzner, P. et al. (1999). "Coherent measures of risk". In: *Mathematical finance* 9.3, pp. 203–228.

Arumugam, M. S., M. Rao, and R. Palaniappan (2005). "New hybrid genetic operators for real coded genetic algorithm to compute optimal control of a class of hybrid systems". In: *Appl. Soft Comput.* 6, pp. 38–52.

Bäck, T. (1993). "Optimal Mutation Rates in Genetic Search". In: *Proceedings of the 5th International Conference on Genetic Algorithms, Urbana-Champaign, IL, USA, June 1993*. Ed. by S. Forrest. Morgan Kaufmann, pp. 2–8.

Bäck, T., A. E. Eiben, and N. A. L. van der Vaart (2000). "An Empirical Study on GAs without Parameters". In: *Parallel Problem Solving from Nature, VI, 6th International Conference, Paris, France, September 18-20, 2000, Proceedings*. Vol. 1917. Lecture Notes in Computer Science. Springer, pp. 315–324.

Baker, B. and M. Ayechew (2003). "A genetic algorithm for the vehicle routing problem". In: *Computers & Operations Research* 30.5, pp. 787–800.

Balaprakash, P., M. Birattari, and T. Stützle (2007). "Improvement Strategies for the F-Race Algorithm: Sampling Design and Iterative Refinement". In: *Hybrid Metaheuristics, 4th International Workshop, HM 2007, Dortmund, Germany, October 8-9, 2007, Proceedings*. Vol. 4771. Lecture Notes in Computer Science. Springer, pp. 108–122.

Ballester, P.J. and J.N Carter (2004). "An effective real-parameter genetic algorithm with parent centric normal crossover for multimodal optimisation". In: *Genetic and Evolutionary Computation Conference*. Springer, pp. 901–913.

Baluja, S. and R. Caruana (1995). "Removing the Genetics from the Standard Genetic Algorithm". In: *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995*. Morgan Kaufmann, pp. 38–46.

Bazaraa, M., H. Sherali, and C. Shetty (2013). *Nonlinear programming: theory and algorithms*. John Wiley & Sons.

Ben Hadj-Alouane, A. and J. C. Bean (1997). "A Genetic Algorithm for the Multiple-Choice Integer Program". In: *Operations Research* 45.1, pp. 92–101.

Beyer, H. and K. Deb (2001). "On self-adaptive features in real-parameter evolutionary algorithms". In: *IEEE Trans. Evol. Comput.* 5.3, pp. 250–270.

Birattari, M. (2003). "The race package for R Racing methods for the selection of the best".

Birattari, M. et al. (2002). "A Racing Algorithm for Configuring Metaheuristics". In: *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, USA, 9-13 July 2002*. Morgan Kaufmann, pp. 11–18.

Borges, B. and J. Knetsch (1998). "Tests of market outcomes with asymmetric valuations of gains and losses: Smaller gains, fewer trades, and less value". In: *Journal of Economic Behavior & Organization* 33.2, pp. 185–193.

Chang, T-J. et al. (2000). "Heuristics for cardinality constrained portfolio optimisation". In: *Computers & Operations Research* 27.13, pp. 1271–1302.

Chang, T-J., S-C. Yang, and K-J. Chang (2009). "Portfolio optimization problems in different risk measures using genetic algorithm". In: *Expert Systems with applications* 36.7, pp. 10529–10537.

Chen, Z. and Y. Wang (2008). "Two-sided coherent risk measures and their application in realistic portfolio optimization". In: *Journal of Banking & Finance* 32.12, pp. 2667–2673.

Coello-Coello, C. A. (2002). "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art". In: *Computer methods in applied mechanics and engineering* 191.11-12, pp. 1245–1287.

Corazza, M., G. Fasano, and R. Gusso (2013). "Particle Swarm Optimization with non-smooth penalty reformulation, for a complex portfolio selection problem". In: *Applied Mathematics and Computation* 224, pp. 611–624.

Corazza, M. et al. (2021). "A novel hybrid PSO-based metaheuristic for costly portfolio selection problems". In: *Annals of Operations Research*, pp. 1–29.

Costa, J. C., R. Tavares, and A. Rosa (1999). "An experimental study on dynamic random variation of population size". In: *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man and Cybernetics*. Vol. 1, 607–612 vol.1.

Costa, L. Da et al. (2008). "Adaptive operator selection with dynamic multi-armed bandits". In: *Genetic and Evolutionary Computation Conference, GECCO 2008, Proceedings*. ACM, pp. 913–920.

Davis, L. (1989). "Adapting Operator Probabilities in Genetic Algorithms". In: *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers Inc., 61–69.

De Jong, K.A. (1975). "An Analysis of the Behavior of a Class of Genetic Adaptive Systems." PhD thesis. University of Michigan, USA.

— (2007). "Parameter Setting in EAs: a 30 Year Perspective". In: *Parameter Setting in Evolutionary Algorithms*.

Deb, K. and H.G Beyer (2001). "Self-adaptive genetic algorithms with simulated binary crossover". In: *Evolutionary computation* 9.2, pp. 197–221.

Deb, K., R.B. Agrawal, and R. Bhushan (1995). "Simulated binary crossover for continuous search space". In: *Complex systems* 9.2, pp. 115–148.

Deb, K., A. Anand, and D. Joshi (2002). "A Computationally Efficient Evolutionary Algorithm for Real-Parameter Optimization". In: *Evolutionary Computation* 10.4, pp. 371–395.

Deep, K. and M. Thakur (2007). "A new crossover operator for real coded genetic algorithms". In: *Applied mathematics and computation* 188.1, pp. 895–911.

DeMiguel, V. et al. (2009). "A generalized approach to portfolio optimization: Improving performance by constraining portfolio norms". In: *Management science* 55.5, pp. 798–812.

di Tollo, G. and A. Roli (2008). "Metaheuristics for the portfolio selection problem". In: *International Journal of Operations Research* 5.1, pp. 13–35.

di Tollo, G. et al. (2015). "An experimental study of adaptive control for evolutionary algorithms". In: *Applied Soft Computing* 35, pp. 359–372.

Dobslaw, F. (2010). "Recent Development in Automatic Parameter Tuning for Metaheuristics". In:

Eiben, A. E. and S. K. Smit (2011). "Parameter tuning for configuring and analyzing evolutionary algorithms". In: *Swarm Evol. Comput.* 1.1, pp. 19–31.

— (2012). "Evolutionary Algorithm Parameters and Methods to Tune Them". In: *Autonomous Search*. Ed. by Y. Hamadi, E. Monfroy, and F. Saubion. Springer, pp. 15–36.

Eiben, A. E. and J. E. Smith (2015). *Introduction to Evolutionary Computing, Second Edition*. Natural Computing Series. Springer.

Eiben, A. E., R. Hinterding, and Z. Michalewicz (1999). "Parameter control in evolutionary algorithms". In: *IEEE Transactions on Evolutionary Computation* 3.2, pp. 124–141.

Eiben, A. E., M. Schut, and A. D. Wilde (2006a). "Is Self-adaptation of Selection Pressure and Population Size Possible? - A Case Study". In: *PPSN*.

Eiben, A. E. et al. (2007). "Parameter Control in Evolutionary Algorithms". In: *Parameter Setting in Evolutionary Algorithms*. Ed. by F. G. Lobo, C. F. Lima, and Z. Michalewicz. Vol. 54. Studies in Computational Intelligence. Springer, pp. 19–46.

Eiben, A.E., E. Marchiori, and V.A. Valko (2004). "Evolutionary Algorithms with on-the-fly Population Size Adjustment". In: *Proceedings of the 8th international conference on Parallel Problem Solving from Nature (PPSN VIII)*. Ed. by X. Yao. Springer, pp. 41–50.

Eiben, A.E. et al. (2006b). "Reinforcement Learning for Online Control of Evolutionary Algorithms". In: *Proceedings of the 4th International Conference on Engineering Self-Organising Systems*. ESOA'06. Berlin, Heidelberg: Springer-Verlag, pp. 151–160.

Eiben, G. and M. C. Schut (2008). "New Ways to Calibrate Evolutionary Algorithms". In: *Advances in Metaheuristics for Hard Optimization*. Ed. by P. Siarry and Z. Michalewicz. Natural Computing Series. Springer, pp. 153–177.

Eshelman, L.J. and J.D. Schaffer (1993). "Real-coded genetic algorithms and interval-schemata". In: *Foundations of genetic algorithms*. Vol. 2. Elsevier, pp. 187–202.

Farmani, R. and J. A. Wright (2003). "Self-adaptive fitness formulation for constrained optimization". In: *IEEE Transactions on Evolutionary Computation* 7.5, pp. 445–455.

Fialho, Á. (2010). "Adaptive Operator Selection for Optimization". PhD thesis. University of Paris-Sud, Orsay, France.

Fishburn, P.C. (1977). "Mean-risk analysis with risk associated with below-target returns". In: *The American Economic Review* 67.2, pp. 116–126.

Fletcher, R. (2013). *Practical methods of optimization*. John Wiley & Sons.

García-Martínez, C. et al. (2008). "Global and local real-coded genetic algorithms based on parent-centric crossover operators". In: *Eur. J. Oper. Res.* 185.3, pp. 1088–1113.

Gilli, M. and E. Schumann (2021). "Risk–Reward Ratio Optimisation (Revisited)". In: *Dynamic Analysis in Complex Economic Environments*. Springer, pp. 29–57.

Gilli, M., E. Këllezi, and H. Hysi (2006). "A data-driven optimization heuristic for downside risk minimization". In: *Swiss Finance Institute Research Paper* 06-2.

Gilli, M. et al. (2011). "Constructing 130/30-portfolios with the Omega ratio". In: *Journal of asset management* 12.2, pp. 94–108.

Goldberg, D. (1988). "Genetic Algorithms in Search Optimization and Machine Learning". In:

Harik, G. R., F. G. Lobo, and D. E. Goldberg (1999). "The compact genetic algorithm". In: *IEEE Transactions on Evolutionary Computation* 3.4, pp. 287–297.

Hatta, K. et al. (1997). "On-the-fly crossover adaptation of genetic algorithms". In:

Herrera, F., M. Lozano, and J.L. Verdegay (1998). "Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis". In: *Artificial intelligence review* 12.4, pp. 265–319.

Herrera, F., M. Lozano, and A.M. Sánchez (2005). "Hybrid crossover operators for real-coded genetic algorithms: an experimental study". In: *Soft Comput.* 9.4, pp. 280–298.

Hinterding, R., Z. Michalewicz, and A. E. Eiben (1997). "Adaptation in evolutionary computation: a survey". In: *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97)*, pp. 65–69.

Holland, J.H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press.

Hoos, H. H. (2012). "Automated Algorithm Configuration and Parameter Tuning". In: *Autonomous Search*. Ed. by Y. Hamadi, E. Monfroy, and F. Saubion. Springer, pp. 37–71.

Huang, C., Y. Li, and X. Yao (2020). "A Survey of Automatic Parameter Tuning Methods for Metaheuristics". In: *IEEE Transactions on Evolutionary Computation* 24.2, pp. 201–216.

Hutter, F. et al. (2007). "Automatic Algorithm Configuration Based on Local Search". In: *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2*. AAAI'07. Vancouver, British Columbia, Canada: AAAI Press, 1152–1157.

Hutter, F. et al. (2009). "ParamILS: An Automatic Algorithm Configuration Framework". In: *J. Artif. Intell. Res.* 36, pp. 267–306.

Jagannathan, R. and T. Ma (2003). "Risk reduction in large portfolios: Why imposing the wrong constraints helps". In: *The Journal of Finance* 58.4, pp. 1651–1683.

Joines, J. A. and C. R. Houck (1994). "On the Use of Non-Stationary Penalty Functions to Solve Nonlinear Constrained Optimization Problems with GA's". In: *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, Orlando, Florida, USA, June 27-29, 1994*. IEEE, pp. 579–584.

Kahneman, D. and A. Tversky (2013). "Prospect theory: An analysis of decision under risk". In: *Handbook of the fundamentals of financial decision making: Part I*. World Scientific, pp. 99–127.

Karafotias, G., A. E. Eiben, and M. Hoogendoorn (2014). "Generic parameter control with reinforcement learning". In: *Genetic and Evolutionary Computation Conference, GECCO '14, Vancouver, BC, Canada, July 12-16, 2014*. Ed. by D. V. Arnold. ACM, pp. 1319–1326.

Karafotias, G., M. Hoogendoorn, and A. E. Eiben (2015). "Parameter Control in Evolutionary Algorithms: Trends and Challenges". In: *IEEE Trans. Evol. Comput.* 19.2, pp. 167–187.

Kaucic, M. (2019). "Equity portfolio management with cardinality constraints and risk parity control using multi-objective particle swarm optimization". In: *Computers & Operations Research* 109, pp. 300–316.

Kazarlis, S. A. and V. Petridis (1998). "Varying Fitness Functions in Genetic Algorithms: Studying the Rate of Increase of the Dynamic Penalty Terms". In: *Parallel Problem Solving from Nature - PPSN V, 5th International Conference, Amsterdam, The Netherlands, September 27-30, 1998, Proceedings*. Ed. by A. E. Eiben et al. Vol. 1498. Lecture Notes in Computer Science. Springer, pp. 211–220.

Keating, C. and K.W. Shadwick (2002). "A universal performance measure". In: *Journal of performance measurement* 6.3, pp. 59–84.

Kita, H. (2001). "A comparison study of self-adaptation in evolution strategies and real-coded genetic algorithms". In: *Evolutionary Computation* 9.2, pp. 223–241.

Kita, H. and M. Yamamura (1999). "A functional specialization hypothesis for designing genetic algorithms". In: *IEEE SMC'99 Conference Proceedings*. Vol. 3, 579–584 vol.3.

Kita, H., I. Ono, and S. Kobayashi (1999). "Theoretical analysis of the unimodal normal distribution crossover for real-coded genetic algorithms". In: *Transactions of the Society of Instrument and Control Engineers* 35.11, pp. 1333–1339.

Konno, H. and H. Yamazaki (1991). "Mean-absolute deviation portfolio optimization model and its applications to Tokyo stock market". In: *Management science* 37.5, pp. 519–531.

Koumousis, V.K. and C. P. Katsaras (2006). "A Saw-Tooth Genetic Algorithm Combining the Effects of Variable Population Size and Reinitialization to Enhance Performance". In: *IEEE Transactions on evolutionary computation* 10(1), pp. 10–28.

Lardeux, F., F. Saubion, and J-K. Hao (2006). "GASAT: a genetic local search algorithm for the satisfiability problem". In: *Evolutionary Computation* 14.2, pp. 223–253.

Laredo, J.L.J. et al. (2009). "Improving genetic algorithms performance via deterministic population shrinkage". In: *Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009*. Ed. by Franz Rothlauf. ACM, pp. 819–826.

Le Riche, R., C. Knopf-Lenoir, and R.T. Haftka (1995). "A Segregated Genetic Algorithm for Constrained Structural Optimization". In: *Proceedings of the 6th International Conference on Genetic Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 558–565.

Ledoit, O. and M. Wolf (2003). "Improved estimation of the covariance matrix of stock returns with an application to portfolio selection". In: *Journal of empirical finance* 10.5, pp. 603–621.

Lin, C-C. and Y-T. Liu (2008). "Genetic algorithms for portfolio selection problems with minimum transaction lots". In: *European Journal of Operational Research* 185.1, pp. 393–404.

Lwin, K., R. Qu, and B. L. MacCarthy (2017). "Mean-VaR portfolio optimization: A nonparametric approach". In: *European Journal of Operational Research* 260.2, pp. 751–766.

Maillard, S., T. Roncalli, and J. Teïletche (2010). "The properties of equally weighted risk contribution portfolios". In: *The Journal of Portfolio Management* 36.4, pp. 60–70.

Markowitz, H. (1959). *Portfolio selection*.

Maturana, J. et al. (2009). "Extreme compass and Dynamic Multi-Armed Bandits for Adaptive Operator Selection". In: *2009 IEEE Congress on Evolutionary Computation*, pp. 365–372.

Maturana, J., F. Lardeux, and F. Saubion (2010). "Autonomous operator management for evolutionary algorithms". In: *J. Heuristics* 16.6, pp. 881–909.

Maturana, Jorge and Frédéric Saubion (2008). "A compass to guide genetic algorithms". In: *International Conference on Parallel Problem Solving from Nature*. Springer, pp. 256–265.

Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. Artificial intelligence. Springer.

— (1995). "A Survey of Constraint Handling Techniques in Evolutionary Computation Methods". In: *Proceedings of the Fourth Annual Conference on Evolutionary Programming, EP 1995, San Diego, CA, USA, March 1-3, 1995*. Ed. by J. R. McDonnell, R. G. Reynolds, and D. B. Fogel. A Bradford Book, MIT Press. Cambridge, Massachusetts., pp. 135–155.

Michalewicz, Z. and N. Attia (1994). "Evolutionary optimization of constrained problems". In: *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pp. 98–108.

Michalewicz, Z. and M. Schoenauer (1996). "Evolutionary Algorithms for Constrained Parameter Optimization Problems". In: *Evol. Comput.* 4.1, pp. 1–32.

Michalewicz, Z., G. Nazhiyath, and M. Michalewicz (1996). "A Note on Usefulness of Geometrical Crossover for Numerical Optimization Problems". In: *Proceedings of the Fifth Annual Conference on Evolutionary Programming, EP 1996, San Diego, CA, USA, February 29 - March 2, 1996*. Ed. by Lawrence J. Fogel, Peter J. Angeline, and Thomas Bäck. MIT Press, pp. 305–312.

Montero, E. and M. C. Riff (2011). "On-the-fly calibrating strategies for evolutionary algorithms". In: *Inf. Sci.* 181.3, pp. 552–566.

Montero, E., M. C. Riff, and B. Neveu (2014). "A beginner's guide to tuning methods". In: *Appl. Soft Comput.* 17, pp. 39–51.

Moral-Escudero, R., R. Ruiz-Torrubiano, and A. Suárez (2006). "Selection of optimal investment portfolios with cardinality constraints". In: *2006 IEEE International Conference on Evolutionary Computation*. IEEE, pp. 2382–2388.

Nannen, V. and A. E. Eiben (2007). "Efficient relevance estimation and value calibration of evolutionary algorithm parameters". In: *2007 IEEE Congress on Evolutionary Computation*, pp. 103–110.

Nocedal, J. and S. Wright (2006). *Numerical optimization*. Springer Science & Business Media.

Ono, I. and S. Kobayashi (1999). "A real-coded genetic algorithm for function optimization using unimodal normal distribution". In: *Proceedings of international conference on genetic algorithms*, pp. 246–253.

Radcliffe, N.J. (1991). "Equivalence class analysis of genetic algorithms". In: *Complex systems* 5.2, pp. 183–205.

Renders, J. and H. Bersini (1994). "Hybridizing genetic algorithms with hill-climbing methods for global optimization: two possible ways". In: *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, 312–317 vol.1.

Rockafellar, R.T. and S. Uryasev (2000). "Optimization of conditional value-at-risk". In: *Journal of risk* 2, pp. 21–42.

Rockafellar, R.T., S. Uryasev, and M. Zabarankin (2006). "Generalized deviations in risk analysis". In: *Finance and Stochastics* 10.1, pp. 51–74.

Roncalli, T. (2013). *Introduction to risk parity and budgeting*. CRC Press.

Simon, D. (2013). *Evolutionary optimization algorithms. Biologically insipred and population-based approaches to computer intelligence*. Wiley.

Smit, S. K. and A. E. Eiben (2010a). "Parameter Tuning of Evolutionary Algorithms: Generalist vs. Specialist". In: *Applications of Evolutionary Computation, EvoApplicatons 2010, Istanbul, Turkey, April 7-9, 2010, Proceedings, Part I*. Vol. 6024. Lecture Notes in Computer Science. Springer, pp. 542–551.

— (2010b). "Using Entropy for Parameter Analysis of Evolutionary Algorithms". In: *Experimental Methods for the Analysis of Optimization Algorithms*. Ed. by T. Bartz-Beielstein et al. Springer, pp. 287–310.

Spears, W. (1995). "Adapting Crossover in Evolutionary Algorithms". In: *Evolutionary Programming*.

Sung, H.J. (June 2007). "Queen-bee and Mutant-bee Evolution for Genetic Algorithms". In: *Journal of Fuzzy Logic and Intelligent Systems* 17, pp. 417–422.

Syswerda, G. (1993). "Simulated Crossover in Genetic Algorithms". In: *Foundations of Genetic Algorithms*. Ed. by L. DARRELL WHITLEY. Vol. 2. Foundations of Genetic Algorithms. Elsevier, pp. 239–255.

Tessema, B. and G. G. Yen (2006). "A Self Adaptive Penalty Function Based Algorithm for Constrained Optimization". In: *2006 IEEE International Conference on Evolutionary Computation*, pp. 246–253.

Thierens, D. (2007). "Adaptive Strategies for Operator Allocation". In: *Parameter Setting in Evolutionary Algorithms*. Ed. by F. G. Lobo, C. F. Lima, and Z. Michalewicz. Vol. 54. Studies in Computational Intelligence. Springer, pp. 77–90.

Tuson, A. and P. Ross (1998). "Adapting Operator Settings in Genetic Algorithms". In: *Evolutionary Computation* 6.2, pp. 161–184.

Voigt, H.M., H. Mühlenbein, and D. Cvetkovic (1995). "Fuzzy recombination for the breeder genetic algorithm". In: *Proc. Sixth Int. Conf. on Genetic Algorithms*. Citeseer.

Wolpert, D. H. and W. G. Macready (1997). "No free lunch theorems for optimization". In: *IEEE Transactions on Evolutionary Computation* 1.1, pp. 67–82.

Wright, A. H. (1991). "Genetic Algorithms for Real Parameter Optimization". In: *Foundations of Genetic Algorithms*. Morgan Kaufmann, pp. 205–218.

Yoon, H. and B. Moon (2002). "An empirical study on the synergy of multiple crossover operators". In: *IEEE Transactions on Evolutionary Computation* 6.2, pp. 212–223.

Yuan, B. and M. Gallagher (2007). "Combining Meta-EAs and racing for difficult EA parameter tuning tasks". In: *Parameter Setting in Evolutionary Algorithms*. Springer, pp. 121–142.

Yuan, Z. et al. (2013). "An Analysis of Post-Selection in Automatic Configuration". In: *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*. GECCO '13. Amsterdam, The Netherlands: Association for Computing Machinery, 1557–1564.