Stochastic Black-Box Optimization and Benchmarking in Large Dimensions

Ouassim Ait Elhara

NNT : 2017SACLS211

Thèse de doctorat de l'Université Paris-Saclay préparée à l'Université Paris-Sud

Ecole doctorale n°580
Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat : Informatique

par

M. Ouassim AIT ELHARA

Stochastic Black-Box Optimization and Benchmarking in Large Dimensions

Thèse présentée et soutenue à Gif-sur-Yvette, France, le 28 Juillet 2017.

Composition du Jury :

M.	Marc Baboulin	Professeur Universtié Paris-Sud	(Président du jury)
M.	Cyril Fonlupt	Professeur Université du Littoral-Côte-d'Opale	(Rapporteur)
M.	Peter A.N. Bosman	Senior Researcher Centrum Wiskunde & Informatica	(Rapporteur)
M.	Tobias Glasmachers	Junior Professor Institut für Neuroinformatik, Bochum	(Examinateur)
M.	Nikolaus Hansen	Directeur de recherche Inria Saclay Ile-de-France	(Directeur de thèse)
Mme	Anne Auger	Chargée de recherche Inria Saclay Ile-de-France	(Directrice de thèse)

Résumé

Cette thèse s'intéresse à l'optimisation stochastique de problèmes de grandes dimensions qui se présentent sous forme de boîtes noires. On s'intéressera surtout aux problèmes dont les variables sont continues (contrairement aux problèmes qu'on appelle discrets ou combinatoires).

L'aspect boite noire de ces problèmes limite le choix des algorithmes à utiliser pour les résoudre. En effet, dans un problème en boîtes noires, un algorithme n'a accès, comme information, qu'à l'image de chaque solution qu'il requête. Celle image (valeur qu'on obtient en appliquant la fonction objectif) reflète la qualité d'une solution, le but étant de trouver la solution optimale; dans notre cas, celle qui minimise la fonction objectif.

On va essayer de répondre, dans cette thèse, à trois questions principales:

1. comment concevoir un mécanisme d'adaptation du pas, peu couteux et efficace, pour les stratégies d'évolution ?

2. comment construire des problèmes à dimension effective réduite et généraliser cette notion en la rendant moins restrictive ?

3. comment étendre, avec un cout raisonnable sans pour autant être triviales à résoudre, un ensemble de fonctions de test (*benchmark*) de dimensions basses/moyennes à une configuration grandes dimensions; et ceci, tout en conservant les propriétés originelles de ces fonctions ?

Après l'introduction générale du Chapitre 1, le Chapitre 2 présentera une étude de l'état de l'art autour du sujet de l'optimisation continue, et plus spécifiquement, l'optimisation continue en grandes dimensions. En suite, les trois questions présentées ci-dessus seront adressées dans les trois chapitres suivants. On synthétise et conclue dans le Chapitre 6 tout en présentant quelques perspectives.

Les contributions principales de cette thèse se présentent comme suit:

Le Mécanisme d'Adaptation du Pas : La *Median Success Rule* (MSR)

Dans le Chapitre 3, on conçoit une nouvelle méthode d'adaptation du pas pour les stratégies d'évolution en général et CMA-ES (Covariance Matrix Adaptation Evolution Strategy) en particulier.

MSR est une méthode de succès, ce qui veut dire que le pas de l'algorithme évolue suivant une mesure de succès de la population actuelle à un temps t comparée à la population de l'itération précédente $(t-1)$. Pour le cas de MSR, c'est le succès, ou non, de la médiane à l'itération t quand comparée au j_σ^{eme} individu de l'itération $t-1$ qui détermine si le pas doit être augmenté ou diminué, j_σ étant un paramètre de MSR.

Le but principal de MSR est de pouvoir être utilisée dans un contexte de grandes dimensions, et donc d'avoir un coût de calcul raisonnable. Ceci est garanti par le fait que MSR est une règle de succès dont la complexité de calcul ne dépend que de la taille de la population, et non pas de la dimension du problème. Ce qui fait de MSR une alternative à CSA, la méthode d'adaptation du pas par défaut de la *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) pour résoudre des problèmes de grandes dimensions.

MSR est une généralisation de la règle du $1/5^{\text{eme}}$ [Rechenberg, 1994]. Cette dernière étant conçue pour le cas des stratégies d'évolution à un seul parent quand MSR s'utilise au cas non-élitiste multi-parent

Un autre avantage de MSR est qu'elle fait moins de suppositions sur la nature de la fonction objectif localement et sur les autres composantes de l'algorithme auxquelles elle s'intègre (typiquement, CMA-ES), ce qui fait d'elle, comme les tests empiriques l'ont montré, une alternative viable à CSA même quand le problème à traiter n'est pas de grande dimension. Ces tests ont été conduits après avoir ajusté les paramètres de MSR, dans un premier temps sur un modèle théorique, puis, empiriquement.

Dimensions Effectives Réduites et Dimensions *epsilon*-Effectives

Le Chapitre 4 examine une classe de problèmes de grandes dimensions qu'on rencontre souvent dans le monde réel, à savoir les problèmes à dimensions effectives réduites. Dans ces problèmes, un sous-ensemble relativement petit de variables (à une transformation linéaire près) définit totalement l'image d'une solution par la fonction objectif. Ce qui veut dire que n'optimiser que ce sous-ensemble suffit pour trouver la solution optimale du problème. Cependant, ce sous-ensemble est inconnu pour l'algorithme étant dans un contexte boîte-noire.

On généralise ce concept de dimension effective réduite en permettant à une portion de l'image d'une solution de parvenir de variables qui ne sont pas dans le sous-ensemble des variables effectives. On fait en sorte que les problèmes à dimension *epsilon*-effective réduite obligent les algorithmes à ne pas se contenter d'explorer les variables effectives pour trouver la solution optimale (ou l'approcher en pratique). De plus, les deux fonctions qu'on utilise pour construire la fonction à dimension *epsilon*-effective réduite ne sont pas alignées sur le même système de coordonnées, ni aucune d'elles sur un système de coordonnées canonique; empêchant les algorithmes d'exploiter une telle propriété.

On benchmark un certain nombre d'algorithmes d'optimisation continue en grandes dimensions sur ces nouveaux problèmes. On propose aussi une variante de CMA-ES (inspirée par le travail fait dans [Wang et al., 2013]) conçue pour traiter ce genre de

problèmes. Cette variante montre des performances compétitives comparée aux autres algorithmes d'optimisation continue en grandes dimensions.

L'Extension de *COCO* aux Grandes Dimensions

Enfin, le Chapitre 5 introduit une approche générique pour construire des matrices de rotation (orthogonales) creuses (*sparse*) qui conservent certaines des propriétés des matrices de rotations complètes, notamment introduire une non-séparabilité qui ne soit pas facilement exploitable. Le fait que la matrice soit creuse permet un coût de calcul de la fonction objectif qui reste linéaire en la dimension de l'espace de recherche, une propriété importante quand il s'agit de problèmes de grandes dimensions permettant le benchmarking d'algorithmes dans des temps raisonnables.

Les matrices de rotation proposées sont ensuite utilisées pour construire une version grandes dimensions de la plateforme de comparaison d'algorithmes d'optimisation continue *COmparing Continuous Optimisers* (COCO). Les fonctions d'origine ont subi quelques modification afin de les rendre compatibles avec un scénario grandes dimensions.

En utilisant cette transformation, on a défini un nouveau benchmark sur lequel un nombre d'algorithmes d'optimisation continue en grandes dimensions (des variantes de CMA-ES) ont été testés. Les performances observées suggèrent que les algorithmes n'arrivent pas à exploiter la forme particulière de la matrice de rotation utilisée. Le fait que ce benchmark fasse partie de la plateforme COCO a permis d'exploiter directement les fonctionnalités de cette dernière, notamment celles concernant le traitement des données générées et leurs visualisation; en plus d'offrir une interface multi-plateformes et multi-langages.

Acknowledgments

I would like to thank my advisors Anne AUGER and Nikolaus HANSEN for giving me this chance of doing a PhD thesis under their supervision. This thesis would simply not have been possible without their continuous support, supervision and high level advise. I also would like to thank Cyril FONLUPT and and Peter A.N. BOSMAN for reviewing my thesis in great detail and Marc BABOULIN and Tobias GLASMACHERS for accepting to be part of my jury and giving me invaluable feedback on my work. This thesis was prepared in the TAO team that I deeply thank for provinding me with an ideal working environement and surrounding me with amazing people. I am grateful to my family and to my friends who were and still are always there for me.

Finally, and most importantly, I thank my parents for their everlasting support and faith, thank you Yemma & Vava.

Contents

Chapter 1

General Introduction

Large-scale optimization is one topic that is getting an increasing amount of attention these last few years. With the increasingly powerful machines, CPU and memory wise, both scientists and engineers conceive and model problems of larger sizes and higher complexities. It is now fairly common to solve a classification problem using deep neural networks with up to millions of weights to be optimized.

When dealing with large and complex problems, we generally do not have much knowledge of the problem at hand, or when we do, incorporating this knowledge in the problem-solving process can end up being a tedious task, too complex to implement. This is where black-box optimization algorithms become handy. A block-box algorithm only needs, as information, the fitness or quality of each solution it queries, which makes it applicable on a large array of problems.

A number of questions are raised when dealing with continuous optimization problems and large-scale problems in particular. We are first interested in the difficulties these problems generally pose, in order to better design algorithms to solve them. Then, in a large scale setting, one must take into account the additional constraints brought by the large-scale property of the problem. The amount of data that can be stored for each decision variable becomes more limited (generally constant or logarithmic in the dimension of the problem). For example, it is not feasible to keep a full matrix of all parameter interactions because of the quadratic cost it would produce. Memory constraints are not the only ones that become increasingly important as the dimension of the problem increases. The time complexity of the algorithm is also a critical matter, even when the data is stored/used efficiently or a limited amount of it is needed. An algorithm that scales badly (a high scaling) in the problem dimension with regards to the number of operations it conducts per iteration or the expected number of iterations it needs to deliver good enough solutions become highly unpractical.

Because of the generally high computational costs that come with large-scale problems, more so on real world problems, the use of benchmarks is a common practice in algorithm design, algorithm tuning or algorithm choice/evaluation. The question is then the forms in which these real-world problems come. Answering this question is generally hard due to the variety of these problems and the tediousness of describing each of them. Instead, one can investigate the commonly encountered difficulties when solving contin-

uous optimization problems. Once the difficulties identified, one can construct relevant benchmark functions that reproduce these difficulties and allow to assess the ability of algorithms to solve them. The impact of the presence of a difficulty on the performance varies from one algorithm to another and might also depend on which other difficulties or properties are present. We generally say that an algorithm succeeds in addressing a difficulty when it becomes insensitive to its presence or, at least, when the impact of this difficulty on its performance is reasonable (for example, does not prevent the algorithm from solving the problem or makes it able to solve it only with an exceedingly high cost).

In the case of large-scale benchmarking, it would be natural and convenient to build on the work that was already done on smaller dimensions, and be able to extend it to larger ones. When doing so, we must take into account the added constraints that come with a large-scale scenario such as the ones described above. We need to be able to reproduce, as much as possible, the effects and properties of any part of the benchmark that needs to be replaced or adapted for large-scales (in our case, orthogonal transformation matrices). This is done in order for the new benchmarks to remain relevant. Only simplifying or reducing the size/cost of these tools might introduce new properties to the problem that algorithms can exploit or remove other relevant properties or reduce their effects. For example, one must be cautious when replacing full matrices with sparse ones, using block-diagonal matrices introduces a block-diagonal property which is exploitable; a diagonal matrix generally reduces the complexity of the problem by a high amount and a highly random choice (high variance) of the non-zero elements of a sparse matrix can reduce its rank and thus impact the effective dimension of the problem and/or result in a lack of control on the difficulty of the resulting problem....

It is common to classify the problems, and thus the benchmarks, according to the difficulties they present and properties they possess. It is true that in a black-box scenario, such information (difficulties, properties...) is supposed unknown to the algorithm. However, in a benchmarking setting, this classification becomes important and allows to better identify and understand the shortcomings of a method, and thus make it easier to improve it or alternatively to switch to a more efficient one (one needs to make sure the algorithms are exploiting this knowledge when solving the problems). Thus the importance of identifying the difficulties and properties of the problems of a benchmarking suite and, in our case, preserving them.

One other question that rises particularly when dealing with large-scale problems is the relevance of the decision variables. In a small dimension problem, it is common to have all variable contribute a fair amount to the fitness value of the solution or, at least, to be in a scenario where all variables need to be optimized in order to reach high quality solutions. This is however not always the case in large-scales; with the increasing number of variables, some of them become redundant or groups of variables can be replaced with smaller groups since it is then increasingly difficult to find a minimalistic representation of a problem. This minimalistic representation is sometimes not even desired, for example when it makes the resulting problem more complex and the trade-off with the increase in number of variables is not favorable, or larger numbers of variables and different representations of the same features within a same problem allow a better exploration.

This encourages the design of both algorithms and benchmarks for this class of problems, especially if such algorithms can take advantage of the low *effective* dimensionality of the problems, or, in a complete black-box scenario, cost little to test for it (low effective dimension) and optimize assuming a small effective dimension.

In this thesis, we address three questions that generally arise in stochastic continuous black-box optimization and benchmarking in high dimensions:

1. How to design cheap and yet efficient step-size adaptation mechanism for evolution strategies?
2. How to construct and generalize low effective dimension problems?
3. How to extend a low/medium dimension benchmark to large-dimensions while remaining computationally reasonable, non-trivial and preserve the properties of the original problem?

We start by presenting the general context of this thesis and a study of the state of the art in Chapter 2. We then address these questions in three contribution-focused chapters that we will briefly describe below. We finally conclude in Chapter 6 by summarizing the contributions and answers this thesis brought and the open questions that it opens for future work.

1.1 Main Contributions

1.1.1 Cheap Step-Size Adaptation: The Median Success Rule

In Chapter 3, we develop a new success-based step-size adaptation mechanism for the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) and evolution strategies in general that has a low computational cost and thus is applicable in a large-scale setting. The use of a success-based rule allows to have a complexity that depends on the number of offspring involved, not the dimension of the problem. The proposed strategy is meant as a generalization of the simpler one-fifth success rule [Rechenberg, 1994], designed for single parent evolution strategies, to the case of multi-parent non-elitist evolution strategies. It also provides a strategy with less assumptions than the Cumulative Step-size Adaptation (CSA) that it is meant to replace on the other components of the algorithm (sampling the solutions).

1.1.2 Relevance of Variables/Dimensions: Low Effective Dimensions and Their Generalization

In Chapter 4, we investigate a class of large-scale problems that often appear in real world applications: the class of low effective dimension problems, where a few variables/dimensions or linear combinations of them define the problem. This is translated by the fitness function being flat (non-changing) in a large proportion of directions/dimensions. We also generalize this concept to one where a small proportion of

the fitness is *still* defined by all the variables while the rest is similar to the low effective dimension part. This results in inherently ill conditioned problems, with a high condition number which is a class of problems that is present in most of the relevant benchmarks. We define the problems such that we encourage optimizing the effective space while making sure the non-effective space is not neglected; otherwise, this becomes a low effective dimension problem. However, we only ask for the fitness in the non-effective space to not degrade in comparison to the initial solution, and make sure that the coordinate systems on which the two functions are defined are not aligned to prevent exploitability. We look into the performance of state of the art large-scale continuous optimization algorithms on these problems and also propose a variant of CMA-ES, inspired by [Wang et al., 2013], specifically designed to solve these problems in an efficient manner by taking into account their low effective dimensionality.

1.1.3 Extension of a Benchmark to Large-Scales: from COCO to Large-Scale COCO

Chapter 5 proposes a generic way of constructing sparse orthogonal (rotation) matrices for large-scale optimization problems while retaining many of the properties a full rotation matrix provides, most importantly, introduce reasonable amounts of non-separability. The proposed transformation is then applied to the widely used COmparing Continuous Optimisers (COCO) bencmarking platform in order to extend its single-objective noiseless test-bed [Hansen et al., 2009], while also introducing a number of tweaks on the functions in order to make them more large-scale compatible. This allows to define a full test-bed of large-scale problems on which we benchmark a number of large-scale CMA-ES variants. The proposed suite offers an alternative to the only other prominent large-scale test-bed, the one used in the CEC competition and special sessions on large-scale optimization [Tang et al., 2007, Tang et al., 2009, Li et al., 2013], and addresses some of its shortcomings such as the performance assessment. In addition, extending the COCO benchmarks allows to use the already well-developed experiment-running and data post-processing tools proposed by the platform.

Chapter 2

Background Study

In this chapter, we start by presenting the general context of this thesis. We will also introduce some notions and notations that will be encountered all along this thesis in addition to defining the conventions that we will be using.

We describe the Covariance Matrix Evolution Strategy (CMA-ES) since it is highly relevant in the context of this thesis and will often be used and referenced. We review the state of the art on large-scale continuous optimization and the different approaches used to tackle high dimensionalities.

2.1 Continuous Black-Box Optimization

In this thesis, we focus on the topic of black-box continuous optimization. That is problems that consist in optimizing (finding the optimum/optima of) a function f defined in a search space which is a subspace of $\mathbb{R}^d$, d being the problem dimension and $\mathbb{R}$ the set of real numbers. We are interested in finding the solution(s) $\mathbf{x}_{\text{opt}}$ that minimizes the function f that we call a *fitness function*:

$$\mathbf{x}_{\text{opt}} = \arg\min_{\mathbf{x} \in S} f(\mathbf{x}) \ , \tag{2.1}$$

where $S \subseteq \mathbb{R}^d$ is the search space, we will note the optimal fitness f_{opt}, $f_{\text{opt}} = f(\mathbf{x}_{\text{opt}})$. When the search-space is not explicitly specified, we assume it to be $\mathbb{R}^d$. We will use bold symbols such as $\mathbf{x}$ and $\mathbf{C}$ to denote vectors and matrices, regular (non-bold) symbols will generally represent scalar values, so $\mathbf{x}$ and $\mathbf{x}_t$ are vectors while x_i is a scalar (here a coordinate of $\mathbf{x}$).

We restrict ourselves to minimization problems without loss of generality since maximizing a function f is the same as minimizing its negative $-f$:

$$\arg\max_{\mathbf{x} \in S \subseteq \mathbb{R}^d} (f(\mathbf{x})) = \arg\min_{\mathbf{x} \in S \subseteq \mathbb{R}^d} (-f(\mathbf{x})) \ . \tag{2.2}$$

In a black-box scenario (also called derivative-free scenario in the context of continuous optimization), an algorithm is provided with no information other than the fitness of

solutions that it chooses to query. In many cases, the cost of an optimization process is expressed in number of function queries or evaluations that were carried. In fact, the call to the objective function is the main, if not the only, interface between the algorithm and the problem. This allows to have a measure for the cost of an optimization that is independent of the optimization algorithm and, contrarily to CPU-time based measures, independent of the machine and implementation of the algorithm. Optimizing the CPU time, memory usage and parallelizing an already established algorithm are independent problems that we do not focus on in this thesis. However, in real-world applications, the actual overall time needed by an algorithm to solve a given problem is the most prominent performance measure. Thus, providing, on top of the cost in number of function evaluations, an estimation of the CPU time needed per function evaluation, in addition to details such as the programming language and the machine on which the algorithm is run, gives a better appreciation of the *speed* of an algorithm and allows a more comprehensive assessment of its performance. Here, what we mean by CPU time per function evaluation is the average CPU time allocated to the internal effort of the algorithm each time a function evaluations is conducted. Algorithms are, thus, confronted with the double, contradicting, objectives of finding the best solutions using the least function evaluations.

In this thesis, we consider single-objective noiseless problems, and especially focus on large-scale problems. This means that in (2.1), the function $f : S \subseteq \mathbb{R}^d \to \mathbb{R}$ is deterministic, so querying a same solution $\mathbf{x}$ always returns the same fitness (the value $f(\mathbf{x})$ is deterministic). The large-scale component is translated into the fact that d is relatively high, or high enough that the scaling of the computational cost (internal complexity) of the algorithm in the problem dimension becomes a critical factor. Values of d that are at least in the hundreds seem a reasonable choice to be considered large-scale problems in practice. However, we take into account the scaling of the algorithms to even larger magnitudes of d and thus consider efficient and usable in practice only algorithms whose computational complexities (both in CPU time and in memory usage) are linear or at most in the order of $\log(d) \times d$. Quadratic scalings and larger are generally not acceptable in large-scale settings.

2.1.1 The Challenges of Large-Scale Continuous Optimization

A number of difficulties arise when dealing with continuous black-box optimization problems in general and large-scale cases in particular.

2.1.1.1 Dimension of the Problem

The first difficulty has to do with the number of parameters, also known as the dimension of the problem, d in (2.1). As the dimension increases, so does the size of the search-space and thus also the effort needed to explore it and find optimal solutions. The exponential increase of the size of the search space in d is often referred to as *the curse of dimensionality* [Richard, 1957].

This exponential cost reduces the usefulness on larger dimensions of basic search methods such as random search which generates solutions randomly without adapting the distribution or exhaustive/brute-force search, more commonly used in discrete optimization with finite search-space and generally adapted to continuous optimization in the form of grid-searches. Thus, more sophisticated search methods are required to solve black-box problems on higher dimensions (brute force methods are already not reasonable choices in continuous optimization for dimensions as small as 5); these methods can, however, include a sub-component (generally local/in a neighborhood) which relies on these simpler search strategies.

2.1.1.2 Non-Separability

We say that a problem is non-separable when its variables interact with each other and thus one can not solve the problem by simply solving the sub-problems defined by each of its variables (can not *separate* the problem into d independent sub-problems and expect to solve it). The term *epistasis* is sometimes used as an analogy to the interactions between genes; when the effect of a gene differs depending on the identity of the other genes that are present.

On a separable problem, the behavior of the fitness function when varying a variable x_i only depends on the values of this same variable x_i, not the other variables. Thus, the optimal solution of the problem is the aggregation of the optimal solutions when considering d sub-problems each defined on a single variable. When a function is separable, we have:

$$\arg\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}) = \left(\arg\min_{x_1\in\mathbb{R}} f(x_1, \tilde{x}_2^1, \ldots, \tilde{x}_d^1), \ldots, \arg\min_{x_d\in\mathbb{R}} f(\tilde{x}_1^d, \ldots, \tilde{x}_{d-1}^d, x_d)\right)^\mathsf{T} , \tag{2.3}$$

where $(\tilde{x}_i^j)_{i,j\in\{1,\ldots,d\},i\neq j}$ constant, arbitrary, values within the search-space and $^\mathsf{T}$ will designate the transpose of a vector or of a matrix.

We see in (2.3) that separable problems can be efficiently solved via d line-searches on each of the sub-problems (a line search is a search method where a single direction is followed with potentially different step-lengths). With the cost of performing these one-dimensional line-searches independent of the dimension (constant in d since the dimension of the problem at hand is 1), the overall cost of solving (2.1) is generally linear in d (in the order of $c \times d$ where c is independent of d); which is considered a low cost in the context of continuous optimization and the problem is considered *easy* as a result of that.

Non-separable problems are detrimental to the performance of coordinate-system aligned line-search methods. Proper and improper rotation matrices (orthogonal matrices) are commonly used to introduce arbitrary linear dependencies between the variables and coordinate system-independence. [Salomon, 1996] showed that some Genetic Algorithms (GA), the Breeder Genetic Algorithm (BGA) [Schlierkamp-Voosen and Mühlenbein, 1993] is used as an example, with small mutation rates see a drop in performance when a rotation matrix is applied to the search-spaces; and suggests to use invariant methods such as Evolution Strategies (ES see Section 2.1.3) to tackle such

problems. This deterioration in performance is also noted on the widely used Particle Swarm Optimization algorithm in [Hansen et al., 2011].

2.1.1.3 Multi-Modality

A multi-modal problem is a problem with one or more local optima in addition to the global optimum. Even though the gradient information is unavailable in a black-box setting, seeking the optimum by following the direction of improvement of the function remains an intuitive and widely used method, especially in local search algorithms. This makes multi-modality a relevant property to take into consideration.

When a problem is multi-modal, finding a solution which is locally optimal does not guarantee its global optimality, and thus solving the problem. This is why restart strategies, in which it is important to define the stopping criteria well in order to allow for early detection of stagnation in local optima, are often used to tackle multi-modality. Well set stopping criteria prevent the algorithms, among other things, from wasting its budget in non-beneficial exploitation around local optima and instead allow it to use the budget in a more promising way. Restarts allow the algorithm to increase the chances of being caught in the basin of attraction of the global optimum (area of the search-space that leads the algorithm to a given *attractor*, in our case an optimum) or increase the chances of finding the global optimum by adapting the parameters of the algorithm (larger number of solutions sampled in each iteration or a more global search with larger steps). Some commonly used restart methods in Evolution Strategies (ES) will be presented in Section 2.1.4.

Other approaches include techniques to escape from the local optima such as epsilon-greediness and starting with a global search (exploration phase) then doing a more local search (exploitation). An epsilon-greedy (ϵ-greedy) algorithm chooses, with a probability $\epsilon \in [0, 1]$, an action which is considered non-greedy; that is an action which does not aim at immediately improving the fitness (this action is generally chosen to be a random action). The simulated annealing algorithm, first formalized in [Metropolis et al., 1953], relies on the use of a *temperature* parameter that dictates the behavior of the search. The higher the temperature, the more the particles move and thus the larger the steps and more global the search. As the iterations go by, the temperature is reduced and the search becomes more focused and local until it converges, ideally, to the global optimum.

2.1.1.4 Ill-Conditioning

The conditioning of a problem is, roughly speaking, the difference in sensitivity of the fitness function when varying different variables. We generally consider directions instead of variables which means that the bases on which the conditioning is computed does not need to be canonical and coordinate-system aligned. The conditioning of a convex-quadratic function is well defined as the condition number of its Hessian matrix; that is the ratio between the largest and smallest eigenvalues of the Hessian matrix.

Ill-conditioned problems are most problematic to algorithms with isotropic sampling or that assume an isotropic landscape (do not learn or adapt the relative importance

of each direction and consider all dimensions to be equal). It is particularly tricky when the conditioning of the problem changes depending on the position in the search-space. Another particularly difficult case is when the ill-conditioned problem is also non-separable (see Section 2.1.1.2) since in such a case, the algorithm can not simply check and learn the sensitivity of each variable apart. Instead, and for such an approach, it needs to first transform the search-space into one where the function is separable. The rotated Ellipsoid function (see f_{10} in Table 5.3 for a variant of this function) is an example of an ill-conditioned non-separable problem that has proven to be particularly difficult for algorithms that lack invariance (robustness or non-sensitiveness) to certain search-space transformations (in this case, orthogonal transformations of the search-space).

In a generic ill-conditioned black-box setting, one can assume that a minimum of d parameters need to be learned in order to account for the differences of importance between the d variables or directions and thus to be able to efficiently solve the problem. This is an example of how the dimension of the problems affects a number of its aspects and thus can increase its difficulty in more than the one direct way of resulting in larger search-spaces.

2.1.1.5 Other Difficulties

Other difficulties that might be encountered when dealing with continuous black-box problems are the absence of a global structure of the landscape and lack of patterns that can be taken advantage of and might dispense an algorithm from learning each local area independently from scratch. Skewness, which is represented by non-symmetric high slops in the fitness landscape, is also another property that generally adds to the difficulty of the problem.

Particular classes of continuous black-box optimization problems, other than large-scale problems, come with their own difficulties.

Noisy Optimization

The noise, on noisy functions, generally requires the use of algorithms that are substantially different from those used on noiseless cases. Often, the difference with the noiseless case is more pronounced as the algorithm approaches the optimal solution in which case correctly guessing which of a set of solutions is the actual best becomes a tedious task. Several evaluations are generally needed in order to estimate the *real* fitness of a solution, and algorithm resort to assuming a model for the noise that needs to be learned/adjusted.

Another type of *deterministic* noise can come from a highly rugged search-space or one that lacks continuity where solutions that are close in the search-space find themselves distant in the objective space (when comparing their fitnesses). One quality that most ES and evolutionary algorithms in general share is robustness when dealing with these kinds of functions and capacity to solve them.

Expensive Optimization

In expensive optimization, the function evaluations are highly costly and the algorithm

can afford to perform only a few of them (for example, this is often the case when a single function evaluation requires a full simulation). These problems are generally solved using model-based approaches (some of which, CMA-ES based, will be presented in Section 2.1.4) where the algorithm assumes, locally, that the fitness function follows a certain model (quadratic, Gaussian,...) and adjusts the parameters of this model by taking into account the few solutions that it evaluates on the real (expensive) fitness function. This is done in the hope of making the model as resemblant as possible to the actual fitness function without making the model too complex, too complicated or too costly.

Constrained Optimization

Constrained optimization is another class where solutions, in addition to having a fitness, are either feasible or not. The goal is then to find the best *feasible* solution. The simplest case of constrained optimization is when the search-space is not $\mathbb{R}^d$ but a strictly smaller sub-set S of it. In this case, solutions outside of S are not accepted and thus are considered unfeasible. In other cases, the most common when describing constrained problems, the constraints are defined using expressions whose truth values define the feasibility of a solution. Constraints that simply limit the interval in which each variable takes its values (for example $[-5, 5]^d$) are generally called box constraints. There are two main approaches to solving constrained-problems: (i) *resampling*, where the algorithm discards unfeasible solutions and tries to resample feasible ones and (ii) *penalization*, which requires the constraint to return more than just a binary value of whether the solution is feasible or not, then a penalty measure is added to the unfeasible solutions to favor feasible ones but still take them into account.

2.1.2 Evolutionary Algorithms for Continuous Black-Box Optimization

When dealing with black-box problems, algorithms do not have a direct access to any information other than the fitness of each solution they query. An overview of the methods that are used to solve these problems, also called derivative-free optimization, can be found in [Rios and Sahinidis, 2013]. Among these methods, population based algorithms perform generally well when dealing with such problems and the difficulties they bring along such as the ones presented in Section 2.1.1. They do not require additional information such as the gradients (that might need to be estimated) and generally do not suppose particular properties of the problem (continuity, differentiability...) which makes them particularly suited for black-box optimization. These algorithms are commonly known as *Evolutionary Algorithms* (EA).

The core principle in evolutionary algorithms is inspired from the process of natural evolution based on mutation and natural selection. A population of individuals is evolved by means of mutation and cross-over and a selection pressure is applied through the fitness functions: the better the fitness of an individual the more likely it is to survive in the population and propagate its genome. In principle, the fitness of the solutions

in the population improves over time, which translates into the algorithm finding better solutions and, ideally, ending up with a population where one or many of the solution are of the desired fitness.

Notable population based methods that are often used to solve continuous optimization problems include Particle Swap Optimization (PSO) [James and Russell, 1995], Differential Evolution (DE) [Storn and Price, 1997] (first appeared in [Storn and Price, 1995], Estimation of Distribution Algorithms (EDA) [Larranaga and Lozano, 2002] and Evolutionary Strategies (ES) that will be the focus of the next section and that are the most relevant to this thesis.

2.1.2.1 Differential Evolution

In the basic version of Differential Evolution introduced in [Storn and Price, 1997], each individual $\mathbf{x}_t^i$ and at each iteration t takes part in a crossover operation with a mutated individual $\hat{\mathbf{x}}_t^i$ in order to construct the candidate individual $\tilde{\mathbf{x}}_t^i$ that will potentially replace it. The mutated individual $\hat{\mathbf{x}}_t^i$ is a linear combination of three distinct individuals $\mathbf{x}_t^{p,i}, \mathbf{x}_t^{q,i}, \mathbf{x}_t^{r,i}$, which are also different from $\mathbf{x}_t^i$, and that are chosen randomly (so a population size of at least four is required):

$$\hat{\mathbf{x}}_t^i = \mathbf{x}_t^{p,i} + F(\mathbf{x}_t^{q,i} - \mathbf{x}_t^{r,i}) \ , \tag{2.4}$$

where $F \in [0, 2]$ is the *differential weight* (adds the weighted difference between $\mathbf{x}_t^{q,i}$ and $\mathbf{x}_t^{r,i}$ to $\mathbf{x}_t^{p,i}$). The candidate individual $\tilde{\mathbf{x}}_t^i$ is defined coordinate-wise:

$$[\tilde{\mathbf{x}}_t^i]_j = \begin{cases} [\hat{\mathbf{x}}_t^i]_j & \text{if } j = k \text{ or } \text{rand}() \leq \text{CR} \\ [\mathbf{x}_t^i]_j & \text{otherwise} \end{cases} \ , \tag{2.5}$$

where $[\mathbf{x}]_j$ the j^{th} coordinate of a vector $\mathbf{x}$, k is a randomly chosen coordinate for each individual at each iteration that insures that at least one coordinate comes from the mutated individual $\hat{\mathbf{x}}_t^i$, rand() is a realization of a random uniform distribution in $[0, 1]$ and $\text{CR} \in [0, 1]$ the crossover rate that determines the expected proportion of coordinates that will be taken from the mutated individual. The candidate individual $\tilde{\mathbf{x}}_t^i$ replaces $\mathbf{x}_t^i$ when it improves on its fitness (or has equal fitness for exploration-promoting purposes):

$$\mathbf{x}_{t+1}^i = \begin{cases} \tilde{\mathbf{x}}_t^i & \text{if } f(\tilde{\mathbf{x}}_t^i) \leq f(\mathbf{x}_t^i) \\ \mathbf{x}_t^i & \text{otherwise} \end{cases} \ . \tag{2.6}$$

A number of variants of DE were developed, among which: Differential Evolution with Neighborhood Search NSDE [Liu and Li, 2010], Adaptive Differential Evolution with Optional External Archive [Zhang and Sanderson, 2009] and a Self adaptive Differential Evolution [Qin et al., 2009].

2.1.2.2 Particle Swarm Optimization

PSO is one of the most widely used EA in continuous optimization. In the basic version, no selection mechanism is used so the algorithm consists in evolving a population of

particles. Each particle $\mathbf{x}_t^i$, at time t, is assigned a speed or velocity $\mathbf{v}_t^i$ that will serve to help define its next position:

$$\mathbf{x}_{t+1}^i = \mathbf{x}_t^i + \mathbf{v}_t^i \ . \tag{2.7}$$

The velocity of each particle is updated by taking into account its best seen position $\mathbf{p}_{\text{best}}^i$ (fitness-wise) and the best seen position of the swarm $\mathbf{g}_{\text{best}}$. The idea is to make the particle move, at time $t+1$, towards these two solutions which have a high fitness. This is done by updating the particle's velocity using a linear combination of the the directions that point to these two solutions:

$$\mathbf{v}_{t+1}^i = \mathbf{v}_t^i + \phi_1 \mathbf{R}_1(\mathbf{p}_{\text{best}}^i - \mathbf{x}_t^i) + \phi_2 \mathbf{R}_2(\mathbf{g}_{\text{best}} - \mathbf{x}_t^i) \ , \tag{2.8}$$

where $\phi_1, \phi_2 \in \mathbb{R}$ are parameters of the algorithm that determine the relative attraction powers of $\mathbf{p}_{\text{best}}$ and $\mathbf{g}_{\text{best}}$ and $\mathbf{R}_1, \mathbf{R}_2 \in \mathbb{R}^{d \times d}$ are diagonal matrices (all non-diagonal elements are zeros) whose diagonal elements are generated randomly at each iteration and for each individual. It was empirically shown in [Hansen et al., 2011] that PSO suffers from rotated search spaces (the functions become, in most cases, not aligned with the coordinate-system). This makes PSO not rotation invariant. The performance of a rotationally invariant algorithm on a function would, roughly speaking, be the same regardless of the rotation that is applied to the search space (an identity transformation is obtained when the rotation matrix is an identity matrix). The impact on the performance of PSO observed in the paper is positively correlated with the condition number of the function. Since the matrices $\mathbf{R}_1$ and $\mathbf{R}_2$ are diagonal and not adapted, their random entries are independent only in an axis-parallel coordinate system.

One of the first improvements on PSO consists in decreasing its tendency to fall into local optima by defining $\mathbf{g}_{\text{best}}^i$ for each particle as the best seen position in a neighborhood or a sub-swarm. Thus one particle finding a local optimum, which generally dominates most of the solutions in the search space, would only affect the behavior of a portion of the population instead of it (the local optimum) becoming an attractor of all individuals.

The variants of PSO that are found in the literature act on its different aspects. [Shi and Eberhart, 1998] introduced an *inertia weight*, w, to the update of the speed in (2.8) in the form of

$$\mathbf{v}_{t+1}^i = w\mathbf{v}_t^i + \phi_1 \mathbf{R}_1(\mathbf{p}_{\text{best}}^i - \mathbf{x}_t^i) + \phi_2 \mathbf{R}_2(\mathbf{g}_{\text{best}} - \mathbf{x}_t^i) \ , \tag{2.9}$$

that controls the speed at which the velocities change, both in direction and module, thus allowing a better control over the local versus global search behavior of the algorithm. In [Montes de Oca and Stützle, 2008], an incremental population size is considered while the neighborhoods that define the $\mathbf{g}_{\text{best}}$ are dynamically updated in several variants of the algorithm [Hu and Eberhart, 2002, Akat and Gazi, 2008, Liu et al., 2009].

2.1.2.3 Estimation of Distribution Algorithms

The purpose of Estimation of Distribution Algorithms is to learn a distribution that maximizes the likelihood of sampling *optimal* solutions. The basic steps in the core of an EDA are as follows:

1. Sample a population of solutions using the current distribution.
2. Evaluate the newly sampled solutions on the fitness function.
3. Select a portion of the population, generally the best fit solutions.
4. Update the current distribution using the information that was gathered from the selected solutions (their distribution) and prepare for the next iteration.
5. Repeat from step 1. until a stopping criterion is met.

The distribution of the solutions (the sampling distribution) is adapted explicitly using only the selected, most fit, individuals. Thus the likelihood of generating similar, and thus generally well-fit, offspring is increased. Ideally, the algorithm converges to a distribution of selected solutions close enough to the optimal solution.

Contrarily to many EAs, EDAs do not use mutation and cross-over in order to evolve the population. Instead, the state of the algorithm, at a given point in time, is determined by the values of the parameters of its parameterized sampling distribution. So, a population is generated at each iteration but it is the distribution that is evolved. In a way, many Evolution Strategies (see Section 2.1.3), particularly those who adapt a fully parameterized multi-variate normal distribution such as CMA-ES (see Section 2.1.4), can be considered as estimation of distribution algorithms. In fact, in CMA-ES, the sampling distribution is adapted given, among other things, the information about the selected offspring. More details about the inner working of CMA-ES will be given in Section 2.1.4.

Multi-variate normal distributions are widely used as distribution models in EDAs. In order to increase the exploratory capabilities of the algorithm and reduce the impact on the performance of falling into local optimal, generally, multiple Multi-variate normal distributions are used and evolved in parallel.

One of the earlier implementations of EDAs can be traced back to [Baluja and Caruana, 1995] on binary problems and in [Mühlenbein and Paass, 1996] and [Mühlenbein et al., 1996] where the Breeder Genetic Algorithm (BGA) [Schlierkamp-Voosen and Mühlenbein, 1993] is investigated. In the latter, two variants are proposed, the Uniform Distribution Breeder Genetic Algorithm and the Univariate Marginal Distribution Breeder Genetic Algorithm. In both variants, the solutions (offspring) of an iteration are sampled following the distribution of the selected points of the preceeding iteration instead of being the result of crossover (as it is done in the original BGA). The paper ends up suggesting the use of the distributi

[Larranaga and Lozano, 2002] lays the basis for using EDAs in discrete and continuous optimization.

2.1.3 Evolution Strategies

Evolution Strategies (ES) are a sub-category of evolutionary algorithms that rely on the use of multi-variate normal distributions to sample the population and that are

generally specialized in continuous search-space optimization. It is important for an evolution strategy to include an adaptation mechanism that allows it to adapt some of the parameters of its (multi-variate normal) distribution online (within the optimization process) since otherwise, it would be a simple variant of a random search algorithm.

[Igel et al., 2007]

The beginnings of evolution strategies can be traced back to [Rechenberg, 1973] and [Schwefel, 1965]. Even though in the earlier implementations of multi-parent evolution strategies (also called multimembered ES) such as [Schwefel, 1977], each individual of the population was associated to a normal distribution, in this thesis, we are interested in the study of the relatively recent variants of evolution strategies in which all the offspring at a given time-step, that we call *iteration*, are generated from the same distribution. Thus, the sampling distribution at iteration t is defined by its parameters, the mean solution $\mathbf{x}_t$ and the covariance matrix $\mathbf{C}_t$, and noted $\mathbf{N}(\mathbf{x}_t, \mathbf{C}_t)$. More recent examples of evolution strategies that sample their populations from multiple normal distributions and adapt the parameters of these distributions include the Multi-Objective Covariance Matrix Adaptation Evolution Strategy (MO-CMA-ES) [Igel et al., 2007], a variant of CMA-ES designed for multi-objective optimization. A good overview of the origins and development of evolution strategies can be found in [Beyer and Schwefel, 2002].

In most evolution strategy studies, the covariance matrix is decomposed into a positive scaling factor σ_t that is called the *step-size* and a symmetric positive-definite matrix that is referred to, by abuse of language, as the covariance matrix $\mathbf{C}_t$. Roughly speaking, the step-size determines the spread of the sampled solutions, that we call offspring, around the mean $\mathbf{x}_t$ while the covariance matrix shapes this spread. For example, a highly conditioned covariance matrix results in the solutions being more spread (far from the mean of the distribution) in some directions than in others while the larger the step-size, the further away the generated solutions are, in expectation, from the mean solution.

We generally distinguish between elitist approaches, also called plus (+) strategies, where solutions from the previous iteration are added to the selection pool of the current iteration (and thus compete with the current offspring for selection) and non-elitist strategies, also called comma (,) strategies, where the selection is restricted to the set of solutions that were generated in the current iteration. We note the number of parents or number of selected solutions μ and the population size λ. We use the notation $(\mu/\mu_w, \lambda)$-ES to designate weighted-recombination based strategies, that is strategies where the selected μ parents are recombined in order to form a new mean solution $\mathbf{x}_{t+1}$; an alternative being to simply choose the best solution as the mean of the next iteration. The weighted recombination process in evolution strategies is the preferred cross-over mechanism, especially since these strategies often deal with continuous domain problems.

In a $(\mu/\mu_w, \lambda)$-ES, at each iteration t, λ offspring $\mathbf{x}_t^1, \ldots, \mathbf{x}_t^\lambda$ are sampled from the current distribution:

$$\mathbf{x}_t^i \sim \mathbf{N}(\mathbf{x}_t, \sigma_t^2 \mathbf{C}_t) \ , \tag{2.10}$$

where $\mathbf{x}_t$ is the current mean solution, σ_t is the current step-size and $\mathbf{C}_t$ is the covariance matrix.

Then, the offspring are ordered, after evaluation, depending on their fitness values

such that:

$$f(\mathbf{x}_t^{1:\lambda}) \leq \cdots \leq f(\mathbf{x}_t^{\lambda:\lambda}) \ , \tag{2.11}$$

where, thus, $\mathbf{x}_t^{i:\lambda}$ designates the i^{th} best individual of iteration t (among the λ individuals that were sampled).

Finally, the mean solution is updated to be the weighted recombination of the best (selected) μ individuals:

$$\mathbf{x}_{t+1} = \sum_{i=1}^{\mu} w_i \mathbf{x}_t^{i:\lambda} \ , \tag{2.12}$$

where $w_1 \geq \cdots \geq w_\mu$ are the recombination weights. Evolution strategies often consider all weights to be positive ($w_\mu \geq 0$), and thus use only the information gathered from the successful (selected) offspring. An alternative approach, called *active update* (and the resulting algorithm active CMA-ES) was proposed in [Jastrebski and Arnold, 2006] where the unsuccessful solutions are given negative weights after [Rudolph, 1997] showed that negative weights can improve the performance of recombination based strategies on some functions. The active update was later implemented into $(\mu/\mu_w, \lambda)$-ES in [Hansen and Ros, 2010]. In addition, the optimal weights derived for the sphere function in [Arnold, 2005] include positive weights for half the population and negative weights for the other half ($\mu = \lambda$).

The other parameters of the distribution (other than the mean solution $\mathbf{x}_t$), the step-size σ_t and covariance matrix $\mathbf{C}_t$, can also be updated taking into account the information gathered thus far. Naturally, the adaptation of σ_t is called *step-size adaptation* and that of the covariance matrix *covariance matrix adaptation*.

We notice that the above description of a $(\mu/\mu_w, \lambda)$-ES makes it invariant to strictly increasing transformations of the fitness function. This means that a $(\mu/\mu_w, \lambda)$-ES is expected to perform the same on a problem $f(\mathbf{x})$ as on any problem $(g \circ f)(\mathbf{x}) = g(f(\mathbf{x}))$ with $g : \mathbb{R} \to \mathbb{R}$ a strictly increasing function ($y_1 > y_2 \implies g(y_1) > g(y_2)$). In order to preserve this property, the distribution parameters need to be updated by using, as information, only the ranking of the solutions (2.11), not their actual function values. Invariances are a well desired concept when designing algorithms since they allow to generalize results found on one problem to a class of problems sharing the properties to which the algorithm is invariant. This is particularly sought-after in evolution strategies whose use of standard normal distributions allows a number of theoretical studies of performance and converge rates to the optimal. We will see some examples of such important theoretical results in Chapter 3.

2.1.4 The Covariance Matrix Adaptation Evolution Strategies

The Covariance Matrix Adaptation Evolution Strategy is the reference evolution strategy for solving black-box continuous optimization problems. As such, it is mainly concerned in the adaptation of the parameters of the multi-variate normal distribution, the mean solution $\mathbf{x}_t$, the step-size σ_t and the covariance matrix $\mathbf{C}_t$, used to sample the solutions. It relies on a number of concepts that were introduced through the years to result in the algorithm described in [Hansen et al., 2003].

The first concept that was introduced is that of de-randomization in [Ostermeier et al., 1994a]. In a de-randomized algorithm, the source of randomness used on the solution parameters (to generate the offspring) is the same one that, after transformation, is used to adapt the parameters of the strategy [Hansen et al., 1995]. In other words, the mutation of the internal parameters of the algorithms depends deterministically on the realizations of the random variables used to generate the solutions (a *single* source of randomness is used).

The second concept is that of covriance matrix adaptation [Hansen and Ostermeier, 1996]. Correct covariance matrix adaptation allows the algorithm to be invariant to any rotation of the coordinate system, resulting in a coordinate system independent algorithm. In addition, by learning the correct scalings of the axes within the covariance matrix, there is no need for a step-size for each dimension but a single, global, step-size σ_t suffices to represent, coupled with $\mathbf{C}_t$ and $\mathbf{x}_t$, any multi-variate normal distribution. An alternative covariance matrix adaptation method was introduced earlier in [Schwefel, 1981]. This methods relies on learning the individual step-sizes of each dimension (d variances) and the $d(d-1)/2$ rotation angles (number of degrees of liberty of a rotation in $\mathbb{R}^d$) that compose the covariance matrix. However, [Hansen et al., 1995] showed experimentally that, because of the canonical base rotations that are used, this approach is coordinate-system dependent.

One of the most important concepts that are used in CMA-ES is the concept of cumulation that was introduced in [Hansen and Ostermeier, 2001]. In cumulation, the update does not rely only on the information gathered form the current iteration but also on that of previous iterations. The steps/direction that were performed in the previous iterations, when updating the different parameters of the strategy (in this case, the step size σ_t and the covariance matrix $\mathbf{C}_t$), are taken into account when updating these same parameters in the current iteration.

Later, the CMA-ES was enhanced and its performance improved, especially when relatively larger population sizes are at play, by the addition of what is called a *rank-μ update* of the covariance matrix in [Hansen et al., 2003]. Contrarily the the rank one update that only uses the information from the step performed by the mean solution, a rank μ update takes into account a weighted sum of the steps, in isotropic space, of the μ selected offspring. The use of the rank-μ update significantly improves the speed at which the covariance matrix is learned, especially when the population size is high.

2.1.4.1 The $(\mu/\mu_w, \lambda)$-CMA-ES

In order to describe the $(\mu/\mu_w, \lambda)$-CMA-ES algorithm with rank-μ update [Hansen et al., 2003], we show the update equations of the step-size and the covariance matrix; the rest of the algorithm uses the standard $(\mu/\mu_w, \lambda)$-ES update equations seen in Section 2.1.3.The offspring generation and mean solution update are carried out the same way as in a default $(\mu/\mu_w, \lambda)$-ES (see equations (2.10), (2.11) and (2.12)). Note that (2.10) can also be written:

$$\mathbf{x}_t^i \sim \mathbf{x}_t + \sigma_t \mathbf{C}_t^{1/2} \mathbf{N}(0, \mathbf{I}_d) \ , \tag{2.13}$$

where $\mathbf{I}_d$ is the identity (eye) matrix in dimension d (all entries are zeros but the diagonal entries who are ones) and $\mathbf{C}_t^{1/2}$ refers to the unique positive-definite symmetric square root matrix of $\mathbf{C}_t$, that is $\mathbf{C}_t^{1/2}\mathbf{C}_t^{1/2} = \mathbf{C}_t$ and $\mathbf{C}_t^{1/2}$ has positive eigenvalues.

Cumulation is applied to both the step-size and covariance matrix using two respective evolution paths, p_t^σ and p_t^c; both initially $\mathbf{0}_d$, the vector of zeros of size d (similarly, we note $\mathbf{1}_d$ the vector of ones of size d). The step-size evolution path is updated as follows

$$p_{t+1}^\sigma = (1-c_\sigma)p_t^\sigma + \sqrt{c_\sigma(2-c_\sigma)}\sqrt{\mu_{\text{eff}}}\mathbf{C}_t^{-1/2}\frac{\mathbf{x}_{t+1}-\mathbf{x}_t}{\sigma_t} \quad , \tag{2.14}$$

where $c_\sigma \in [0,1]$ a discount factor that determines the cumulation time for p_t^σ ($\approx 1/c_\sigma$), $\mu_{\text{eff}} = 1/\sum_{i=1}^{\mu} w_i^2$ the *variance effective selection mass* and $\mathbf{C}_t^{-1/2}$ is defined similar to $\mathbf{C}_t^{1/2}$ but with regards to $\mathbf{C}_t^{-1}$, the inverse of $\mathbf{C}_t$, instead ($\mathbf{C}_t^{-1}\mathbf{C}_t = \mathbf{I}_d$). Thus, p_t^σ is updated by taking into consideration the displacement of $\mathbf{x}_t$ in the isotropic space.

On the other hand, the path of the covariance matrix is updated in a similar fashion but not in the isotropic space (no multiplication by $\mathbf{C}_t^{-1/2}$)

$$p_{t+1}^c = (1-c_c)p_t^c + \sqrt{c_c(2-c_c)}\sqrt{\mu_{\text{eff}}}\frac{\mathbf{x}_{t+1}-\mathbf{x}_t}{\sigma_t} \quad , \tag{2.15}$$

where $c_c \in [0,1]$ another discount factor that determines the cumulation time for p_t^c (which is $\approx 1/c_c$).

These two paths are then used to update the parameters of the distribution. The new step-size is computed as follows

$$\sigma_{t+1} = \sigma_t \times \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|p_{t+1}^\sigma\|}{E\|\mathbf{N}(0,\mathbf{I}_d)\|} - 1\right)\right) \quad , \tag{2.16}$$

where d_σ is the damping parameter that controls the amount of change of the step-size in one iteration, and $E\|\mathbf{N}(0,\mathbf{I}_d)\|$ is the expected length of the path under random selection (the offspring are ordered randomly). Thus, the step-size adaptation mechanism, Cumulative Step-size Adaptation (CSA), compares the length of the cumulation path p_{t+1}^σ to the length that would be obtained under random selection and increases or decreases the step-size accordingly. Note that as a result of the update, $\log\sigma_t$ follows an unbiased random walk under random selection.

For the covariance matrix:

$$\mathbf{C}_{t+1} = (1-c_1-c_\mu)\mathbf{C}_t + c_1 p_t^c (p_t^c)^\mathsf{T} + c_\mu \sum_{i=1}^{\mu} w_i \frac{\mathbf{x}_t^{i:\lambda}-\mathbf{x}_t}{\sigma_t}\left(\frac{\mathbf{x}_t^{i:\lambda}-\mathbf{x}_t}{\sigma_t}\right)^\mathsf{T} \quad , \tag{2.17}$$

where $c_1, c_\mu \in [0,1]$ are the learning rates for, respectively, the rank-one update ($p_t^c(p_t^c)^\mathsf{T}$) and the rank-μ update ($\sum_{i=1}^{\mu} w_i \frac{\mathbf{x}_t^{i:\lambda}-\mathbf{x}_t}{\sigma_t}\left(\frac{\mathbf{x}_t^{i:\lambda}-\mathbf{x}_t}{\sigma_t}\right)^\mathsf{T}$).

The internal complexity of the $(\mu/\mu_w, \lambda)$-CMA-ES as presented above is in $O(d^3)$ per iteration. This complexity comes from the singular value decomposition of $\mathbf{C}_t$ needed

both for sampling the solutions (2.13) and updating the isotropic path (2.14). In order to reduce this complexity, [Hansen and Ostermeier, 2001] suggests to perform the decomposition only once every a linear number of iterations in the dimension of the problem ($d/10$) since the covariance matrix is not expected to change significantly in the course of a few iterations. This amount of change depends, not only on the landscape of the function, but also on the different learning rates of the covariance matrix (c_1 for the rank one update and c_μ for the rank μ update) and on the discount factor for the covariance matrix cumulation path c_c. The resulting complexity is in $O(d^2)$ which is acceptable in a small to medium dimension setting; it is also the complexity of computing a vector-matrix multiplication. However, further complexity reductions will be needed for the algorithm to be usable in a large-scale setting as we will see in Section 2.2.1.3.

2.1.4.2 CMA-ES variants

Given its good performance on a variety of problems, several variants of CMA-ES were developed through the years, either to further improve its performance both in general and on specific function classes or to be applied in different optimization contexts such as multi-objective, expensive, constrained and large-scale optimizations.

Restarted CMA-ES
After [Hansen and Kern, 2004] established that larger population sizes improve the performance of CMA-ES on a number of multi-modal problems, [Auger and Hansen, 2005b] proposed an increasing POPulation size CMA-ES (I-POP-CMA-ES). In I-POP-CMA-ES after each restart triggered by a met stopping criterion (the stopping criteria of CMA-ES are documents in [Hansen, 2009]), the population size is multiplied by 2. BI-POP-CMA-ES [Hansen, 2009] relies on two regimes when changing the population size. The first regime doubles the population size the same as I-POP while in the second regime, the population size is decreased to a value in the interval $[\lambda_{\text{def}}, \lambda_l]$, with λ_{def} the initial, default, population size and λ_l the last value of the population size (largest) obtained at the end of the first regime. Both variants are further tweaked in [Loshchilov, 2013] with a decreasing initial step-size upon each restart for I-POP in NIPOP and an adapted budget for the two regimes depending on their performance in NBIPOP. The regime with the better best-found solution is given double the budget of the other regime. These changes resulted in some improved performance especially on multi-modal functions.

Online Parameter Adaptation
An online adaptation scheme for other parameters of CMA-ES (other than the population size) is proposed in self-CMA-ES [Loshchilov et al., 2014]. The main idea of the algorithm is to choose a parameter setting that maximizes the likelihood of generating the best fit individuals (similarly to what is done in EDAs). In order to find these parameter values, a second CMA-ES algorithm, called *auxiliary* CMA-ES, is run on the parameters of the algorithm while the *primary* CMA-ES operates in the search-space of the problem. The proposed algorithm improved the performance of CMA-ES when considering population

sizes that are larger than the default ones.

Meta-Model Based Variants

Several meta-model based methods were developed for CMA-ES. These methods are generally suited for expensive optimization where the cost of performing a function evaluation is high and thus only a small budget can be afforded. They try to approximate the original fitness function with a given model that they optimize instead of the original fitness, thus avoiding to deplete the budget. The best solutions found on the meta-model are then evaluated on the original function and injected to the model so it can be updated.

$(1, \lambda)$-LS-CMA-ES [Auger et al., 2004] uses a quadratic model to approximate the fitness functions. The model is constructed by minimizing the square error observed on the quadratic model on the d^2 most recently evaluated points on the original function. Two modes are used, a local search mode (LS) and a CMA mode. The LS mode is used whenever the precision of the approximated model exceeds a given threshold, otherwise, the CMA mode is used. The model is re-computed once every $n_{\text{upd}} = 100$ iterations.

In lmm-CMA-ES [Kern et al., 2006] a locally weighted quadratic regression (locally weighted regression [Atkinson et al., 1997] with a full quadratic model) is used to build a model that gives importance to the points (in the form of the weights associated) depending on their distances to a current central solution. The covariance matrix learned by CMA-ES is used to defined the metric needed for this weighting. The paper also investigates the performance loss of CMA-ES given the number of perturbations of the offspring rankings and finds out that the performance loss has a minor dependency on the fitness function. [Bouzarkouna et al., 2010] investigates the shortcomings of lmm-CMA-ES when larger population sizes and numbers of parents are considered and proposes an alternative to the too restrictive condition for acceptance of a meta-model. The original condition which was to preserve the ranking of all the μ best solution is replace by a less restrictive condition of having the *same* μ best solution, regardless of their order except the best individual which needs to remain the same between the model and the original fitness function. After $\lambda/4$ evaluations on the original function, only the criterion on the best individual is verified. The proposed method, nlmm-CMA-ES, outperforms lmm-CMA-ES especially with larger than default population sizes thanks to its less restrictive conditions on the meta-model precision. nlmm-CMA-ES is applied to the real-world problem of well placement in [Bouzarkouna et al., 2012] showing better performance than a Genetic Algorithm but improved little over the default CMA-ES. In [Auger et al., 2013], lmm-CMA-ES is benchmarked using another relaxed acceptance criterion for the meta-model which now requires a threshold precision. In addition, the meta-model construction phase is started earlier. The resulting algorithm shows results on par with those of saACM [Loshchilov et al., 2012a] and improves on the original lmm-CMA-ES on most functions.

[Loshchilov et al., 2010] proposes to use rank-based meta-models (in this case a rank-based Support Vector Machine) as the meta-model of the algorithm. By doing so, the surrogate algorithm preserves the important invariance property of the CMA-ES to monotoneous transformations of the fitness function. This approach was improved

in [Loshchilov et al., 2012b] by self-adapting the hyper parameters of the model using an internal CMA-ES algorithm and setting its lifespan (number of iterations before it is retrained) depending on the ranking error made by the surrogate on the new set of points. A more intense exploitation of the meta-model by using larger population sizes when optimizing it (the model) while keeping the default population-size on the original function is proposed in [Loshchilov et al., 2013b] in order to improve the performance, especially on uni-modal functions. The algorithm is further augmented by adding a line-search method for better performance on separable function and using the NEWUOA [Powell, 2006] algorithm in the first $10 \times d$ evaluations given its rapid convergence on a number of simple functions. The resulting algorithm is called Hybrid CMA (HCMA) [Loshchilov et al., 2013a] and shows the best performance on the Black-Box Optimization Benchmarking workshops on a number of functions.

Multiplicative Covariance-Matrix Update and Multi-Objective CMA-ES
A different approach to covariance matrix adaptation that uses a multiplicative update instead of the additive update in (2.17) is proposed in [Krause and Glasmachers, 2015]. This approach was first applied to the Natural Evolution Strategy (NES) [Wierstra et al., 2008] in xNES [Glasmachers et al., 2010] and proposes to update the covariance matrix through matrix exponentiation that can be carried out with a complexity that is similar to that of the additive update.

In [Igel et al., 2007], a Multi-Objective CMA-ES algorithm (MO-CMA-ES) is introduced that uses a number of $(1 + \lambda)$-CMA-ES algorithms to search for a Pareto front (in multi-objective optimization, we are generally more interested in finding a set of non-dominated solutions, the Pareto front, than in finding a single best solution. In fact, due to the multiple objectives, a pair of solutions can end up being non-comparable, generally leaving the choice of which of them is better in the hands of experts on the problem).

A number of large-scale variants of CMA-ES were also developed especially in the last few years. These variants will be discussed in Section 2.2.1.

2.2 Large-Scale Continuous Optimization

The ever increasing computational power at the disposal of engineers and scientists is naturally accompanied with a desire to solve larger and more complex problems. This leads to more complex and more accurate models that are generally described by larger numbers of parameters; which in turn increases the appeal of large scale optimization. This can, for example, be seen on neural networks, and more specifically, deep neural networks [Claudiu Ciresan et al., 2010, Coates et al., 2011, Dean et al., 2012, Krizhevsky et al., 2012, Ngiam et al., 2011, Silver et al., 2016, Bottou et al., 2016]. These problems end up having large numbers of parameters, namely the weights of the network that need to be optimized. This is generally done using gradient descent on a fitness function that is a least square error. Some other less recent examples include meteorology [Wang et al., 1998] and shape optimization: turbine blades [Sonoda et al., 2003], aircraft wings [Vicini and Quagliarella, 1999], heat exchangers [Foli et al., 2006].

The large number of parameters involved in these so called large-scale problems makes solving them using *regular* optimization methods unpractical. This is mainly due to their complexities or computational costs with regards to the problem dimensions. For example, CMA-ES (see Section 2.1.4) by adapting a full covariance matrix needs, in order to store this covariance matrix, an amount of memory which scales quadratically with the number of variables. The sampling of offspring is also at least quadratic (we have seen in the last paragraph of Section 2.1.4.1 that it is originally cubic when a full covariance matrix is adapted) in time as it requires a matrix vector multiplication. These quadratic complexities are generally considered un-affordable in a large scale setting since they only allow to tackle, within reasonable time and space costs, low to medium dimensions. Thus large-scale optimization requires specifically designed algorithms with additional constraints on the cost.

We can divide the methods and algorithms used to solve large-scale optimization problems into two main categories: direct approaches and divide and conquer approaches.

2.2.1 Direct Approaches

Direct approaches solve the problems as it is, as a whole. The algorithms in question are generally variants of standard algorithms that perform well on smaller dimensions. These variants are conceived while taking into account the large-scale specific constraints, mainly constraints of space and time costs in addition to potential properties that large-scale problems might present (low effective dimensionality that will be seen in Chapter 4, limited variable interactions....).

2.2.1.1 Descent-Based Approaches

Gradient Descent and Newton Method

Some of the earliest large-scale optimization algorithms were designed around the gradient descent and Newton approaches. Gradient descent follows the negative of the gradient of the function (when available) in order to improve the fitness and, eventually, find a local optimum. In optimization, the Newton method is used in order to find the zeros of the gradient of the function, and thus potential local optima. Note that in a black-box setting, the information about the gradient or the Hessian is not available, and thus can only be estimated by the algorithm.

The Broyden-Fletcher-Goldfarb-Shanno algorithm (BFGS) that was developed independently in [Broyden, 1970], [Fletcher, 1970], [Goldfarb, 1970] and [Shanno and Kettler, 1970] is a quasi-Newton method that relies on the estimation of the inverse of the local Hessian matrix of the problem using rank one updates. Quasi-Newton methods are generally used when the Hessian matrix is unavailable (for example in black-box optimization) or too expensive to compute (high dimension). Instead, an estimation of this matrix is computed and used in order to find the extrema of the function.The large-scale version of the algorithm, called limited memory BFGS, first introduced in [Nocedal, 1980] and then formalized for large-scale optimization via L-BFGS in [Liu and Nocedal, 1989], is a variant that, instead of estimating a full, dense, inverse of the Hessian matrix,

stores a certain, limited, number of vectors that were previously generated. The matrix is then simulated using these same vectors and never stored explicitly as a full matrix. In the same context, [Steihaug, 1983] used a trust region mechanism (a local, generally quadratic, approximation model of the fitness function whose size changes depending on its accuracy) coupled with a conjugate gradient to solve the local approximation.

More recently, [Liao et al., 2005] proposes a gradient-based method that relies on solving an ordinary differential equation that approximates the optimization problems while [Fasano and Lucidi, 2009] introduces an adaptive line-search approach that uses two directions: an approximation of a Newton step and a negative curvature direction. A different approach, on a particular class of problems, is introduced in [Nesterov, 2014]. In this paper, problems with sparse sub-gradients (generalization of gradient for non-differentiable functions) are considered and this sparsity is taken advantage of when computing matrix vector products. Because of this sparsity, only a few entries are expected to change from one iteration to an other; this allows an efficient update of the results of these matrix-vector products and coupled with simple sub-gradient methods, results in algorithms that can handle large-dimension problems. A summary of the performance of several gradient-based methods on large-scale problems can be found in [Yuan, 2010].

Coordinate Descent

Coordinate-descent algorithms consider sub-problems where one coordinate (or block of coordinates) is optimized at a time. The problems are solved by iterating over these sub-problems until a satisfactory solution is found (multiple passes are generally needed on non-separable problems). Coordinate-descent algorithms are generally used to solve convex problems with certain known properties such as having the expression of the gradient or being partially separable. They are not often applied in the context of *complete* black-box optimization, and do not perform well on fully non-separable problems. However, they remain an important class of algorithms for large-scale continuous-domain optimization thanks to their reduced complexity. In addition, in the recent years, and with the rise of *big data*, these methods, along with other convex descent-based optimization approaches, saw an increasing interest. This is especially the case when dealing with what is commonly referred to as huge-scale problems (problems with millions of variables where, in extreme cases, even linear complexities are considered prohibitive). They are, in fact, some of the few methods that can be applied in such scenarios. In addition, one does not expect huge-scale problems to have a fully dependent structure (all variables depend on all the other variables) but rather to have groups of dependent variables and a generally sparse dependence matrix.

[Nesterov, 2012] treats the case of (non black-box, convex) huge-scale optimization problems. In order to limit the amount of expensive operations that are needed (even vector additions are considered expensive in this scenario), the proposed approach, Random Coordinate Descent Method (RCDM), applies partial updates on randomly chosen variables at each iteration. The sampling of the random coordinates, which generally has a complexity of $O(d)$, is done in logarithmic time to accommodate the huge-scale scenario. The results obtained in [Nesterov, 2012] are extended in [Richtárik and Takáč, 2014] to

composite problems using randomized block-coordinate descent. In block-coordinate descent, at each phase, the optimization process is focused on a single *block* of variables that it optimizes; thus reducing the per-iteration complexity of the algorithm which no longer depends on the total number of variable d but only on the size of the current block (which is generally significantly smaller). This is similar to the block-diagonal structure which will be in the core of the dependency-introducing transformation proposed in Chapter 5.

In [Richtárik and Takáč, 2012], random and greedy choices of the descent coordinates are considered on huge-scale (dimensions up to more than 3×10^6) instances of the Truss Topology Design problem (real-world optimization problems encountered when designing mechanical structures such as bridges).

A parallelized coordinate-descent method, similar to the Cooperative Coevolution approach described in Section 2.2.2, is proposed in [Richtárik and Takác, 2013]. The variables of the problem are partitioned and assigned to different nodes. Each node then performs, at each iteration and in parallel, a *standard* coordinate descent on the set of variables it was assigned. The resulting algorithm, named Hydra for HYbriD cooRdinAte descent method, is applied to loss minimization in big-data problems.

Convex partially-separable functions with a block structure are considered in [Richtárik and Takáč, 2016][1]. The idea is to take advantage of the block-separable structure of the function in order to speed-up the optimization process via the parallelization of independent sub-components (a perfect scenario would be a speed up equal to the number of separable sub-components). The proposed approach serves also as a more general framework that includes [Richtárik and Takác, 2013] as a special case (with regards to the sampling of the blocks).

Smooth but non-convex problems with a predefined structure are studied in [Patrascu and Necoara, 2015] (and also in [Lu and Xiao, 2013]) proposing random coordinate descent approaches for solving them. In [Patrascu and Necoara, 2015], the problems are formed as the sum of a black-box smooth non-convex term and a convex term with known structure. The proposed method, called 1-Random Coordinate Descent method (1-RCD), approaches the problem one variable (or block of variables) at a time and solves a quadratic approximation of the problem (which is accurate as long as the smooth non-convex function satisfies some gradient continuity conditions). Once a variable (or block) is chosen, the solution is updated by following a direction that minimizes the quadratic approximation.

Large-Scale Optimization on Classification Problems

A number of coordinate descent (as seen above), gradient descent and Newton methods are used in solving large-scale classification problems. These methods do, generally, not consider a black-box setting since the fitness function is known and given in a closed form that the solver can exploit. It is even up to the user to choose the fitness function for a given classification problem that generally includes a penalization term (for the wrongly classified examples) and a regularization term (to keep control on the parameters such as

[1] [Richtárik and Takáč, 2016] and [Richtárik and Takác, 2013] were written and submitted around the same time period.

preventing them from becoming too large or having too complex models). For example, in Support Vector Machine (SVM) models, two main approaches are used to penalizing: L_1-SVM for a linear penalization and L_2-SVM for a quadratic one.

Examples of notable large-scale algorithms applied to linear classification include stochastic gradient descent (the gradient descent is applied on only one example per iteration) in [Zhang, 2004] that was later extended upon and coupled with sub-gradient projections in PEGASOS [Shalev-Shwartz et al., 2011]; a coordinate descent algorithm on the dual problem in Dual-CD [Hsieh et al., 2008a]; a trust region based newton method TRON [Lin and Moré, 1999] that was applied in [Lin et al., 2008] to logistic regression on large-scale classification problems and also on non-linear SVMs and SVM$^{\text{perf}}$ [Joachims, 2006] which uses a cutting planes methods (starts with an empty set of constraints that is, at each step, augmented with the, new, most violated constraint) on an alternative formulation of the SVM optimization problem. A good overview of large-scale optimization algorithms applied to linear classification, mainly SVMs, can be found in [Yuan et al., 2012].

One thing to note about most of the methods presented in this section is that they consider the problem of converging to a state where the gradient tends to zero. They are, by default, not suitable for multi-modal optimization and may perform sub-optimally on non-smooth functions. They are also mostly concerned about solving the convex problems that generally appear when dealing with machine-learning problems such as classification problems.

2.2.1.2 PSO, DE and EDA Variants

The recent years saw an increase in the use of population-based algorithms to solve continuous optimization problems. More specifically, Evolutionary Algorithms (EA) based, for the most part, on Particle Swarm Optimization (PSO), Differential Evolution (DE) and Evolution Strategies (ES).

PSO

In the variants of PSO (see Section 2.1.2.2 for a brief description), Rotated Particle Swarm [Korenaga et al., 2007] applies a rotation to the coordinate system to address the degeneracy in particle speeds observed on large-scale problems. The rotation is applied on a limited number of pairs of axes such that the algorithm remains computationally reasonable is high dimensions. Dynamic Multi-Swarm (DMS-PSO) [Zhao et al., 2008] uses a large number of sub-swarms of reduced sizes. After each sub-swarm is done with its search, a proportion of the best solutions is refined using a quasi-Newton local search. The particles are then shuffled to form new sub-swarms and the process is reiterated. After a certain number of iterations, the overall best found solution is refined with another local search. In Dynamic Neighborhood Topology PSO (PSO-DNT) [Han and Fan, 2010], the particles are, here also, clustered into sub-swarms. A *neighbourhood diversity* measure is used in order to detect when a sub-swarm reaches a local optimum, thus hopefully allowing to dynamically renew the neighborhood connections when needed.

Incremental PSO with Local Search (IPSOLS) [Montes de Oca et al., 2008] uses an increasing population size based on the framework first proposed in [Montes de Oca and Stützle, 2008]. A local search procedure is also used to try and improve the best solution found by each particle. It was redesigned in [Montes de Oca and Stützle, 2011] to handle large-scale problems via an automatic algorithm configuration using Iterated F-Race[2] [Birattari et al., 2010] on problems of smaller dimensions than the original problem. An other approach to population size adaptation for large-scale PSO is the Efficient Population Utilization Strategy for PSO (EPUS-PSO) [Hsieh et al., 2008b]. In EPUS-PSO, if the global best solution found does not improve after a given number iterations k, the population size in incremented as long as it does not exceed a fixed threshold, in which case it is decremented instead to allow room for a new particle. This increase in the population size is done in order to promote exploration. On the other hand, if the global best solution is improved in successive iterations, exploration is encouraged by decrementing the population size This is done in hope of increasing the speed and efficiency of exploitation and of reducing the redundancy of solutions.

DE

In the area of DE (see Section 2.1.2.1 for a brief description), jDEdynNP-F [Brest et al., 2008] is a self adaptive differential evolution algorithm that incorporates a population reduction method. The population size is halved with a frequency that grants each population size value an equal number of function evaluations (thus the larger, more costly, population sizes are given less iterations). In the DE/current-to-$p$best mutation strategy, instead of selecting the best individual for mutation, one is selected randomly from the the $p \times 100\%$ best individuals.

EDA

EDAs, and more specifically Gaussian-Distribution based EDAs generally do not scale well in the problem dimension since they need to estimate the covariance matrix (matrices) of the distribution which, by default, contain quadratic numbers of elements; thus are not directly applicable in large-scale. Similarly to what we will see in Section 2.2.1.3 for Evolution Strategies, large-scale approaches for Gaussian Distribution based EDA rely mostly on using a restricted model of the covariance matrix to limit both the learning and the sampling complexities.

The earlier implementations of EDAs in [Mühlenbein and Paass, 1996] and [Mühlenbein et al., 1996] consider what is called a univariate EDA. In a univariate EDA, only the means and variances of the variables are learned, which results in a linear complexity and makes the approach applicable for high dimensions. It can also be seen as restricting the learning of the covariance matrix to only the diagonal elements (the rest are set to zeros), similarly to separable CMA-ES [Ros and Hansen, 2008]. As noted in [Bosman

[2] In F-Race, configurations are run in parallel with the worst configurations being gradually discarded. The iterated version does so on several iterations; the candidate configurations are sampled from distributions that are updated, upon each iterations, by taking into account the best configurations of the previous iteration.

et al., 2013], considering a diagonal covariance matrix reduces the number of problems solved by an EDA. However, the computational time needed with the diagonal approach is considerably lower than when estimating a full-covariance matrix; which makes it a reasonable alternative and compromise when dealing with large-scale problems. In [Wang et al., 2010], a univariate EDA, MUEDA [Wang and Li, 2009] that uses a mixed Cauchy-Gaussian distribution, is run in the first phase of optimization. Then, once a predefined stopping criterion is triggered (small improvement of the best fitness over a number of iterations), a differential evolution algorithm is used in order to better improve the results on the region of interest identified by the univariate EDA. The resulting serial cooperation based algorithm, named ED-DE, is applied to a representation of a multi-model real-world problem (Economic Load Dispatch), and produced improved best-known solutions on a number of instances of this problem. It also compares well, performance-wise, to a number of classical DE and PSO variants on standard benchmark functions that are generally used in continuous optimization (although only on dimension $d = 30$).

[Dong et al., 2013] uses two steps in the large-scale EDA it proposes. First *independent* variables are identified using a method called Weakly Dependent variable Identification (WI) which computes the correlation, in the set of selected solutions, between the different variables of the problem and assumes independent (with regards to all the other variables) each variable whose correlations are all smaller than a given threshold. The independent variables form a class W and the rest of the, assumingly dependent, variables another class S. A full model is to be learned for the variables in S while a diagonal, univariate, one is used for the variables in W (since they are supposed independent, so have null correlations). Often, the size of the subset S remains too large, especially in a large-scale settings, both for the computational cost that one can afford and also given the relatively reduced number of selected offspring (in comparison to the dimension of S) which affects the precision of the model. Thus S is randomly partitioned into a number of c subsets (of variables) and a multi-variate model is learned on each subset by projecting the selected offspring into the subspace spanned by this subset. So, the same set of selected solutions is taken advantage of multiple-times by projecting it into the different subspaces spanned by these subsets. This approach can be seen as learning a restricted block-diagonal covariance matrix with the variables in W all belonging to blocks of size 1 (part diagonal part block-diagonal matrix).

In [Kabán et al., 2015], a number of sub-spaces of smaller dimensions than that of the problem are defined using random projection matrices. The selected solutions are, here also, projected into these sub-spaces and a covariance matrix is learned for each sub-space. By learning the covariance matrix in the sub-spaces, this approach reduces the computation cost of the algorithm (in comparison to learning in the original search-space). Once the covariance matrices are learned, new solutions are sampled in each sub-space and combined, using scaled averages (to accommodate the loss in vector lengths and variance due to the orthogonal sub-space projections), in order to create the new population in the original search-space.

2.2.1.3 CMA-ES Variants

One of the earliest alternatives to CMA-ES for large-scale optimization was proposed in [Poland and Zell, 2001] with the Main Vector Adaptation (MVA) approach. The idea in MVA is to find the most desired mutation direction and follow it instead of adapting a full covariance matrix, making the algorithm less costly. In L-CMA-ES [Knight and Lunacek, 2007], the most prominent m eigenvectors (those associated with the largest eigenvalues) are learned instead of all the d vectors (assuming full rank) that form the full covariance matrix. In order to avoid searching in a sub-space of dimension $m < d$ and thus potentially be unable to solve simple functions (because of the optimum not being in the targeted subspace), an additional isotropic normal distribution on all dimensions is considered with a variance equal to the square of the smallest computed eigenvalue. Later, the separable CMA-ES algorithm (sep-CMA-ES) [Ros and Hansen, 2008] was developed with a restricted diagonal covariance matrix. With its linear number of free parameters, sep-CMA-ES is well suited for large-scale cases. However, and as experiments show, the diagonal covariance matrix allows to efficiently solve only problems that are separable or block-separable with a relatively low block-condition number (see Chapter 5).

VD-CMA-ES

VD-CMA-ES [Akimoto et al., 2014] proposes to use a covariance matrix that covers a larger set of problems by considering, in addition to the diagonal elements as in sep-CMA-ES, an additional vector as follows:

$$\mathbf{C} = \mathbf{D}(\mathbf{I}_d + \mathbf{v}\mathbf{v}^\mathsf{T})\mathbf{D} \ , \tag{2.18}$$

where $\mathbf{I}_d$ is the identity matrix in dimension d, $\mathbf{D} \in \mathbb{R}^{d\times d}$ is the diagonal matrix and $\mathbf{v} \in \mathbb{R}^d$. In this version of CMA-ES, the step-size is adapted using a variant of CSA where in (2.14) $\mathbf{C}_t^{-1/2}$ is replaced by $(\mathbf{I}_d+\mathbf{v}\mathbf{v}^\mathsf{T})^{-1/2}\mathbf{D}^{-1}$ in order to achieve linear complexity (the inverse of $\mathbf{I}_d + \mathbf{v}\mathbf{v}^\mathsf{T}$ can be computed in linear time using the Sherman-Morrison formula since $\mathbf{I}_d$ is diagonal). VD-CMA-ES was generalized into VkD-CMA-ES in [Akimoto and Hansen, 2016b] where instead of considering one additional vector $\mathbf{v}$, k vectors are considered:

$$\mathbf{C} = \mathbf{D}(\mathbf{I}_d + \mathbf{V}\mathbf{V}^\mathsf{T})\mathbf{D} \ , \tag{2.19}$$

where $\mathbf{V} \in \mathbb{R}^{d\times k}$ is no longer a vector but a matrix comprised of $k \in [0, d-1]$ vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ that need to be learned. The step-size is adapted using Two Point Adaptation [Hansen et al., 2014]. VkD-CMA-ES has the advantage of generalizing both sep-CMA-ES ($k = 0$) and the default full covariance matrix adapting CMA-ES ($k = d - 1$). One important parameter to set in VkD-CMA-ES is the number of vectors k since it impacts not only the performance of the algorithm (the larger the values of k the more likely the problem can be well approximated using the corresponding restricted covariance matrix) but also the complexity of the algorithm which is in $O(k \times d)$. The parameter k is automatically adapted online in [Akimoto and Hansen, 2016a]. The authors start by describing the effects that one expects to see when the covariance matrix is richer than needed or not rich enough given a convex-quadratic problem. The effect of dropping

each of the current axes is estimated by the change of condition number in the resulting matrix (after dropping said vector). If this results in an increase of the condition number that is not larger than a given threshold, β_{dec}, the vector is dropped (the algorithm also makes sure that it is not in a covariance-matrix adaptation phase that can lead to a larger decrease in the condition number if said vector is dropped). The number of vectors k is increased when the restricted covariance matrix does not change significantly and the step-size converges slowly. This is reflected in all the condition number increases being larger than a given threshold β_{inc}. The empirical results showed that the model learned by the proposed algorithms, and the number of vectors k, are nearly the optimal ones on many functions. It does, however, result in a slowdown of the performance on fully non-separable problems because of the time needed to adapt $k = d - 1$.

LM-CMA-ES

Another variant of CMA-ES, Limited Memory CMA-ES (LM-CMA-ES) [Loshchilov, 2014] was inspired from the limited memory BFGS [Nocedal, 1980] algorithm. It retains the same idea of not conserving the full covariance matrix but a set of the m last directions generated by the algorithm. These directions are then used to sample the new solutions without ever needing to reconstruct the full convariance matrix. LM-CMA-ES uses a new step-sized adaptation mechanism, the Population Success Rule (PSR) introduced in [Loshchilov, 2014] and inspired by the Median Success Rule (MSR) [Ait Elhara et al., 2013] which will be the main topic of Chapter 3. Similarly to MSR, PSR is a success-based rule whose complexity is not directly dependent on the problem dimension (can still depend on it through λ) which makes it usable in a large-scale scenario. A later version of LM-CMA-ES [Loshchilov, 2015] improves its performance by considering a number of vectors m in the square root of d instead of taking $m = \lambda$ whose default value is logarithmic in d. In order to remain in a reasonable complexity setting ($m = \sqrt{d}$ means the overall complexity would be in $O(d^{3/2})$), a smaller number m^* of vectors is used in order to generate each individual. Another significant change that was done in [Loshchilov, 2015] was to consider a Rademacher distribution (returns -1 or 1 equiprobably) instead of the normal distribution that is commonly used in evolution strategies. The sampling of Rademacher distributed variables is cheaper and PSR, the same as MSR, does not assume a particular distribution of the offspring which allows such an approach.

2.2.1.4 Other Methods

We can find, in the literature, a number of large-scale optimization algorithms that are not based on the algorithms described above. For example, Multi-Agent Genetic Algorithm (MAGA) [Zhong et al., 2004] combines a multi-agent system with a genetic algorithm while MA-SW-Chains [Molina et al., 2010] is a memetic algorithm that mixes a Steady State Genetic Algorithm (SSGA) [Whitley et al., 1989] with the *Solis and Wets'* algorithm [Solis and Wets, 1981], chaining different local searches by *passing on the parameters* from one to another. Memetic algorithms is a term that is generally used to describe hybrids of evolutionary algorithms and local search methods. Many of the examples cited above rely on this approach to improve the performance of the

evolutionary algorithms on large-scale problems. The idea is to rely on the EA mostly for the exploration part while the local search is used as an exploitation mechanism that tries to improve the best individuals found by the EA.

2.2.2 Divide & Conquer Approaches

The second category of approaches to solve large-scale problems relies on the divide and conquer paradigm (D&C), which, in large-scale continuous optimization, is mainly represented by the Cooperative Co-evolution (CC) approach, first proposed in [Potter and De Jong, 1994]. Unlike the direct methods that tackle the whole problem at once, the D&C methods divide the it into smaller, more affordable and easier to solve sub-problems that are, each, solved with an efficient optimization method.

In CC, the partial solutions are generally called *species*. Each species is evolved via an evolutionary algorithm, and its fitness is defined by that of the complete solution it forms with the current best representatives of the other species. Generally, the sub-problems are defined over subsets of variables that do not overlap, and when a sub-problems is optimized, the variable from the other sub-problems are kept constant. A solution to the global problem is, then, the aggregation of the partial solutions on each sub-problem (a cross over between the solutions of the sub-problems).

The grouping phase of the CC process is a crucial one, more so on non-separable problems where dependent variables need to be grouped and optimized together for the optimization to be efficient. Many of the CC based works focus specifically on finding efficient and well-performing ways of choosing these species. The original framework, which puts each variable in a group of its own, turned out to be inefficient whenever the problems it faces are non-separable. [Shi et al., 2005] proposed a dividing in half strategy (variables are put in two groups of equal size). This strategy performed significantly better on non-separable problems but has a major shortcoming in its scaling in d since it reduces the complexity of the sub-problems tackled by the underlying evolutionary algorithms by only a constant factor and applies the EA that same factor of times (dividing in half leads to 2 runs of the EA each on half the global dimension...). Most strategies divide the d variables into an intermediate number of groups n_G such as in [Van den Bergh and Engelbrecht, 2004] with an ideally upper-bounded largest sub-component size (in order to have a reasonable scaling in d).

In [Liu et al., 2001], the original CC approach is applied to Fast Evolutionary Programming [Yao et al., 1999] in order to speed-up its convergence rates on large-scale optimization problems. The results are promising, showing linear scaling of the performance in the problem dimension d. However, all the problems that were considered are separable problems, which does not tell much about the results that one expects to see on non-separable problems where the grouping process becomes more relevant.

Random Groups

DECC-I and DECC-II [Yang et al., 2007] use the Self-adaptive DE with Neighborhood Search (SaNSDE, described in details in [Yang et al., 2008c]) a variant of DE to solve

the sub-problems. DECC-I randomly, and uniformly, groups the variables into m sub-components. Each group is then optimized and given a weight, the weights of the best, the worst and a random group are evolved after each iteration. DECC-II overcomes the static grouping of variables in DECC-I by randomly selecting, at each iteration, a set of variables that will be optimized; the other variables are kept constant. Overall, DECC-I ended up better performing than DECC-II.

In [Yang et al., 2008a], the concepts of *random grouping* and *adaptive weighting* are introduced. The first generates new random groups at each iteration, and by doing so, increases the chances of evolving interacting/dependent variables together (see Section 3 of [Yang et al., 2008a] for the proof). The latter (adaptive weighting) attributes a weight to each of the m groups, these weights are then evolved (m dimensional evolution instead of the d dimensional one when considering the variables) in order to improve the overall solution quality. The process is iterated in what is called *cycles* and the whole algorithm is denoted EACC-G.

Co-Operative Micro Differential Evolution (COMDE) [Parsopoulos, 2009] applies co-operation to the micro DE which uses very small population sizes (a population size of 6 in [Parsopoulos, 2009]), regardless of the problem dimension d. The reduced population size allows for a fast converge which, however, prevents good exploration. This lack of exploration is addressed through the use of the cooperative scheme.

The Multi-Level CC (MLCC) [Yang et al., 2008b] tries to find the group sizes that fit the best to the considered problem. Small group sizes are expected to work better on separable problems while larger ones are generally necessary for a good performance on non-separable problems thanks to a larger chance of optimizing interacting variables together. At each cycle, each group size is given a probability of being selected that is based on its past performances. [Omidvar et al., 2010a] proposes DECC-ML, an improved version of MLCC. The paper starts by showing that a frequent variable grouping (once a cycle) improves significantly the chances to evolve interacting variables together, generalizing the result of [Yang et al., 2008a] to more than two variables. In addition to this, the selection of the group sizes is made uniform instead of weighted, and only carried out when the quality of the solution does not improve in two successive cycles. It also gets rid of the *adaptive weighting* that, the paper shows, generally fails to improve the fitness of the solution.

In variants of PSO for large-scale optimization, we can find Cooperative PSO (CPSO) [Van den Bergh and Engelbrecht, 2004]. CPSO is improved upon in CCPSO [Li and Yao, 2009] by injecting *random grouping* and *adaptive weighting*. CCPSO is further improved in CCPSO2 [Li and Yao, 2012] by mixing (using a probability p) between a *Cauchy* distribution and a *Gaussian* distribution for the sampling of new points in the PSO process. This is done to grant better search capabilities to the algorithm, a crucial aspect in large scale optimization. It also, like in [Omidvar et al., 2010a], removes the *adaptive weighting* in favor of a more frequent random grouping.

Variable-Interaction Based Groups

The concept of *Delta Grouping* is introduced in [Omidvar et al., 2010b] with the DECC-

D/DML algorithm. Delta Grouping aims at finding the interacting variables in order to group them together for improved performances on non-separable problems. The core idea of the concept is that interacting variables have generally narrow *improvement intervals* (intervals in which the fitness can be improved while keeping the other variables constant) [Salomon, 1996]. Thus, at each cycle, the average amount of change in each variable is calculated, and variables are sorted according to this amount; then, the grouping takes place by successively filling the groups with the sorted variables, increasing the chances of grouping interacting variables together. DECC-DML self-adapts the group sizes using simple random uniform sizes the same way as in [Omidvar et al., 2010a].

In Cooperative Coevolution with Variable Interaction Learning (CCVIL) [Chen et al., 2011], a different approach, first proposed in [Weicker and Weicker, 1999], is used to learn variable interactions. Two variables x_i and x_j are assumed dependent if there exist values x_i^1, x_i^2 for x_i and x_j^1, x_j^2 for x_j and values $(x_k)_{1\leq k\leq d, k\notin\{i,j\}}$ for the remaining variables such that:

$$f(x_1,\dots,x_i^1,\dots,x_j^1,\dots,x_d) < f(x_1,\dots,x_i^2,\dots,x_j^1,\dots,x_d) \tag{2.20}$$

and

$$f(x_1,\dots,x_i^1,\dots,x_j^2,\dots,x_d) > f(x_1,\dots,x_i^2,\dots,x_j^2,\dots,x_d)\ . \tag{2.21}$$

This means that the result of the comparison of the fitness for a pair of values of x_i, $f(\dots,\mathbf{x}_i^1,\dots)$ VS $f(\dots,\mathbf{x}_i^2,\dots)$, depends on the value of x_j for these same values of x_i (x_i^1 and x_i^2).The algorithm starts by considering all the variables independent and merges the dependent variables whenever equations (2.20) and (2.21) are both satisfied. At each step, equations (2.20) and (2.21) are verified using two individuals that are based on the current best individual and whose coordinates i and j (the variables whose dependence is investigated) are replaced with values from the current individual and a randomly chosen one. It does not consider a *round robin* tournament to detect all possible interactions (quadratic number of possible interactions), the dependency of the current variable/group is only checked with the previous one, with a random arrangement of variables generated at each cycle. The simulations show a good performance with regards to finding the correct number of groups of a problems; however, the cost of learning these groups in numbers of additional function evaluations (that are not exploited elsewhere) remains a major drawback of the algorithm. A similar approach, called Differential Grouping, was proposed in [Omidvar et al., 2014] where instead of a simple binary comparison of the fitness as in equations (2.20) and (2.21), the fitness difference is tracked by defining:

$$\Delta_1 = f(x_1,\dots,x_i^1,\dots,x_j^1,\dots,x_d) - f(x_1,\dots,x_i^2,\dots,x_j^1,\dots,x_d) \tag{2.22}$$

and

$$\Delta_2 = f(x_1,\dots,x_i^1,\dots,x_j^2,\dots,x_d) - f(x_1,\dots,x_i^2,\dots,x_j^2,\dots,x_d)\ . \tag{2.23}$$

Then if $|\Delta_1 - \Delta_2|$ is larger than a certain predefined threshold ϵ, the variables i and j are marked as dependent. [Sun et al., 2015] first notes the fact that differential grouping captures only direct interactions between variables. Then, it proposes the *Extended*

differential grouping that also identifies indirect interactions and groups indirectly interacting variables together. Differential grouping is further improved in [Mei et al., 2016] in what is called Global Differential Grouping by keeping track of the variable-interaction matrix (the value of $|\Delta_1 - \Delta_2|$ for each pair of variables). It is also one of the few algorithms that take advantage of the good performance of CMA-ES on small to moderate dimension problems and use it as the EA to optimize the sub-components. The resulting algorithm, CC-GDG-CMAES, outperformed its state of the art counterparts in The CEC 2010 Large Scale Global Optimization Special Session.

A good survey of large-scale optimization methods and especially of Cooperative Coevolution based ones can be found in [Mahdavi et al., 2015]. Overall these methods give promising results on several problems. However, their performance remains limited when dealing with fully non-separable problems since the approach consists in optimizing subsets of variables, of reasonably small sizes, together to bypass the use of an optimization algorithm on a high dimension problem.

2.3 Benchmarking

Benchmarking is an important task in algorithm design. It allows to test and assess the performance of algorithms before they deployment in a real-world setting. This is especially practical when resources in the real-world scenario are limited or when the runs take a long time to resolve so one can not afford a large number of runs. Benchmarking generally consists in, first, running the algorithm(s) in question on a set of predefined functions/problems. Then, the data produced by the experiments, ideally quantifying the performance, is processed and interpreted. A good benchmarking platform typically provides, in addition to the resources needed to run the experiments, additional tools to post-process the generated data and visualize it in a simple and easily interpretable way. The possibility of comparing the performance of several algorithms is in the core of the benchmarking procedure and should be provided in a simple way since it in the core of the benchmarking process. An example of a benchmarking platform that provides the features described above is COmparing Continuous Optimisers (COCO) that we will describe in the following section.

Given the purpose of benchmarking, and in order for them to be meaningful, the functions on which the algorithms are tested should relate to real-world problems. This is required in order for the performance of the algorithm on a real-world scenario to not differ from that on the benchmark, thus making the decisions taken on the benchmark as relevant as possible in the real world scenario (reduce the difference to the outcome observed on the benchmark). This results in a lesser tinkering effort needed on the, generally more expensive, real world problems.

One way to design benchmark functions that are similar to real-world problems is to identify the difficulties these real-world problems when confronting optimization algorithms. Then, one can represent these problems via functions that possess properties that translate into these same difficulties such as (see Section 2.1.1 for a more detailed but not exhaustive list of the difficulties encountered in continuous black-box optimization):

dimensionality, non-separability, multi-modality, ill-conditioning...

2.3.1 The BBOB-2009 test-bed

One widely used benchmarking platform in continuous black-box optimization is COmparing Continuous Optimisers, COCO (https://github.com/numbbo/coco). As previously mentioned, it provides both the tools needed to run the simulations (experiment code) in different programming languages (C, C++, Java, Python, Matlab and Octave for now) and a Python code that allows to post-process the data. Using the post-processing code of COCO produces several practical and useful plots such as the scaling plots (e.g., Figure 5.8) and the Cumulative Distribution plots (e.g., Figure 5.9). It also provides a set of tables and a data structure that allows the user to have access to and visualize individual data-sets and personalized data-set lists.

The BBOB-2009 test-bed, and more specifically the noiseless test-bed, is the core of the benchmarking procedure in COCO. It contains a set of 24 different problems whose definitions can be found later in Table 5.3 and Table 5.4 and that are based on *raw* functions that can be found in Table 5.1 and Table 5.2. Different transformations are applied on these raw functions in order to generate the 24 problems. These transformations will be detailed in Section 5.1.1. For each set of values of the parameters of these transformations, an *instance* of the problem is generated; which allows, in theory, an infinite number of instances for problem. The problems can also be defined on any dimension $d \geq 2$.

These problems are organized in five categories, each category containing a set of problems that share some property/difficulty:

Separable Functions (f_1-f_5): these functions are considered to be easy since they can be solved by a line search on each dimension. The Rastrigin functions (f_3 and f_4) present the additional difficulty of being highly multi-modal; which makes them harder for most algorithms that do not explicitly exploit the separable nature of these problems. These five functions are the only separable ones in the testbed.

Low/Moderately Conditioned Functions (f_6-f_9): these functions have a condition number of about 100. The two first functions are uni-modal (single optimum) while the Rosenbrock functions (f_8 and f_9) have a second, local, optimum in dimensions higher than 2. The non-rotated Rosenbrock function (f_8) is partially separable, with a band like dependency structure (each variables interacts directly only with its immediate neighbors).

Unimodal Highly Conditioned Functions (f_{10}-f_{14}): these functions have a single optimum. A descent method should, in theory (depending on the differentiability of the function, the presence of plateaus and how they are handled), be able to find the optimum. They have a relatively high condition number (10^6 for f_{10}, f_{11} and f_{12}) in comparison to the functions of the previous category.

Multi-Modal Functions with Adequate Global Strucure (f_{15}-f_{19}): these functions are highly multi-modal, with a number of local optima that depends on the dimension (e.g., exponential in the dimension for f_{15}). However, the adequate structure that might be exploited by algorithms means the local optima are generally similarly shaped and distributed in a regular way. Thus, seen on a global scale, the landscape of the function contains repetitive, symmetric, patterns.

Multi-Modal Functions with Weak Global Strucure (f_{20}-f_{24}): similar to the previous category in multi-modality but the landscapes of the function has less structure and symmetries are broken. Most algorithms find these problems to be the hardest.

In addition to its noisy test-bed and the possibility of having an expensive setting when visualizing the date, the COCO platform has recently extended its list of problem suites to include that of bi-objective problems [Tusar et al., 2016, Brockhoff et al., 2016]. The expensive setting introduces the notion of runlength-based target values. Instead of fixing constant target precisions (as it is done in the *classical* approach approach), the target precisions are, instead, generated depending on the performance of another algorithm (in the present version, an *artificial* portfolio algorithm of the best results collected in the 2009 BBOB workshop [Hansen et al., 2010b]) by choosing the targets that were just not reached at given run-lengths (numbers of functions evaluations). Furthermore, a constrained test-suite and a large-scale test-suite are in development to extend the platform even more. Chapter 5 of this thesis deals with the large-scale extension.

2.3.2 The CEC Benchmarks for Large-Scale Global Optimization

The most prominent large-scale continuous optimization benchmarks are the ones used in the CEC Special Sessions and Competitions on Large-Scale Global Optimization. The first iteration of this test-suite, proposed in [Tang et al., 2007] consisted in 20 problems organized in four classes depending on their levels of separability:

- separable functions,
- partially separable functions with a single group of dependent variables,
- partially separable functions with several independent groups each group comprised of dependent variables,
- fully non-separable functions.

The problems are constructed in a similar fashion to [Hansen et al., 2009] by applying transformations on a number of basic/raw functions. The CEC benchmarks are based on a subset of the raw functions used in COCO: the sphere function, the ellipsoid function, the Rastrigin function and the Rosenbrock function (see Table 5.1 and Table 5.2 without the normalization factor $\gamma(d)$) in addition to the Ackley function and a different variant

of the Schwefel function, Schwefel 1.2. The Ackley function, in its raw form, is defined as follows

$$f_{\text{raw}}^{\text{Ackley}}(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos\left(2\pi x_i\right)\right) + 20 + e \ , \quad (2.24)$$

where x_i designates the i^{th} coordinate of the vector $\mathbf{x}$. The variant Schwefel 1.2 on the other hand:

$$f_{\text{raw}}^{\text{Schwefel1.2}}(\mathbf{x}) = \sum_{i=1}^{d}\left(\sum_{j=1}^{i}x_i\right)^2 \ . \quad (2.25)$$

The partially separable problems are obtained by first dividing the variables into groups and then applying a full rotation matrix to each group. The complexity of evaluating the problems is limited thanks to constant group-sizes (set to 50 in the test-suite) that make the cost of applying the rotations (the bottleneck of the computation) reasonable despite the large-scale setting (since this group size is independent of the problem dimension).

The main shortcoming of this test-bed is possible exploitation of its *flat* group structure (non-overlapping groups of the same size). This was noted in [Omidvar et al., 2015], in addition to showing that the differential grouping approach [Omidvar et al., 2014] manages to learn the exact group structure on most problems of the benchmark. [Omidvar et al., 2015] also proposes a number of features that a large-scale benchmark needs to satisfy to reproduce real-world problem:

- non-uniform group sizes,
- non-uniform contributions of the groups,
- overlapping groups.

These features, in addition to a number of transformations borrowed from [Hansen et al., 2009], are integrated in order to construct the CEC 2013 Large-Scale Benchmark Functions [Li et al., 2013].

The problems are classified differently in the new test-suite [Li et al., 2013]:

- fully-separable problems,
- partially, additively, separable problems (contains problems with a group of separable variables and problems without a group of separable variables)
- overlapping problems (the groups share a number of variables, sub-problems formed by these groups may or may not have the same optimal values on the shared variables)
- fully non-separable problem (the Schwefel Problem).

We note that the Rosenbrock problem, and because if its band-like structure (all variables interact directly with the variables that are adjacent to them), is no-longer considered a fully non-separable problem but an overlapping one (here, groups of 2 to 3 interacting variables that overlap).

One major shortcoming in the CEC benchmarks is the performance evaluation procedure. In COCO, the performance is measured in terms of the *expected* average number of function evaluations needed to reach a given target precision. In the CEC benchmarks, a fixed budget of function evaluations is given and the performance is measured in terms of the best fitness observed once the budget is exhausted. In most cases, the best fitness is a measure that can not be interpreted in a quantitative way. For example, one does not know, a priori, and on different problems, the significance of the quantitative difference between a precision of 10^{-4} and a precision of 10^{-8}. The effort needed to improve from the first to the second depends on the problem, and can vary drastically form one problem to the other. In addition, applying an increasing transformation to the fitness function (such as a scaling with a positive factor) leads to difference performance results even for comparison-based algorithm (such as CMA-ES) that are inherently invariant to such transformations (this measure does not allow to see this invariance directly). On the other hand, using the number of function evaluations needed to reach a target precision allows to quantitatively compare algorithms and their speeds. An algorithm can be twice faster than an other in reaching a target f_Δ, can witness a speedup of 10 when a certain parameter is self-adapted... Results and interpretations that are easier to understand and apply.

Chapter 3

The Median Success Rule

This chapter aims at designing a new success based step-size adaptation mechanism for evolution strategies, and more specifically, for CMA-ES and its variants. The proposed mechanism has a complexity that does not directly depend on the problem dimension d, which makes it large-scale friendly. In addition, it relies on the presence of fewer properties of the sampling distribution than the step-size adaptation mechanism that it replaces (Cumulative Step-size Adaptation) and can thus, in theory, be applied and easily adapted to a large variety of population-based evolutionary algorithms. It can be seen as a generalization of the one-fifth success rule to multi-parent non-elitist evolution strategies.

3.1 Introduction

The default step-size adaptation mechanism in CMA-ES, the Cumulative Step-size Adaptation (CSA, see Section 2.1.4.1), and despite it being an efficient method with state of the art performance, has some shortcomings that make it unpractical to use in our large-scale context. CSA relies on having normally distributed offspring, limiting its use to algorithms that sample normally distributed offspring. It also performs better when it is coupled with a covariance-matrix adaptation-mechanism such as CMA-ES that presents it with an isotropic search space (after transformation via the covariance matrix, thanks to $\mathbf{C}_t^{-1/2}$ in (2.14)). The complexity of learning this covariance matrix is, at best, quadratic when done in a standard way (learning the full covariance matrix). Such levels of complexity are generally not allowed in large-scale settings. In addition, CSA needs the inverse of the covariance matrix in its update to end up in the isotropic space (see (2.14)), so a strategy that adapts the covariance matrix in an efficient (linear) way, such as VD-CMA-ES [Akimoto et al., 2014], needs to provide, in a no less efficient way, the inverse of the restricted covariance matrix it learns (as seen in Section 2.2.1.3, VD-CMA-ES does provide such a cheaply computed inverse by taking advantage of the particular covariance matrix it adapts). This leads to additional efforts needed if one wants to apply CSA in a large-scale setting. Other methods, such as LM-CMA-ES and the VkD-CMA-ES [Akimoto and Hansen, 2016b] (generalized VD-CMA-ES), follow a

similar approach to that of this chapter and use different, cheaper, step-size adaptation mechanisms for large-scale optimization (Population Success Rule [Loshchilov, 2014] in LM-CMA-ES and Two-Point [Hansen et al., 2014] Adaptation in VkD-CMA-ES). One other shortcoming of CSA is found in Chapter 4 in the form of its slow convergence rate on the simplest looking functions when these functions have low effective dimension. This is due to the fact that the step-size adaptation signal of CSA is derived from a comparison with the case of random selection, in which the steps are normally distributed. The expected length of these steps depends on the problem dimension but the algorithm uses a deceptive overall problem dimension d instead of the effective dimension of the problem d_{eff}. So, the algorithm mis-estimates the *true* expected value of the length of the cumulation-path under random selection.

We propose the Median Success Rule (MSR) as an alternative for CSA that can be used in large-scale optimization but also more generally as a step-size adaptation mechanism that can be integrated to evolution strategies, and eventually to a larger array of population-based algorithms when adapted properly. The proposed method and its complexities do not depend directly on the problem dimension so it can be used in a large-scale setting. Thus, MSR can be used with efficient covariance matrix approximation algorithms tackling large-scale problems. It is applied to an embedding-based version of CMA-ES in Chapter 4 which produced competitive results, surpassing CSA and being on par with other large-scale algorithms (it does not suffer from the problem encountered by CSA on low effective dimension problems). MSR can also constitute, coupled with the core of $(\mu/\mu_w, \lambda)$-ES (equations (2.10), (2.11) and (2.12)), a *cheap* stand-alone evolution strategy with no covaraince matrix adaptation.

In Section 3.2, we explain the notion of Linear Convergence and how it relates to the design of step-size adaptation methods. We introduce the Median Success Rule in Section 3.3 where we also present the details of its implementation. The parameters of MSR are investigated and tuned in Section 3.4 and the step-size adaptation mechanism is benchmarked against CSA in Section 3.5. We conclude in Section 3.6.

3.2 Step-size Adaptation and Linear Convergence

In continuous optimization problems, we are generally interested in the speed at which an algorithm finds or converges the optimal solution. It was already established that the step-size, and more specifically, the way it is adapted, impacts the speed at which evolution strategies converge to the optimal solution on a wide class of problems. In fact, and with a well adapted step-size, one can achieve what we call *linear convergence*, on a certain class of functions [Auger and Hansen, 2011]. Achieving a linear converge speed is generally the desired theoretical goal when conceiving evolution strategies in general and step-size adaptation mechanisms in particular. Thus, several methods of adapting σ_t can be found in the literate of the domaine such as the one-fifth success rule [Rechenberg, 1994], Self-Adaptation (SA) [Schwefel, 1995], Two-Point Adaptation (TPA) [Hansen, 2008] and Cumulative Step-size Adaptation (CSA) [Ostermeier et al., 1994b].

In the context of evolution strategies, an ES converges linearly, in expectation, when

there exists a positive constant CR that satisfies:

$$\lim_{t\to\infty} \frac{1}{\lambda} E\left[\ln \frac{\|\mathbf{x}_{t+1} - \mathbf{x}_{\text{opt}}\|}{\|\mathbf{x}_t - \mathbf{x}_{\text{opt}}\|}\right] = -\,\text{CR}\ , \tag{3.1}$$

where $\mathbf{x}_{\text{opt}}$ is the optimal solution that is assumed to be unique and CR is called the *convergence rate* of the algorithm. In evolution strategies, $\mathbf{x}_t$ designates the mean solution at iteration t and is generally considered to be the solution suggested by the algorithm as the currently best solution. We obtain almost sure convergence and say that an algorithm converges, almost surely, linearly when the following equality is satisfied almost surely:

$$\lim_{t\to\infty} \frac{1}{\lambda}\frac{1}{t}\left[\ln \frac{\|\mathbf{x}_t - \mathbf{x}_{\text{opt}}\|}{\|\mathbf{x}_0 - \mathbf{x}_{\text{opt}}\|}\right] = -\,\text{CR}\ , \tag{3.2}$$

where $\mathbf{x}_0$ represents the initial solution of the algorithm at iteration 0.

This linear convergence can be empirically investigated and is observed for certain step-size adaptation schemes (we remain in the context of evolution strategies) on several functions. However, formal proofs of such convergence remain relatively scarce and exist only for some particular step-size adaptation algorithms on some function classes. For example, sufficient conditions to obtain linear convergence on scale-invariant functions were established, using Markov Chain analysis, in [Auger and Hansen, 2013a] for a large class of evolution strategies. These conditions were verified for the (1+1)-ES with one-fifth success rule on positively homogeneous functions (a subclass of scale-invariant functions satisfying $\exists k \in \mathbb{N}, \forall \alpha > 0, f(\alpha\mathbf{x}) = \alpha^k f(\mathbf{x})$) in [Auger and Hansen, 2013b].

One important step-size adaptation scheme is one that produces a step-size proportional to the distance to the optimum of the current mean solution,

$$\sigma_t = \sigma^* \times \frac{\|\mathbf{x}_t - \mathbf{x}_{\text{opt}}\|}{d}\ , \tag{3.3}$$

where σ^* is constant. The resulting *artificial* algorithm achieves linear convergence on spherical functions, whiche are functions of the form $f(\mathbf{x}) = g(f^{\text{sphere}}(\mathbf{x} - \mathbf{x}_{\text{opt}}))$ with g strictly increasing, for finite dimensions when used with single parent strategies ((1+1)-ES and (1,λ)-ES) [Auger and Hansen, 2011]. The sphere function is defined as follows:

$$f^{\text{sphere}}(\mathbf{x}) = \sum_{i=1}^{d} ([\mathbf{x}]_i)^2 = \|\mathbf{x}\|^2\ , \tag{3.4}$$

where $[\mathbf{x}]_i$ designates the i^{th} coordinate of $\mathbf{x}$ and $\|\mathbf{x}\|$ the euclidean norm of $\mathbf{x}$. We use the notation $[\mathbf{x}]_i$ (instead of x_i) in this chapter for convenience. This algorithm is an artificial one since the distance to the optimum is supposed unknown to the algorithm in a black-box setting.

Furthermore, linear convergence is the best convergence speed a single parent ES can achieve on any function (Theorem 10.17 of [Auger and Hansen, 2011]). The resulting convergence rate (CR in (3.1)) using distance to the optimum proportional step-size (3.3)

is *optimal* (maximal) on the spherical functions (Theorem 10.20 of [Auger and Hansen, 2011]), for some value of σ^*. This sets an upper bound on the convergence rate of any step-size adaptation method.

Regarding the multi-parent non-elitist recombination based evolution strategies ($(\mu/\mu_w, \lambda)$-ES), a step-size which is proportional to the distance to the optimum (3.3) achieves, also, linear convergence on finite dimension spherical functions [Auger et al., 2011]. In addition, and due to its scale-invariance, (3.1) is satisfied not only for the limit on t but for any t, i.e.,

$$\frac{1}{\lambda} E\left[\ln \frac{\|\mathbf{x}_{t+1} - \mathbf{x}_{\text{opt}}\|}{\|\mathbf{x}_t - \mathbf{x}_{\text{opt}}\|}\right] = -\,\text{CR} \;\text{ for all } t \in \mathbb{N} \;. \tag{3.5}$$

Optimality of this convergence rate is still to be proven but we assume it to be true.

We develop the Left-hand Side (LHS) of (3.5) and expend the expression of $\mathbf{x}_{t+1}$ (2.12) knowing that the offspring are sampled from our parameterized normal distribution (2.10) with $\mathbf{C}_t = \mathbf{I}_d$, while assuming, without loss of generality (WLOG), that the optimum is at zero ($\mathbf{x}_{\text{opt}} = \mathbf{0}_d$), and obtain:

$$\begin{aligned} -\,\text{CR} &= \frac{1}{\lambda} E\left[\ln \frac{\|\mathbf{x}_{t+1}\|}{\|\mathbf{x}_t\|}\right] \\ &= \frac{1}{\lambda} E\left[\ln \frac{\|\sum_{i=1}^{\mu} w_i \mathbf{x}_t^{i:\lambda}\|}{\|\mathbf{x}_t\|}\right] \\ &= \frac{1}{\lambda} E\left[\ln \frac{\|\mathbf{x}_t + \sigma_t \sum_{i=1}^{\mu} w_i \mathbf{N}^{i:\lambda}\|}{\|\mathbf{x}_t\|}\right] \\ &= \frac{1}{\lambda} E\left[\ln \|\frac{\mathbf{x}_t}{\|\mathbf{x}_t\|} + \frac{\sigma_t}{\|\mathbf{x}_t\|} \sum_{i=1}^{\mu} w_i \mathbf{N}^{i:\lambda}\|\right] \;, \end{aligned} \tag{3.6}$$

where $(\mathbf{N}^{i:\lambda})_{1 \le i \le \lambda}$ are the standard independent and identically distributed multivariate normal vectors used to generate the offspring (the realizations of the standard multivariate normal distribution in (2.13)):

$$\mathbf{x}_t^i = \mathbf{x}_t + \sigma_t \mathbf{N}^i \;, \tag{3.7}$$

and ranked such that

$$\|\mathbf{x}_t + \sigma_t \mathbf{N}^{1:\lambda}\| \le \|\mathbf{x}_t + \sigma_t \mathbf{N}^{2:\lambda}\| \le \cdots \le \|\mathbf{x}_t + \sigma_t \mathbf{N}^{\lambda:\lambda}\| \;. \tag{3.8}$$

In other words, $\mathbf{N}^{i:\lambda}$, with $1 \le i \le \lambda$ corresponds to the realization of the multi-normal distribution used for $\mathbf{x}_t^{i:\lambda}$ that satisfies (2.11), $\mathbf{x}_t^{i:\lambda} = \mathbf{x}_t + \sigma_t \mathbf{N}^{i:\lambda}$ with the sphere function as the fitness function.

We have $\dfrac{\sigma_t}{\|\mathbf{x}_t\|} = \dfrac{\sigma^*}{d}$ (3.3) and we set $\dfrac{\mathbf{x}_t}{\|\mathbf{x}_t\|} = \mathbf{e}_1$, where $\mathbf{e}_1$ is the first canonical basis vector $(1, 0, \ldots, 0)$, without loss of generality due to the isotropy of both the spherical functions and the multi-variate normal distribution used to generate the offspring. Then we have:

$$-\,\text{CR} = \frac{1}{\lambda} E\left[\ln \|\mathbf{e}_1 + \frac{\sigma^*}{d} \sum_{i=1}^{\mu} w_i \mathbf{N}^{i:\lambda}\|\right] \;. \tag{3.9}$$

Taking the limit case of spherical functions with dimension tending to infinity ($d \to \infty$), the asymptotic convergence rate satisfies [Arnold, 2006]:

$$\lim_{d\to\infty} d \times \mathrm{CR} = -\frac{\sigma^*}{\lambda}\left(\sum_{i=1}^{\mu} w_i E[\mathcal{N}^{i:\lambda}] + \frac{1}{2}\sigma^* \sum_{i=1}^{\mu} w_i^2\right) \ , \tag{3.10}$$

where $\mathcal{N}^{i:\lambda}$, $1 \leq i \leq \lambda$ is the i^{th} order statistic (smallest individual) in a population or λ standard-normal distributed samples. Equation (3.10) is important as it allows us to identify the parameter values that maximize the convergence rate in the asymptotic case.

For starters, we can derive with respect to σ^* and find the optimal value of this parameter:

$$\sigma^*_{\text{opt}} = -\frac{\sum_{i=1}^{\mu} w_i E[\mathcal{N}^{i:\lambda}]}{\sum_{i=1}^{\mu} w_i^2} = -\mu_{\text{eff}} \sum_{i=1}^{\mu} w_i E[\mathcal{N}^{i:\lambda}] \ , \tag{3.11}$$

where $\mu_{\text{eff}} = (\sum_{i=1}^{\mu} w_i^2)^{-1}$ is the *variance effective selection mass*. We refer to the corresponding step-size, σ^*_{opt}, as the *optimal step-size*. We extend this appellation to the case of finite dimension and refer to the step-size that maximizes the convergence rate as optimal step-size.

Then, the optimal weights are obtained by plugging, in (3.10), the value of σ^*_{opt} from (3.11) and deriving with respect to the weights w_i. The resulting optimal weights are the negatives of the order statistics of a standard normal distribution [Arnold, 2005] with λ individuals: $-E[\mathcal{N}^{i:\lambda}]$, $i = 1, \ldots, \mu$. Dividing the weights by a constant factor α leads to an α times larger optimal step-size, but the resulting weights retain their optimal status [Ait Elhara et al., 2013]. Thus, we can normalize the optimal weights to have them summing to 1:

$$w_i^{\text{opt}} = \frac{E\left[\mathcal{N}^{i:\lambda}\right]}{\sum_{k=1}^{\mu} E\left[\mathcal{N}^{k:\lambda}\right]}, \ i = 1, \ldots, \mu \ . \tag{3.12}$$

When $\mu \neq \lambda/2$, we choose to set the weights to the values that would be obtained, given μ, if $\lambda = 2\mu$, i.e., given μ, we use:

$$w_{i:2\mu}^{\text{opt}} = \frac{E\left[\mathcal{N}^{i:2\mu}\right]}{\sum_{k=1}^{\mu} E\left[\mathcal{N}^{k:2\mu}\right]}, \ i = 1, \ldots, \mu \ . \tag{3.13}$$

This guarantees that all weights remain positive and are normalized to sum to 1. We retain these weights as the default weights in this chapter (unless otherwise stated) and simply refer to them, to lighten the notation, by w_i (2μ will be deduced from the context). Note that in Chapter 4, we use the default weights of CMA-ES (see Table 4.3).

3.3 The Median Success Rule Working-Principle

3.3.1 Motivations

The Median Success Rule (MSR) described in this chapter was mainly aimed to be an alternative to the Cumulative Step-size Adaptation (CSA) for multi-parent non-elitist

evolution strategies in large dimensions and/or for large values of μ. In fact, the reliance of CSA on the adaptation of the covariance matrix, in order to adapt the isotropic evolution path, leads to, at best, quadratic time and space complexities.

Success-based methods are, computationally (and comparatively) cheap, with a complexity that is generally independent of the search-space dimension, or has a scaling in it that is shadowed by other complexities (algorithms are generally not expected to have less than linear complexities). For example, and as we will see, the complexity of MSR depends on the population size λ, which in turn is set by default to depend on the problem dimension d. However, since the population size is chosen to be logarithmic in the dimension, MSR does not change the overall complexity of the algorithm. In addition, success-based methods rely on less restrictive properties; the main assumption being the ability to use the success probability, or a measure of it, as a reliable signal for step-size adaptation.

3.3.2 Preliminaries

In a (1+1)-ES, we do not have much of a choice on how to define success, at least not over a single iteration: *the iteration is successful if, and only if, the offspring has an equal or better fitness than the parent*, i.e., $f(\mathbf{x}_t^{1:1}) \leq f(\mathbf{x}_t)$, where the equality is considered a success in order to prevent stagnation on plateaus. As a result, and since we have single parent single offspring elitist selection, a successful iteration means the next parent is no worse than the current one ($f(\mathbf{x}_{t+1}) \leq f(\mathbf{x}_t)$, although one does not necessarily need to have the same success definition for the update of the mean $\mathbf{x}_t$ as for the update of the step-size).Once the notion of success is defined, we investigate the success probability that is observed when desirable/optimal conditions (usually, convergence rate or progress rate of the algorithm) are met on some usual functions such as the sphere function. This procedure resulted in an optimal success probability $p_{\text{opt}} \approx 1/5$ for the (1+1)-ES [Rechenberg, 1994].

In order to apply the same approach for $(\mu/\mu_w, \lambda)$-ES, a clear notion of success (and thus of success probability) needs to be defined. On a $(\mu/\mu_w, \lambda)$-ES, we can think of several success definitions over two successive iterations. We consider two iterations since we are in a non-elitist configuration where only the offspring generated at time t will be used to defined the next mean solution $\mathbf{x}_{t+1}$. For example, we can consider the improvement of the mean solution $\mathbf{x}_t$ ($f(\mathbf{x}_{t+1}) \leq f(\mathbf{x}_t)$) as a straightforward generalization of the $1/5^{\text{th}}$ success rule; the improvement of the i^{th} best individual ($f(\mathbf{x}_{t+1}^{i:\lambda}) \leq f(\mathbf{x}_t^{i:\lambda})$), with i a parameter to set; comparison of the i^{th} best individual at iteration $t+1$ with the mean at t ($f(\mathbf{x}_{t+1}^{i:\lambda}) \leq f(\mathbf{x}_t)$)...

Each definition might produce a different optimal success probability p_{opt} anywhere between 0 and 1. However, and in order to have the most accurate estimation of the success probability, p_{opt} must be close to 1/2 since the algorithm will be estimating the parameter of a Bernoulli distribution. Comparing the *same individual* ($\mathbf{x}_{t+1}$ versus $\mathbf{x}_t$, $\mathbf{x}_{t+1}^{1:\lambda}$ versus $\mathbf{x}_t^{1:\lambda}$...) at successive iterations gives p_{opt} closer to 0 and 1 than to 1/2 for large enough μ since with an optimal step-size, the overall quality of the population is

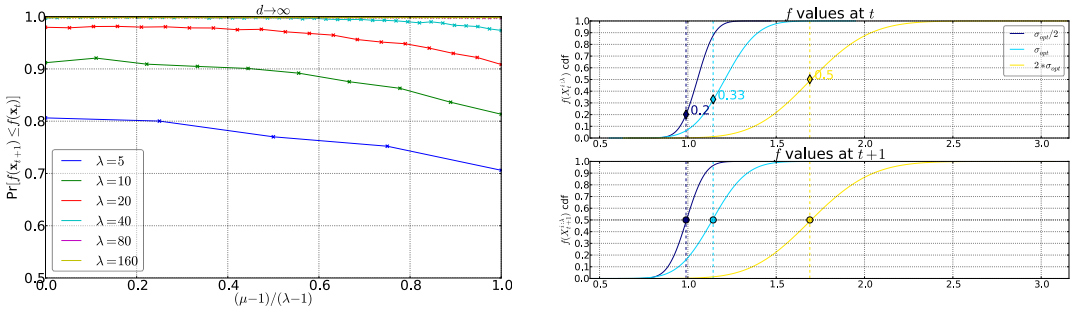

Figure 3.1: **Left:** Estimated target success probability when considering the improvement of the mean solution ($f(\mathbf{x}_{t+1})$ vs $f(\mathbf{x}_t)$) for different values of μ on the asymptotic sphere function ($d \to \infty$). A $(\mu/\mu_w, \lambda)$-ES is considered with optimal weights (see (3.13)), $\mathbf{x}_t = \mathbf{e}_1$ and the step-size is set to its optimal value for each configuration (3.11). The simulated derived formula is shown in (3.14). A total of 10^4 samples is used to estimated each probability and all possible values of μ are considered ($1 \leq \mu \leq \lambda$). **Right:** Empirical Cumulative Distribution of offspring fitness at two successive iterations (t and $t+1$) on a sphere function of dimension 20 with a population size of $\lambda = 10$. A total of 10^5 pairs of populations at successive iterations are used for each value of the step-size. The mean solution of the first iteration $\mathbf{x}_t$ is set to $\mathbf{e}_1$ so $f(\mathbf{x}_t) = 1$. We see the objective function values on the x-axis with their corresponding empirical cumulative distribution values considering all the offspring, over the different trials, at iterations t and $t+1$. The "$\diamondsuit$"s show to which percentile of the population at t the median at $t+1$ (designated by "$\circ$"s) must be compared in order to have a success probability of 1/2 (aligned in the x-axis). The graphs are shown for three multiples of the optimal step-size, $\sigma_{\text{opt}}/2$, σ_{opt} and $2\sigma_{\text{opt}}$.

expected to improve over the iterations in order for it to converge to the optimum.

For example, in the left plot of Figure 3.1, we show the target success probabilities of a $(\mu/\mu_w, \lambda)$-ES operating on the asymptotic sphere function ($d \to \infty$) when defining success as being $f(\mathbf{x}_{t+1}) \leq f(\mathbf{x}_t)$. We define the target success probability to be the success probability that is observed when the step-size is optimal (maximal possible convergence rate). We use the optimal step-size as defined in (3.11) and derive the formula of the success probability when optimal weights (3.13) are used:

$$\Pr\left(f(\mathbf{x}_{t+1}) \leq f(\mathbf{x}_t)\right) = \Pr\left(2\sum_{i=1}^{\mu} w_i \mathcal{N}^{i:\lambda} \leq \sum_{i=1}^{\mu} w_i E[\mathcal{N}^{i:\lambda}]\right) \quad , \tag{3.14}$$

where $(\mathcal{N}_t^{i:\lambda})_{1 \leq i \leq \lambda}$ the order statistics of a normal standard distribution with a pool of λ samples.

We see that most of the target success probabilities in the asymptotic case are closer to 1 than to 1/2. As to be expected from (3.14), the target success probability is positively correlated with the population size λ.

3.3.3 The Definition of Median Success

We are interested in finding a definition of success that would result in optimal success probabilities close to 1/2. To do so, we consider the populations at two successive iterations $(\mathbf{x}_t^{i:\lambda})_{1\leq i\leq\lambda}$ and $(\mathbf{x}_{t+1}^{i:\lambda})_{1\leq i\leq\lambda}$ and define a notion of success that is parametized by an index j_σ. Then we investigate the values of j_σ that produce the desired target success probabilities on the asymptotic spherical functions where the notion of optimal step-size is well defined.

Given j_σ, an iteration is successful if and only if *the median offspring at $t+1$ is at least as good as the j_σ^{th} best individual at t*. Then, we need to find j_σ that satisfies:

$$\Pr\left(f(\mathbf{x}_{t+1}^{m:\lambda}) \leq f(\mathbf{x}_t^{j_\sigma:\lambda})\right) \approx 1/2 \ , \tag{3.15}$$

where $m:\lambda$, and when there is no ambiguity just m, designates, the index of the median offspring (function-value wise). Note that $\mathbf{x}_{t+1}^{m:\lambda}$ is well defined only when λ is odd but we will see in the implementation section (Section 3.3.4) how we deal with even values of λ. We call the probability in the left hand side of (3.15) *the Median Success Probability*.

We want our comparison index j_σ, to allow a rapid convergence to the optimal step-size (when it is defined), so we choose it to be the value that, when *optimal conditions* are met, satisfies (3.15). As seen in Section 3.2, the optimal conditions on the sphere function when $d\to\infty$ are well known (see for example (3.11)). Then, we can set j_σ to the value that, on the asymptotic sphere function and when the step-size is optimal, the median success probability is 1/2. We will see later in this chapter (Section 3.4.2) how we can chose the value of j_σ when the dimension is finite and on functions other than the sphere.

The success probability introduced in (3.15) is usually different from 1/2 for all possible integer values of j_σ, thus we allow j_σ to be real valued. To determine j_σ given the ranked offspring of two successive iterations $(\mathbf{x}_t^{i:\lambda})_{1\leq i\leq\lambda}$ and $(\mathbf{x}_{t+1}^{i:\lambda})_{1\leq i\leq\lambda}$, we identify the index, j_σ^- with a median success probability, p_s^-, closest to 1/2 but smaller than 1/2. We set, $j_\sigma^+ = j_\sigma^- + 1$ with its corresponding median success probability p_s^+. Then, j_σ^+ has the closest median success probability to 1/2 that is larger than 1/2.

In order to estimate the value of j_σ, we linearly interpolate it to the value that would give a 1/2 median success probability. The line passing by the points (j_σ^-, p_s^-) and (j_σ^+, p_s^+) has as equation

$$p = (p_s^+ - p_s^-)j + p_s^- - (p_s^+ - p_s^-)j_\sigma^- \ . \tag{3.16}$$

To get the interpolated value of j_σ, we set $p = 1/2$ in (3.16) and obtain

$$j_\sigma = j_\sigma^- + \frac{1/2 - p_s^-}{p_s^+ - p_s^-} \ . \tag{3.17}$$

This amounts to giving the indexes j_σ^+ and j_σ^- a weight proportional to the distance of the success probability obtained using the other index (p_s^- and p_s^+ respectively) to 1/2:

$$j_\sigma = \frac{1}{p_s^+ - p_s^-}\left[\left(p_s^+ - 1/2\right) j_\sigma^- + \left(1/2 - p_s^-\right) j_\sigma^+\right] \ , \tag{3.18}$$

where $1/(p_s^+ - p_s^-)$ is a normalization allowing the weights to sum up to 1.

Assuming for the moment that we have identified a real number j_σ (the process of identifying j_σ will be explained later), we need, to be able to implement the algorithm, to give a meaning to the random vector $\mathbf{x}_t^{j_\sigma:\lambda}$ that is coming into play in (3.15). Since we allow j_σ to be non-integer, $\mathbf{x}_t^{j_\sigma:\lambda}$ is not necessarily part of the population.

To do so, we introduce a random variable $\tilde{j_\sigma}$ that takes its values in $\{j_\sigma^-, j_\sigma^+\}$. The probabilities of $\tilde{j_\sigma}$ taking each value are chosen such that $E[\tilde{j_\sigma}] = j_\sigma$:

$$\tilde{j_\sigma} = j_\sigma^- \mathbb{1}_{u \leq 1-\{j_\sigma\}} + j_\sigma^+ \mathbb{1}_{u \leq \{j_\sigma\}} \ , \tag{3.19}$$

were $\{j_\sigma\}$ designates the fractional part of j_σ ($j_\sigma - \lfloor j_\sigma \rfloor$), $\mathbb{1}$ the indicator function and u a random variable uniformly distributed in $[0, 1]$. These notations will remain relevant in the few following equations.

We can now define a generalized version of $\mathbf{x}_t^{j_\sigma:\lambda}$ for a real valued index j_σ given the offspring at iteration t, $(\mathbf{x}_t^{i:\lambda})_{1 \leq i \leq \lambda}$, as the random variable $\mathbf{x}_t^{\tilde{j_\sigma}:\lambda}$. Hence, $\mathbf{x}_t^{j_\sigma:\lambda}$ satisfies

$$\mathbf{x}_t^{j_\sigma:\lambda} = \mathbf{x}_t^{j_\sigma^-:\lambda} \mathbb{1}_{u \leq 1-\{j_\sigma\}} + \mathbf{x}_t^{j_\sigma^+:\lambda} \mathbb{1}_{u \leq \{j_\sigma\}} \ . \tag{3.20}$$

Using this definition of $\mathbf{x}_t^{j_\sigma:\lambda}$ (3.20), the median success probability reads:

$$\begin{aligned} \Pr\left(f(\mathbf{x}_{t+1}^{m:\lambda}) \leq f(\mathbf{x}_t^{j_\sigma:\lambda})\right) =& (1 - \{j_\sigma\}) \times \Pr\left(f(\mathbf{x}_{t+1}^{m:\lambda}) \leq f(\mathbf{x}_t^{j_\sigma^-:\lambda})\right) \\ &+ \{j_\sigma\} \times \Pr\left(f(\mathbf{x}_{t+1}^{m:\lambda}) \leq f(\mathbf{x}_t^{j_\sigma^+:\lambda})\right) \ . \end{aligned} \tag{3.21}$$

In practice, we are interested in the estimation of the success probability in (3.21) over a single iteration given offspring $(\mathbf{x}_t^{i:\lambda})_{1 \leq i \leq \lambda}$ and $(\mathbf{x}_{t+1}^{i:\lambda})_{1 \leq i \leq \lambda}$ where the comparisons yield binary results. The empirical estimation of the probability in (3.21), that we note $\hat{\Pr}_t^{\text{MSR}}(m, j_\sigma)$ and call *empirical success probability*, is defined as follows:

$$\hat{\Pr}_t^{\text{MSR}}(m, j_\sigma) := (1 - \{j_\sigma\}) \times \mathbb{1}_{f(\mathbf{x}_{t+1}^{m:\lambda}) \leq f(\mathbf{x}_t^{j_\sigma^-:\lambda})} + \{j_\sigma\} \times \mathbb{1}_{f(\mathbf{x}_{t+1}^{m:\lambda}) \leq f(\mathbf{x}_t^{j_\sigma^+:\lambda})} \ . \tag{3.22}$$

Note that an approach similar to what is done in equations (3.19) and (3.20) can also be applied to $\mathbf{x}_{t+1}^{m:\lambda}$ when λ is even (m is not an integer). However, the implementation of the MSR (see Section 3.3.4) does not use the median offspring directly. Thus, we do not explicitly need to compute or estimate $\mathbf{x}_{t+1}^{m:\lambda}$ when it is not part of the offspring.

Looking into the asymptotic sphere function, we find that the median success probability is negatively correlated with the step-size (see (3.57) in Section 3.4.2.2). This correlation makes it possible to use the median success probability as a signal that guides the step-size towards its optimal value for a well set value of j_σ. The optimal step-size here refers to the step-size that results in the largest possible convergence rate. The way we defined j_σ, a well set /optimal value is one that results in a *targeted* median success probability of $1/2$ when the step-size is optimal.

The right plot of Figure 3.1 shows an example on how the ideal comparison index (or comparison percentile) j_σ depends on the step-size on a finite dimension ($d = 20$) sphere

function. It shows the empirical cumulative distribution of the fitness of the offspring at two successive iterations for different values of the step-size. The offspring are obtained from a couple of iterations of a $(\mu/\mu_w, \lambda)$-ES.

The negative correlation between step-size and median success probability means that the bigger the step-size, the larger the comparison index producing a median success probability of $1/2$. In the example of Figure 3.1, one would set the comparison index to the 33%-tile ($\lambda/3$) as this corresponds to an empirical median success probability (3.22) of $1/2$ on the estimated optimal step-size.

3.3.4 Implementation of the Median Success Rule

Limiting the estimation of success to only that of the median would result, similarly to the single parent single offspring scenario, in a binary signal. However, being in a configuration where the population size is larger than 2, we are expected to have a better suited mechanism where more of the information at one's disposal is used to estimate the drift from the optimal step-size at iteration t.

In order to have a more accurate estimation of the success probability (more precisely, of the distance to $1/2$ which is our optimal success probability), we *count* the number of successful offspring at $t+1$ when compared to the j_σ^{th} best individual at t. By doing so, and roughly speaking, a successful median would mean that at least half the offspring are successful (and an unsuccessful one otherwise). We, first, generalize (3.22) to define the *empirical success probability* of an individual $\mathbf{x}_{t+1}^{i:\lambda}$ (or an index i) given the offspring at iteration t and the comparison index j_σ:

$$\hat{\Pr}_t^{\text{MSR}}(i, j_\sigma) := (1 - \{j_\sigma\}) \times \mathbb{1}_{f(\mathbf{x}_{t+1}^{i:\lambda}) \leq f(\mathbf{x}_t^{j_\sigma^- :\lambda})} + \{j_\sigma\} \times \mathbb{1}_{f(\mathbf{x}_{t+1}^{i:\lambda}) \leq f(\mathbf{x}_t^{j_\sigma^+ :\lambda})} \ . \tag{3.23}$$

To *count* the number of successes we have, we sum these empirical success probabilities over all the offspring of iteration $t+1$:

$$K_t^{\text{succ}} = \sum_{i=1}^{\lambda} \hat{\Pr}_t^{\text{MSR}}(i, j_\sigma) \ . \tag{3.24}$$

We see that in (3.24), we do not need to estimate $\mathbf{x}_{t+1}^{m:\lambda}$ when λ is even. In fact, we sum over all integer indexes of iteration $t+1$, and if λ is even, the median is, simply, not taken into consideration in the computation of K_t^{succ} as it is not part of the offspring.

We see in the left plot of Figure 3.2 an example where the estimated empirical success probabilities, $\hat{\Pr}_t^{\text{MSR}}(i, j_\sigma)$ (3.23), are shown under each point. In the lower plot, K_t^{succ} is an integer ($K_t^{\text{succ}} = 2$) since all individuals at $t+1$ are either better than $\mathbf{x}_t^{j_\sigma^- :\lambda}$ or worse than $\mathbf{x}_t^{j_\sigma^+ :\lambda}$. However, in the upper plot, we have $\mathbf{x}_{t+1}^{4:\lambda}$ whose fitness lies between those of these two individuals ($\mathbf{x}_t^{j_\sigma^- :\lambda}$ and $\mathbf{x}_t^{j_\sigma^+ :\lambda}$). Then for this case, and following (3.23) and (3.24), we have $K_t^{\text{succ}} = 3 + \{j_\sigma\}$.

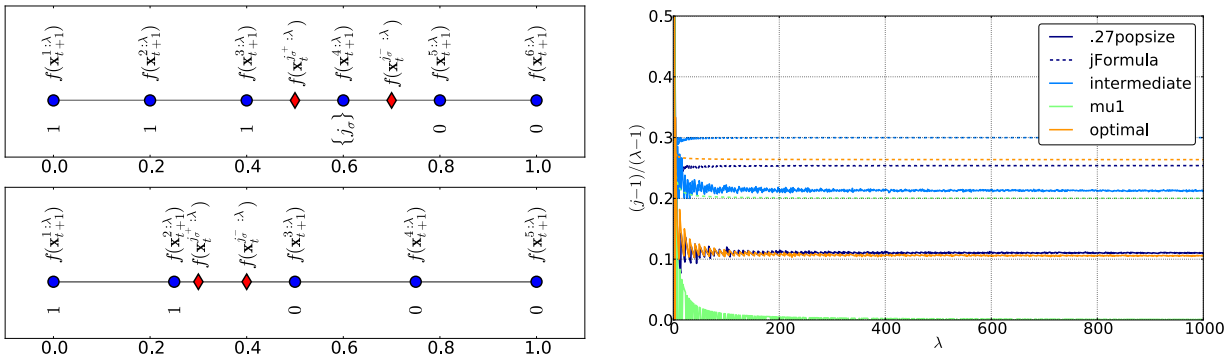

Figure 3.2: **Left:** Two examples showing how K_t^{succ} is computed. On the x-axes are shown the fitness of an offspring on a fictive objective function, f. The "•"s represent the offspring at iteration $t+1$ while "♦"s show the position, on the same axis, of the two offspring of ranks j_σ^- and j_σ^+ that are used to compute K_t^{succ} (3.24). The numbers under each data point show the empirical success probabilities $\hat{\Pr}_t^{\text{MSR}}(i, j_\sigma)$ (3.23) for each corresponding point. **Right:** Smallest value of the comparison percentile yielding a Median Success Probability larger that 1/2 on the linear function (3.39). The success probabilities, (3.44), are empirically estimated over 10^3 realizations. The dotted lines illustrate a lower bound on the default values of j_σ (3.62) established in [Ait Elhara et al., 2013] for which we take $d \to \infty$ for each weighting scheme while the solid lines are for $j_\sigma = 0.27\lambda$. Blue is for intermediate (equal) weights; Orange is for optimal weight (3.13) and Green is for the single parent case $\mu = 1$, $w_i = \mathbb{1}_{i=1}$.

We then normalize K_t^{succ} such that we obtain a signal at iteration t, z_t, that is positive whenever the median is successful (roughly half the offspring are successful):

$$z_t = h\left(\frac{2}{\lambda}\left(K_t^{\text{succ}} - \frac{\lambda}{2}\right)\right) \quad , \tag{3.25}$$

where h is a monotonous function. We set h to the identity function in this chapter; using a sign function (as explained in the beginning of this section) gave inferior results. This normalization makes $z_t \in [-\frac{\lambda+1}{\lambda}, \frac{\lambda-1}{\lambda}]$ with a successful median pushing towards (since we use a learning rate, c_σ, the change in the step-size does not necessarily follow the sign of z_t) an increase to the step-size with $z_t \geq 0$ and an unsuccessful one to a decrease. The success or not of the median, then, marks the frontier between a successful iteration and and unsuccessful one. In our context, we say that an offspring $\mathbf{x}_{t+1}^{i:\lambda}$ is successful when $\hat{\Pr}_t^{\text{MSR}}(i, j_\sigma) = 1$.

When λ is odd, it is straightforward: the median offspring is $\mathbf{x}_{t+1}^{(\lambda+1)/2:\lambda}$, $K_t^{\text{succ}} \geq \frac{\lambda+1}{2}$ if $\hat{\Pr}_t^{\text{MSR}}(m, j_\sigma) = 1$; i.e., the median is successful. The lower-left plot of Figure 3.2 is an example of an unsuccessful median.

When λ is even, the median is not part of the offspring since $\frac{\lambda+1}{2} \notin \mathbb{N}$. We look into

$\mathbf{x}_{t+1}^{\lceil(\lambda+1)/2\rceil:\lambda}$ and find that:

$$\begin{aligned} \left(\hat{\Pr}_t^{\text{MSR}}(\lceil(\lambda+1)/2\rceil, j_\sigma) = 1\right) &\implies \left(K_t^{\text{succ}} \geq \frac{\lambda}{2}\right) \ , \\ \left(\hat{\Pr}_t^{\text{MSR}}(\lceil(\lambda+1)/2\rceil, j_\sigma) = 0\right) &\implies \left(K_t^{\text{succ}} < \frac{\lambda}{2}\right) \ . \end{aligned} \tag{3.26}$$

For intermediate values ($0 < \hat{\Pr}_t^{\text{MSR}}(\lceil\frac{\lambda+1}{2}\rceil, j_\sigma) < 1$), the sign of z_t (same sign as $K_t^{\text{succ}} - \frac{\lambda}{2}$) depends mainly on $\{j_\sigma\}$. In the upper-left plot of Figure 3.2, $z_t \geq 0$ if, and only if, $\{j_\sigma\} \geq 1/2$.

The signal, z_t, is smoothed to take into consideration previous values by introducing a learning rate c_σ:

$$s_{t+1} = (1 - c_\sigma)s_t + c_\sigma z_t \ , \tag{3.27}$$

where $c_\sigma \in]0, 1]$ and $s_0 = 0$.

Finally, s_{t+1} is used to update the value of the step-size:

$$\sigma_{t+1} = \sigma_t \exp(\frac{s_{t+1}}{d_\sigma}) \ , \tag{3.28}$$

where d_σ is the damping parameter that dictates the maximal amount of change on the step-size in a single iteration (since s_t is bounded).

The step-size adaptation mechanism of MSR is described by equations (3.24) to (3.28), bar (3.26), and summed up in Algorithm 1.

Algorithm 1 The Median Success Rule (MSR)

j_σ : the comparison index; default value: see equation (3.62)
σ_t : step-size at previous iteration,
d_σ : damping parameter, $d_\sigma > 0$; default value: $d_\sigma = 2 - 2/d$
c_σ : learning rate of the signal, $0 < c_\sigma < 1$; default value: $c_\sigma = 0.3$
h: monotonous function; default choice: identity function,
f : objective function,
$(\mathbf{x}_t^{i:\lambda})_{1\leq i\leq\lambda}$: offspring at previous iteration, ordered by fitness values,
$(\mathbf{x}_{t+1}^{i:\lambda})_{1\leq i\leq\lambda}$: offspring at current iteration, ordered by fitness values,
s_t : last value of the step-size update coefficient.

1: $K_t^{\text{succ}} = \sum_{i=1}^{\lambda} \hat{\Pr}_t^{\text{MSR}}(i, j_\sigma)$ (see (3.23))
2: $z_t = h(\frac{2}{\lambda}(K_t^{\text{succ}} - \frac{\lambda+1}{2}))$
3: $s_{t+1} = (1 - c_\sigma)s_t + c_\sigma z_t$
4: $\sigma_{t+1} = \sigma_t \exp(\frac{s_{t+1}}{d_\sigma})$

3.4 Parameter Setting

Looking at Algorithm 1, we identify three main parameters for the Median Success Rule: (i) the learning rate c_σ, (ii) the comparison index j_σ and (iii) the damping parameter d_σ.

The next subsections will deal with setting the values of these three parameters. We set the monotonous function h to be the identity function through this chapter.

3.4.1 Learning Rate

The learning rate $c_\sigma \in]0, 1]$ allows to smooth, to some extent, the *step-size adaptation coefficient*, s_t, over time. It also allows to limit the maximal amplitude of change in this coefficient from an iteration to the next one. When $c_\sigma < 1$, s_t is not a 100% reliant on the information gathered from a single iteration, granting the strategy a certain level of robustness against extreme single occurrences, the information in question being z_t. Old values of s_t contribute to the current value, with a contribution that fades geometrically, with reason $(1 - c_\sigma)$, over time (see (3.27)). We choose to set the learning rate to the smallest value allowing two iterations to be enough for the coefficient s and the signal z to match signs when the signal is constant and its amplitude (absolute value) maximal, i.e.,

$$z_t = z_{t+1} = \pm 1 - 1/\lambda \implies s_{t+2} \times z_t \geq 0 \ . \tag{3.29}$$

This way, two successive observations of the same, maximal, signal are enough to make the step-size change in the direction suggested by this signal (increase if positive and decrease if negative). Meanwhile, a level of robustness against single extreme signals (all successful, or no success) is guaranteed by choosing the smallest value of c_σ. These extreme signals might be observed for example when the population sizes is too small.

Lemma 1. *Let us note $z_{\min}$ the smallest possible value of z_t, $z_{\min} = -\frac{\lambda+1}{\lambda}$. If $c_\sigma \geq 1 - \frac{\sqrt{2}}{2}\sqrt{(2 + z_{\min})}$ then*

$$z_t = z_{t+1} \in \{z_{\min}, z_{\min} + 2\} \implies s_{t+2} \times z_t \geq 0 \ . \tag{3.30}$$

Proof. Let $z_{\max}$ be, similarly, the largest values that z_t can take, i.e.,

$$z_{\max} = z_{\min} + 2 = \frac{\lambda - 1}{\lambda} \ . \tag{3.31}$$

Starting from s_{t+2}:

$$s_{t+2} = (1 - c_\sigma)s_{t+1} + c_\sigma z_{t+1} \ , \tag{3.32}$$

replacing s_{t+1} by its values from (3.27) and with $z_t = z_{t+1}$:

$$\begin{aligned} s_{t+2} &= (1 - c_\sigma)^2 s_t + (2 - c_\sigma)c_\sigma z_t \\ &= s_t - 2c_\sigma s_t + c_\sigma^2 s_t + 2c_\sigma z_t - c_\sigma^2 z_t \\ &= -(z_t - s_t)c_\sigma^2 + 2(z_t - s_t)c_\sigma + s_t \ . \end{aligned} \tag{3.33}$$

On the other hand, we have:

$$z_{\min} \leq s_0 \leq z_{\max} \implies z_{\min} \leq s_t \leq z_{\max}, \forall t \in \mathbb{N} \ . \tag{3.34}$$

This can easily be proved using mathematical induction.

Let us take the first *worst case scenario* (extreme unfavorable case for (3.30)). That is $z_t = z_{t+1} = z_{\max}$ and $s_t = z_{\min}$. This is the worst case scenario since we have the largest distance possible between the signal and the coefficient, $z_t - s_t = 2$, and more importantly the largest distance for them to match signs. By replacing in (3.33) we obtain:

$$s_{t+2} = -2c_\sigma^2 + 4c_\sigma + s_t \ . \tag{3.35}$$

This is a second degree equation with a positive discriminant. For s_{t+2} to be positive and match signs with z_t, and thus for (3.30) to be satisfied, we need:

$$1 - \frac{\sqrt{2}}{2}\sqrt{(2 + z_{\min})} \leq c_\sigma \leq 1 + \frac{\sqrt{2}}{2}\sqrt{(2 + z_{\min})} \ , \tag{3.36}$$

with the right-side inequality being irrelevant since $c_\sigma \leq 1$.

The other worst case scenario, $z_t = z_{\min}$ and $s_t = z_{\max}$, $z_t - s_t = -2$, and by using the same steps as the first worst case scenario, results in a formula similar to (3.36):

$$1 - \frac{\sqrt{2}}{2}\sqrt{(2 - z_{\max})} \leq c_\sigma \leq 1 + \frac{\sqrt{2}}{2}\sqrt{(2 - z_{\max})} \ . \tag{3.37}$$

Since in our case $z_{\min} \leq -z_{\max}$, the bound in (3.36) is tighter than the one in (3.37); thus $c_\sigma \geq 1 - \frac{\sqrt{2}}{2}\sqrt{(2 + z_{\min})}$ covers both cases. □

In practice, we choose to relax the values of $z_{\min}$ and $z_{\max}$ and set them to -1 and 1 respectively, ignoring the $-1/\lambda$. Then, the bound on c_σ becomes: $c_\sigma \geq 1 - \frac{\sqrt{2}}{2} \simeq 0.293$. We take the rounded value with a single significant digit: $c_\sigma = 0.3$.

3.4.2 Comparison Index

In this section, we are interested in setting the second parameter of MSR, namely the comparison index j_σ which is in the core of the proposed step-size adaptation strategy.

In order to asses the behavior of the Median Success Rule with different values of the comparison index, and then to choose the values of this index in practice, we start by considering three function settings, each setting having a different desired behavior of the step-size:

Linear Function: *Increasing step-size*; increasing the step-size ad infinitum.

Sphere Function: *Decreasing step-size*; distance to the optimum proportional step-size, see Section 3.2

Ridge Function: *Constant step-size*; when in the stationary state, the optimal step-size is constant.

A more systematic approach for designing and assessing step-size adaptation mechanisms can be found in [Hansen et al., 2014]. In [Hansen et al., 2014], five scenarios are suggested in order to assess step-size adaptation mechanisms. In addition to the (i) linear and (ii)

sphere functions that we consider here, the ridge function is replaced with a (iii) *stationary* sphere function that simulates a constant optimal step-size in a spherical scenario. The paper also assesses the step-size adaptation methods on (iv) random and flat functions, to see whether a given strategy is biased towards a particular behavior (for example, Self Adaptive step-size [Schwefel, 1995] has a step-size biased towards increase on random functions) and (v) an ill-conditioned function, typically the ellipsoid function.

Since on the linear and ridge functions the optimal solution is infinite, using *convergence rates* as a measure of performance is unpractical (the algorithm is supposed to diverge solution-wise). This is why we use, instead, the notion of progress that quantifies the evolution of the fitness of the mean solution in the desired direction (see, for example, (3.61)).

3.4.2.1 The Linear Function

The linear function is defined as follows:

$$f^{\text{linear}}(\mathbf{x}) = \sum_{i=1}^{d} [x]_i \ , \tag{3.38}$$

where $\mathbf{x} \in \mathbb{R}^d$, d (positive) is the search-space dimension and $[x]_i$ the i^{th} coordinate of $\mathbf{x}$. For the sake of simplicity, and with no loss of generality, we define the function on 1^{st} coordinate only; i.e.,

$$f^{\text{linear}}(\mathbf{x}) = [x]_1 \ , \tag{3.39}$$

making the function dimension-independent. We use (3.39) as the definition of the linear function through this chapter.

Then, the Median Success Probability on the linear function (left-hand side of (3.15)) reads:

$$\begin{aligned} \text{Pr}_{\text{linear}} &:= \text{Pr}\left(f^{\text{linear}}\left(\mathbf{x}_{t+1}^{m:\lambda}\right) \leq f^{\text{linear}}\left(\mathbf{x}_t^{j_\sigma:\lambda}\right)\right) \\ &= \text{Pr}\left(\left[\mathbf{x}_{t+1}^{m:\lambda}\right]_1 \leq \left[\mathbf{x}_t^{j_\sigma:\lambda}\right]_1\right) \\ &= \text{Pr}\left(\left[\mathbf{x}_{t+1} + \sigma_{t+1}\mathbf{N}_{t+1}^{m:\lambda}\right]_1 \leq \left[\mathbf{x}_t + \sigma_t \mathbf{N}_t^{j_\sigma:\lambda}\right]_1\right) \ , \end{aligned} \tag{3.40}$$

where $(\mathbf{N}_t^{i:\lambda})_{1\leq i\leq\lambda}$ are multi-variate standard-normal vectors ordered such that:

$$f^{\text{linear}}(\mathbf{x}_t + \sigma_t\mathbf{N}_t^{1:\lambda}) \leq f^{\text{linear}}(\mathbf{x}_t + \sigma_t\mathbf{N}_t^{2:\lambda}) \leq \cdots \leq f^{\text{linear}}(\mathbf{x}_t + \sigma_t\mathbf{N}_t^{\lambda:\lambda}) \ , \tag{3.41}$$

which derives, for $\sigma_t > 0$, into:

$$\left[\mathbf{N}_t^{1:\lambda}\right]_1 \leq \left[\mathbf{N}_t^{2:\lambda}\right]_1 \leq \cdots \leq \left[\mathbf{N}_t^{\lambda:\lambda}\right]_1 \ . \tag{3.42}$$

Since $\mathbf{N}_t^1 \sim \mathbf{N}(0, \mathbf{I})$ (the non-ordered offspring) and the linear function is only defined on the first coordinate (3.39), this order is the same as that of the one-dimensional standard

normal variables ($[\mathbf{N}_t^1]_1$, $[\mathbf{N}_t^2]_1$,..., $[\mathbf{N}_t^\lambda]_1$) that compose the first coordinate of each vector. Thus:

$$\begin{aligned}\mathrm{Pr}_{\text{linear}} &= \mathrm{Pr}\left(\left[\mathbf{x}_{t+1}\right]_1 + \sigma_{t+1}\mathcal{N}_{t+1}^{m:\lambda} \leq [\mathbf{x}_t]_1 + \sigma_t \mathcal{N}_t^{j_\sigma:\lambda}\right) \\ &= \mathrm{Pr}\left(\left[\mathbf{x}_t + \sigma_t \sum_{i=1}^{\mu} w_i \mathbf{N}_t^{i:\lambda}\right]_1 + \sigma_{t+1}\mathcal{N}_{t+1}^{m:\lambda} \leq [\mathbf{x}_t]_1 + \sigma_t \mathcal{N}_t^{j_\sigma:\lambda}\right) \\ &= \mathrm{Pr}\left(\sigma_t \sum_{i=1}^{\mu} w_i \mathcal{N}_t^{i:\lambda} + \sigma_{t+1}\mathcal{N}_{t+1}^{m:\lambda} \leq \sigma_t \mathcal{N}_t^{j_\sigma:\lambda}\right) \ ,\end{aligned} \tag{3.43}$$

where $\mathcal{N}_t^{i:\lambda}$ is defined the same as in (3.10), and different time subscripts designate independent samples. If we consider a constant step-size over two iterations ($\sigma_t = \sigma_{t+1}$), the resulting Median Success Probability on the linear function would be:

$$\boxed{\mathrm{Pr}_{\text{linear}} = \mathrm{Pr}\left(\mathcal{N}_{t+1}^{m:\lambda} \leq \mathcal{N}_t^{j_\sigma:\lambda} - \sum_{i=1}^{\mu} w_i \mathcal{N}_t^{i:\lambda}\right) \ .} \tag{3.44}$$

We can see that this success probability is independent, not only of the search space dimension d, but also of the the step-size σ_t. On the linear function, the desired behavior of a good step-size adaptation mechanism is to increase the step-size indefinitely since the optimum in at infinity. Thus, the linear success probability in (3.44) needs to be larger than 1/2 for a correct behavior of the algorithm since success probabilities larger than 1/2 push towards increasing the step-size (see Section 3.3.4).

We assess the performances of an algorithm over one iteration on the linear function using its *progress rate*. The progress rate is defined as the one iteration improvement of the fitness of the mean solution:

$$\varphi_t := f(\mathbf{x}_{t-1}) - f(\mathbf{x}_t), \ \forall t > 0 \ , \tag{3.45}$$

and reads in the case of the linear function:

$$\varphi_{\text{linear}} = \left[\mathbf{x}_{t-1}\right]_1 - [\mathbf{x}_t]_1 = -\sigma_{t-1} \sum_{i=1}^{\lambda} w_i \mathcal{N}^{i:\lambda} \ . \tag{3.46}$$

The larger the value of the progress rate, the better the algorithm is.

Assuming $w_1 \geq \cdots \geq w_\mu$ and $\mu < \lambda$ (both these conditions are generally satisfied), we see in (3.46) that the expected progress rate on the linear function for a $(\mu/\mu_w, \lambda)$-ES is positively correlated to the step-size. In the extreme case of equal weights (called *intermediate weights*) and $\mu = \lambda$, we obtain null expected progress regardless of the step-size:

$$\begin{aligned}E(\varphi_{\text{linear}}) &= -\sigma_{t-1}\frac{1}{\lambda} E\left(\sum_{i=1}^{\lambda} \mathcal{N}^{i:\lambda}\right) \\ &= -\sigma_{t-1}\frac{1}{\lambda} E\left(\sum_{i=1}^{\lambda} \mathcal{N}^{i}\right) \ \text{(no need for ordering since } \mu = \lambda) \\ &= 0 \ .\end{aligned} \tag{3.47}$$

Adding any selection mechanism, either by decreasing the number of parents μ, making the weights not all equal or both can only result in a better expected progress rate (since this leads to favoring better solutions). Thus, the generalization of the positive correlation between step-size and progress only requires that the weights are non-increasing.

As expected in (3.44), we see a straightforward correlation between the success probability and the value of the index to which the median is compared (the comparison index j_σ). In fact, $j_{\sigma 1} \geq j_{\sigma 2} \Leftrightarrow \mathrm{Pr}_{\mathrm{linear}}^{j_{\sigma 1}} \geq \mathrm{Pr}_{\mathrm{linear}}^{j_{\sigma 2}}$ as the higher the value of this index j_σ, the worse its objective value is in expectation; and thus, the better chances the median of the next iteration has to be successful.

Regarding the weights, what can be considered as a *worst case scenario*, when linear success probability is considered, is the case of *intermediate weights*. Here, the notion of worst case scenario is taken in the sense that, all other things being equal, this scenario gives the smallest linear success probability (3.44). An example of this can be seen in Figure 3.2 as the blue graph (corresponding to intermediate weighted recombination with $\mu = \lfloor \lambda/2 \rfloor$) is clearly above the orange graph which has the same μ but with optimal, strictly decreasing, weights (3.13) instead.

The right plot of Figure 3.2 depicts the smallest choices of the comparison index (or comparison percentile) allowing $\mathrm{Pr}_{\mathrm{linear}} \geq 1/2$ (increasing step-size) for different weighting schemes and different values of μ. We see that the formula for the comparison index (3.62), first suggested in [Ait Elhara et al., 2013] (the dashed lines), and for which a lower bound is used by taking $d \to \infty$, lies within the "good behavior" region for the considered weights and μ. For example, the configuration (optimal weights, $\mu = \lfloor \lambda/2 \rfloor$) results in a Median Success Probability over $1/2$ as long as $j \geq 0.11\lambda$ while the value used in the algorithm is around $0.27 \times \lambda$ for most values of λ. Thus, we can proceed in our investigation to other objective functions as the behavior on the linear function is shown to be *correct*.

3.4.2.2 The Sphere Function

The sphere function is widely studied when investigating convergence rates of evolution strategies. One reason for this is the presence of a solid theoretical model around the spherical functions (see Section 3.2) that provides the optimal behavior of an evolution strategy on them (maximal convergence rate, optimal step-size, optimal weights). Furthermore, the use of Coavariance Matrix Adaptation, with which MSR will be eventually coupled, is expected to result, locally and when possible, in a sphere-like/isotropic transformed problem (if the function is convex-quadratic, the inverse of the Hessian would be estimated). Thus, one purpose of the step-size adaptation rule would be to solve the *remaining* sphere function efficiently. We investigate the convergence rate of the algorithm on the sphere function (3.4) in order to find the comparison percentile/index allowing this convergence rate to be maximal on a spherical function. This will help set this parameter's default value.

We now derive the Median Success Probability (3.15) of a $(\mu/\mu_w, \lambda)$-ES on the sphere

function [Ait Elhara et al., 2013]:

$$\Pr_{\text{sphere}} := \Pr\left(f^{\text{sphere}}\left(\mathbf{x}_{t+1}^{m:\lambda}\right) \leq f^{\text{sphere}}\left(\mathbf{x}_{t}^{j_\sigma:\lambda}\right)\right) \quad . \tag{3.48}$$

Similarly to Section 3.4.2.1, we consider that the step-size is constant in the two successive iterations ($\sigma_{t+1} = \sigma_t = \sigma$):

$$\begin{aligned} \Pr_{\text{sphere}} &= \Pr\left(\|\mathbf{x}_{t+1}^{m:\lambda}\|^2 \leq \|\mathbf{x}_{t}^{j_\sigma:\lambda}\|^2\right) \\ &= \Pr\left(\|\mathbf{x}_{t+1} + \sigma\mathbf{N}_{t+1}^{m:\lambda}\|^2 \leq \|\mathbf{x}_t + \sigma\mathbf{N}_t^{j_\sigma:\lambda}\|^2\right) \quad , \end{aligned} \tag{3.49}$$

where the $(\mathbf{N}_t^{i:\lambda})_{1\leq i\leq\lambda}$ (and similarly $(\mathbf{N}_{t+1}^{i:\lambda})_{1\leq i\leq\lambda}$) are ordered, each independently, following the fitness of their corresponding offspring; that is:

$$f^{\text{sphere}}(\mathbf{x}_t + \sigma\mathbf{N}_t^{1:\lambda}) \leq \cdots \leq f^{\text{sphere}}(\mathbf{x}_t + \sigma\mathbf{N}_t^{\lambda:\lambda}) \quad , \tag{3.50}$$

and

$$f^{\text{sphere}}(\mathbf{x}_{t+1} + \sigma\mathbf{N}_{t+1}^{1:\lambda}) \leq \cdots \leq f^{\text{sphere}}(\mathbf{x}_{t+1} + \sigma\mathbf{N}_{t+1}^{\lambda:\lambda}) \quad . \tag{3.51}$$

Then,

$$\Pr_{\text{sphere}} = \Pr\left(\|\mathbf{x}_t + \sigma\sum_{i=1}^{\mu} w_i\mathbf{N}_t^{i:\lambda} + \sigma\mathbf{N}_{t+1}^{m:\lambda}\|^2 \leq \|\mathbf{x}_t + \sigma\mathbf{N}_t^{j_\sigma:\lambda}\|^2\right) \quad . \tag{3.52}$$

We set $\sigma = \|\mathbf{x}_t\|\dfrac{\sigma^*}{d}$, which leads to a decomposition that suggests a distance to the optimum proportional step-size (see Section 3.2) and algorithms trying to find the optimal multiplying constant σ^*_{opt} (a change of variable that is transparent to the algorithm). In addition, and thanks to the isotropy of the standard multivariate normal distribution and of the sphere function, we can replace the mean solution $\mathbf{x}_t$ with a vector of the same length without changing the success probability; in our case we choose to set, without loss of generality, $\dfrac{\mathbf{x}_t}{\|\mathbf{x}_t\|} = \mathbf{e}_1$. Replacing then simplifying by $\|\mathbf{x}_t\|$:

$$\Pr_{\text{sphere}} = \Pr\left(\|\mathbf{x}_t + \|\mathbf{x}_t\|\frac{\sigma^*}{d}\sum_{i=1}^{\mu} w_i\mathbf{N}_t^{i:\lambda} + \|\mathbf{x}_t\|\frac{\sigma^*}{d}\mathbf{N}_{t+1}^{m:\lambda}\|^2 \leq \|\mathbf{x}_t + \|\mathbf{x}_t\|\frac{\sigma^*}{d}\mathbf{N}_t^{j_\sigma:\lambda}\|^2\right) \quad . \tag{3.53}$$

Finally,

$$\Pr_{\text{sphere}} = \Pr\left(\|\mathbf{e}_1 + \frac{\sigma^*}{d}\sum_{i=1}^{\mu} w_i\mathbf{N}_t^{i:\lambda} + \frac{\sigma^*}{d}\mathbf{N}_{t+1}^{m:\lambda}\|^2 \leq \|\mathbf{e}_1 + \frac{\sigma^*}{d}\mathbf{N}_t^{j_\sigma:\lambda}\|^2\right) \quad . \tag{3.54}$$

3.4.2.2.1 Asymptotic case We now consider the asymptotic case of $d \to \infty$:

$$\lim_{d\to\infty} \Pr_{\text{sphere}} = \lim_{d\to\infty} \Pr\left(\|\mathbf{e}_1 + \frac{\sigma^*}{d}\sum_{i=1}^{\mu} w_i\mathbf{N}_t^{i:\lambda} + \frac{\sigma^*}{d}\mathbf{N}_{t+1}^{m:\lambda}\|^2 \leq \|\mathbf{e}_1 + \frac{\sigma^*}{d}\mathbf{N}_t^{j_\sigma:\lambda}\|^2\right) \quad . \tag{3.55}$$

By developing the squares and simplifying we find:

$$\lim_{d\to\infty} \Pr\left(2\left[\sum_{i=1}^{\mu} w_i \mathbf{N}_t^{i:\lambda} + \mathbf{N}_{t+1}^{m:\lambda}\right]_1 + \frac{\sigma^*}{d}\|\sum_{i=1}^{\mu} w_i \mathbf{N}_t^{i:\lambda} + \mathbf{N}_{t+1}^{m:\lambda}\|^2 \right. \\ \left. \leq 2\left[\mathbf{N}_t^{j_\sigma:\lambda}\right]_1 + \frac{\sigma^*}{d}\|\mathbf{N}_t^{j_\sigma:\lambda}\|^2\right) \quad . \tag{3.56}$$

The asymptotic median success probability on the sphere function is derived in Proposition 5 of [Ait Elhara et al., 2013]:

$$\boxed{\lim_{d\to\infty} \Pr_{\text{sphere}} = \Pr\left(\mathcal{N}_{t+1}^{m:\lambda} \leq \mathcal{N}_t^{j_\sigma:\lambda} - \sum_{i=1}^{\mu} w_i \mathcal{N}_t^{i:\lambda} - \frac{1}{2}\sigma^* \sum_{i=1}^{\mu} w_i^2\right) \quad ,} \tag{3.57}$$

where $\mathcal{N}_t^{i:\lambda}$ (and similarly $\mathcal{N}_{t+1}^{i:\lambda}$) is the i^{th} order statistic of a standard normal distribution with λ samples. The proof can be found in the Appendix of [Ait Elhara et al., 2013] where the authors assume that

$$2\left[\sum_{i=1}^{\mu} w_i \mathbf{N}_t^{i:\lambda} + \mathbf{N}_{t+1}^{m:\lambda}\right]_1 + \frac{\sigma^*}{d}\|\sum_{i=1}^{\mu} w_i \mathbf{N}_t^{i:\lambda} + \mathbf{N}_{t+1}^{m:\lambda}\|^2$$

converges almost surely when $d \to \infty$ to

$$2\left(\sum_{i=1}^{\mu} w_i \mathcal{N}_t^{i:\lambda} + \mathcal{N}_{t+1}^{m:\lambda}\right) + \sigma^*\left(\sum_{i=1}^{\mu} w_i^2 + 1\right) \quad .$$

Since the optimal step-size for this model (asymptotic sphere) is theoretically known (3.11), we can replace its value in (3.57) to obtain the expression of the asymptotic median success probability on optimal conditions:

$$\boxed{\lim_{d\to\infty} \Pr_{\text{sphere}}^{\sigma^*=\sigma^*_{\text{opt}}} = \Pr\left(\mathcal{N}_{t+1}^{m:\lambda} \leq \mathcal{N}_t^{j_\sigma:\lambda} - \sum_{i=1}^{\mu} w_i(\mathcal{N}_t^{i:\lambda} - \frac{1}{2}E[\mathcal{N}_t^{i:\lambda}])\right) \quad .} \tag{3.58}$$

Then, we can define the optimal value of j_σ, j_σ^{opt}, on the asymptotic sphere function, as the value that, when replaced in (3.58), results in a success probability of 1/2. Figure 3.3 shows the results of simulating (3.58) with different values of λ and j_σ. We are mostly interested in the (relative) comparison indexes that result in a success probability close to or equal to 1/2.

We see that for all but the bottom leftmost plot ($\mu = 1$), these points of interest lie, for most values of λ, within an interval of the normalized rank between 0.20λ and 0.30λ. The case $\mu = 1$ results in more spread points of interest on the x-axis. This suggests a linear correlation between the optimal comparison index j_σ^{opt} and the population size λ.

To check this linearity, Figure 3.4 shows, from the data used in Figure 3.3, the evolution of the two integer indexes j_σ^- and j_σ^+ with the closest success probabilities to 1/2

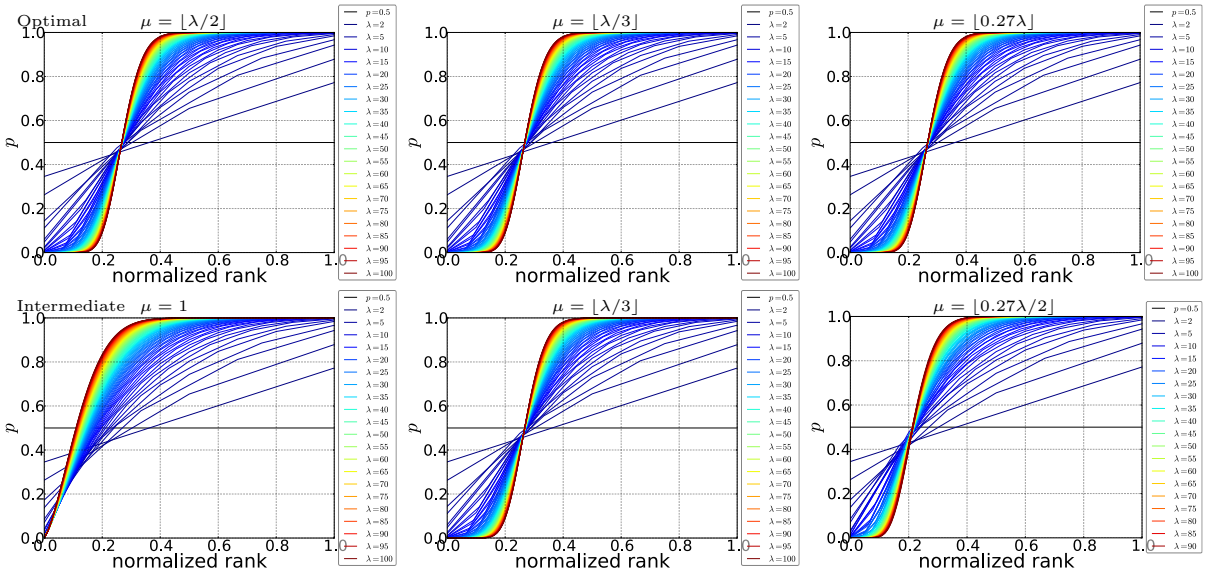

Figure 3.3: Median Success Probabilities when $\sigma^* = \sigma^*_{\text{opt}}$ on the asymptotic sphere function for different values of λ and all possible integer values of j_σ simulated from (3.58). The normalized ranks in the x-axis depict $(j_\sigma - 1)/(\lambda - 1)$ Different weightings schemes and values of μ are considered. **Upper plots**: normalized optimal weights (3.13), from left to right: $\mu = \lfloor \lambda/2 \rfloor$, $\mu = \lfloor \lambda/3 \rfloor$, $\mu = \lfloor 0.27\lambda \rfloor$; **Lower plots**: intermediate (equal) weights summing up to 1, from left to right: $\mu = 1$, $\mu = \lfloor \lambda/3 \rfloor$, $\mu = \lfloor 0.27\lambda \rfloor$. Each success probability is estimated using 10^6 data points.

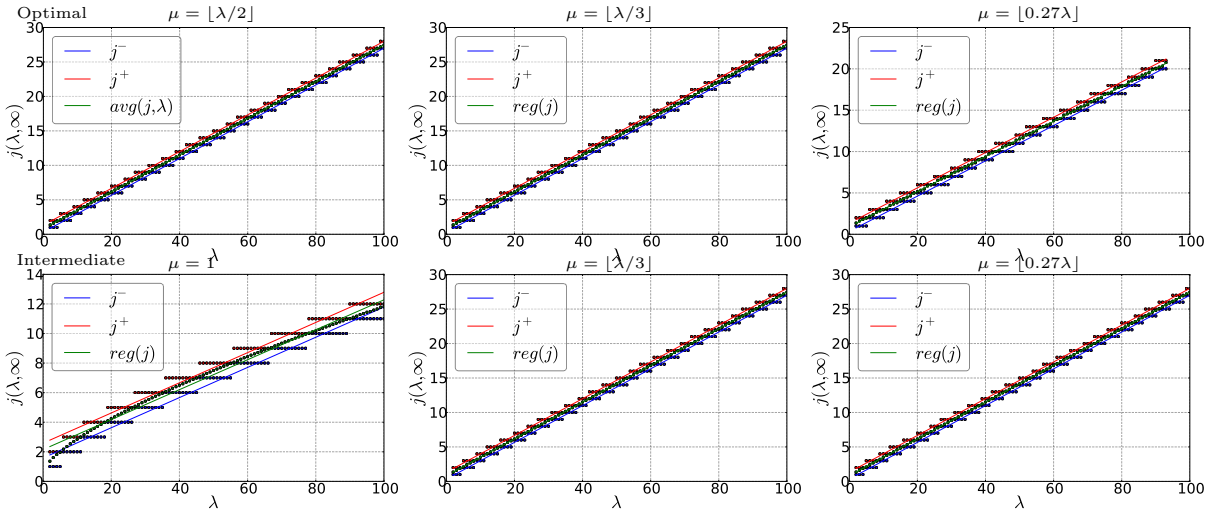

Figure 3.4: Evolution of the optimal comparison index with population size and for different weighting schemes and numbers of parent (same configurations as Figure 3.3). Shown are the lower optimal comparison indexes j^-_σ "●", the upper optimal comparison indexes j^+_σ "●" and the estimated interpolated optimal comparison index j^{opt}_σ (3.17) "●". The same data as for Figure 3.3 is used.

and respectively smaller and larger. It also shows the estimation of j_σ^{opt} as described in (3.17) for the different configurations.

First, we see a step curve that is to be expected from Figure 3.3 since the indexes j_σ^- and j_σ^+ are both integer valued. However, this is not translated into the interpolated average which has a fairy linear scaling in λ in all but the case of $\mu = 1$. This is due to the way j_σ^{opt} is computed in (3.17), even though successive values of λ might have the same j_σ^- and j_σ^+, their corresponding success probabilities, p_s^- and p_s^+ are generally different; thus the flexibility of the interpolated line.

For information, the estimated value of j_σ^{opt} for the asymptotic sphere with optimal weights and $\mu = \lfloor \lambda/2 \rfloor$ reads:

$$j_\sigma^{opt}(\lambda, \infty) = 0.27\lambda + 0.83 \ . \tag{3.59}$$

3.4.2.2.2 Finite dimension We now look into the more practical cases of finite dimensions.

In the case where the dimension does not tend to infinity, we do not have a theoretically derived expression for the optimal step-size of a $(\mu/\mu_w, \lambda)$-ES on the sphere function. However, we can still apply the definition of convergence rate in (3.9) and simulate it. Then, we consider the empirically observed best convergence rate as being optimal, and note the corresponding step-size multiplier σ^*_{opt} accordingly. We can now, similarly to the asymptotic case, look into the success probabilities for this *optimal* step-size (and for other values of the step-size), following (3.54).

Figure 3.5 shows the evolution of the convergence rates ((3.9) and (3.10)) and success probabilities ((3.54) and (3.57)) for different values of the distance to the optimal proportional step-size multiplier σ^* on the sphere function for a $(\mu/\mu_w, \lambda)$-ES with optimal weights (3.13). This figure allows us to see the success probabilities of the different indexes around optimal step-size conditions. It also gives estimates of the cost of drifting from the optimal step-size in convergence-rate loss.

The convergence rate plots on finite and infinite dimensions seem to have similar shapes. Left of the optimal step-size ($\sigma^* < \sigma^*_{\text{opt}}$), the convergence rates decrease with decreasing step-size, with $\lim_{\sigma \to 0} \text{CR} = 0$. However, on the right of σ_{opt} ($\sigma^* > \sigma^*_{\text{opt}}$), the convergence rate decreases much quicker and the algorithm diverges (negative CR), in the best of cases, as soon as the step-size reaches three times the optimal step-size. This divergence is due to the generated offspring over-shooting the optimal solution, resulting in fitness values of the offspring worse than those of the mean solution $\mathbf{x}_t$ of the distribution from which they were generated.

We also notice fairly similar success-probability graphs for the relatively small values of j_σ. With larger j_σ and on the larger dimensions, the median success probabilities decrease faster with the increasing step-size. For example in the case $\lambda = 10$, the success probability is around 0.3 when comparing the current median to the previous median (purple dashed line) with 10 times the optimal step-size and $d = 20$ whereas it is close to zero when $d = 160$ and goes to zero much faster (at about 3 times the optimal step-size) for the asymptotic case of $d \to \infty$.

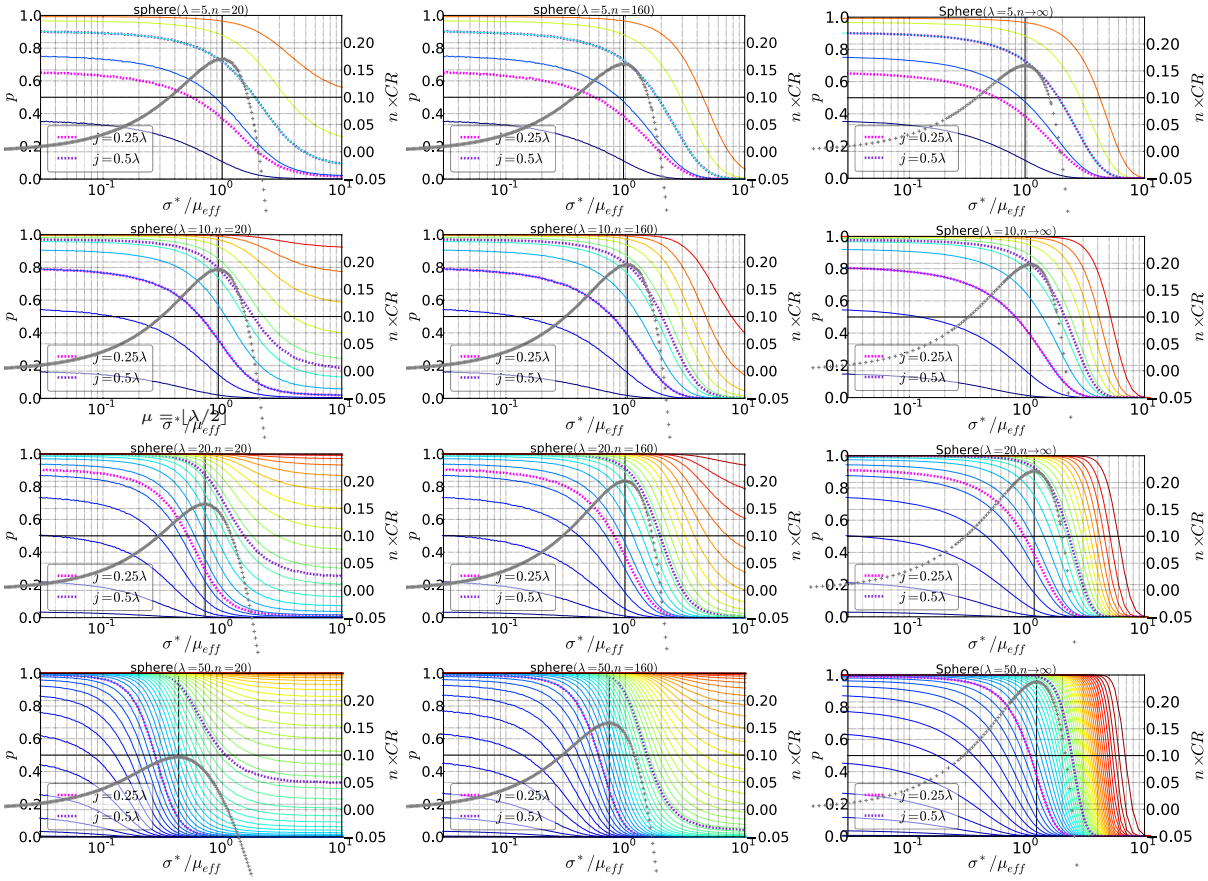

Figure 3.5: Median success probabilities for all integer values of the comparison index ($j_\sigma = 1, \ldots, \lambda$) and normalized progress rates versus step-size on the sphere function. The problem dimension and the population size are indicated on top of each plot (λ, n) (here the dimension is noted n), with the rightmost column plots dedicated for the asymptotic case (denoted by $d \to \infty$). The rainbow-colored solid lines represent the median success probabilities (equation (3.54) for finite dimension and (3.57) for $d \to \infty$). The comparison index increases as the color goes from blue $j_\sigma = 1$ to red $j_\sigma = \lambda$. The first quartile and the median are indicated by pink and purple dashed lines respectively. Corresponding success-probability values are shown on the left-side y-axes. The gray "+"s correspond to the normalized convergence rates; their values are shown on the right-side y-axis and are either simulated from (3.9) for the finite dimension case or directly computed from the limit in (3.10) for the asymptotic case. The solid vertical black line indicates the optimal step-size for the given configuration (step-size that coincides with the maximum of the gray "+" graph). Empirical simulations were done for a total of 10^5 realizations per data point.

Note that the limits of the asymptotic medians success probabilities when $\sigma^* \to 0$ correspond to the success probabilities on the linear function as we can see by comparing (3.57) with $\sigma^* \to 0$ to (3.44). These success probabilities can be seen on the leftmost part of the plots, as the success probabilities seem to converge to stationary values with decreasing σ^*.

In each plot, we are most interested in two points in particular:
(i) The first point is the crossing of the two perpendicular black lines, success probability = 1/2 (horizontal) and $\sigma^* = \sigma^*_{\text{opt}}$ (vertical). This point gives an estimation of the *optimal comparison index* for the given configuration since it corresponds to the j_σ whose resulting median success probability with optimal step-size is 1/2. It does not necessarily need to be on one of the median success probability plots since we already explained in Section 3.3.3 how we allow and manage non integer comparison indexes. For example, for a population size of $\lambda = 20$, we see the evolution of j_σ^{opt}: $d = 20 \implies j_\sigma^{opt} \simeq 7$, $d = 160 \implies j_\sigma^{opt} \simeq 6.3$ and $d \to \infty \implies j_\sigma^{opt} \simeq 6.1$.
(ii) The second point of interest is, once a comparison index j_σ is chosen, the intersection of the success-probability plot for the given j_σ (or its interpolated one if $j_\sigma \notin \mathbb{N}$) and the 1/2 success probability line. This shows the expected convergence rate of the algorithm with this value of j_σ (on the sphere function in this case). In fact, the abscissa of said intersection is the target step-size of the algorithm. This is due to the fact that the algorithm tries to achieve a Median Success Probability of 1/2, and by doing so, makes its step-size converge to the corresponding step-size that we can call *the half median-success step-size*, that we note $\hat{\sigma}_{1/2}(j_\sigma)$. The expected convergence rate of the algorithm can, thus, be deducted by looking into the convergence rate corresponding to $\hat{\sigma}_{1/2}(j_\sigma)$ in Figure 3.5. If we look back at our example, $(n = 20, \lambda = 20)$, and set $j_\sigma = 4$, $\hat{\sigma}_{1/2}(j_\sigma)$ would be about $1/3^{\text{rd}}$ of the optimal step-size and the normalized convergence rate would be around 0.10 instead of more than 0.15 (a bit over 30% loss).

The later point leads to our next set of plots: Figure 3.6; where we look into the convergence rates that correspond to the different choices of j_σ on each configuration.

In this figure, and in contrast to Figure 3.5 where the drift from the optimal step-size is investigated, we are interested in the effects and amount of performance loss resulting from a sub-optimal choice of j_σ. That is values of j_σ whose median success probabilities are different from 1/2, or conversely, values of j_σ whose corresponding target step-sizes at median success probability 1/2 are different from the optimal step-size. The performance is measured in convergence rate multiplied by dimension (as a normalization). The plots of Figure 3.6 are obtained, for each value of j_σ, by taking the convergence rate observed on its corresponding $\hat{\sigma}_{1/2}(j_\sigma)$. The results are shown for different values of d and λ with optimal normalized weights and $\mu = \lfloor \lambda/2 \rfloor$.

The plots seem quite similar for most cases, being more and more curved as the dimension increases. The optimum j_σ^{opt} (that corresponds to maximal convergence rate) shifts to the left with growing dimension until reaching its limit, when $d \to \infty$, of around the 27%-tile (see (3.59)).

As expected, progress becomes null or negative for values of j_σ larger or equal to $\lambda/2$ (in our case $\mu = \lfloor \lambda/2 \rfloor$) as the corresponding $\hat{\sigma}_{1/2}(j_\sigma)$ is too large. This can

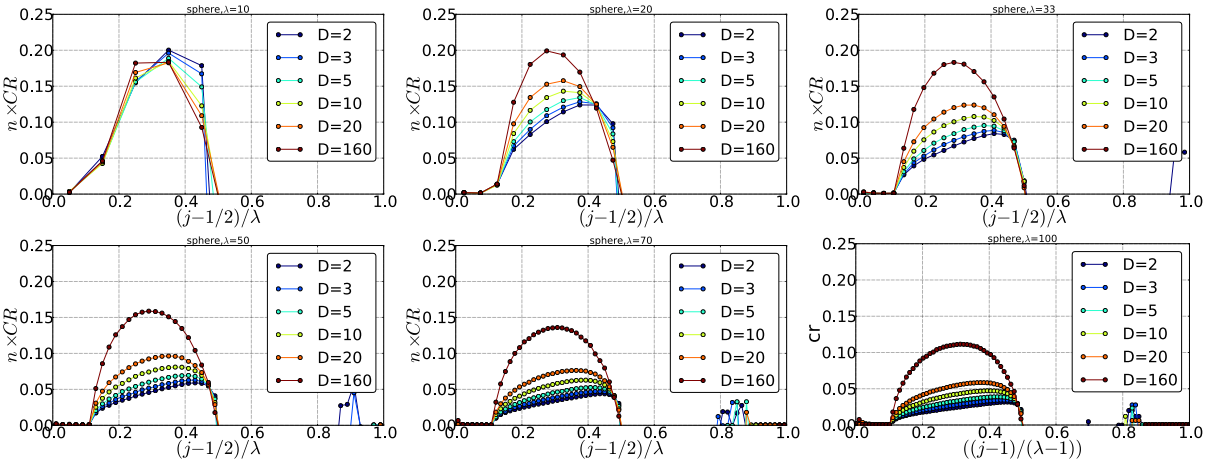

Figure 3.6: Convergence rates multiplied by problems dimension at $\hat{\sigma}_{1/2}(j_\sigma)$ versus fitness comparison quantile (either $(j_\sigma - 1/2)/\lambda$ or $(j_\sigma - 1)/(\lambda - 1)$), for different values of λ and the problem dimension (noted D) on the sphere function.

also be seen from Figure 3.5 by looking, for example, into the progress for $\hat{\sigma}_{1/2}(j_\sigma)$ with $j_\sigma = 0.5 \times \lambda$ (purple dashed lines); this progress seems to be close to zero for all considered configurations. Similarly, too small values of j_σ (10%-tile or smaller) give too small a $\hat{\sigma}_{1/2}(j_\sigma)$ to see significant progress. In fact, and for the smallest values of j_σ (again looking back at Figure 3.5) we see that the corresponding median success probabilities actually never cross the 0.5 line (e.g., $j_\sigma = 2$ in $(d = 20, \lambda = 20)$); which means that $\hat{\sigma}_{1/2}(j_\sigma)$ would tend to 0. Since the median success probability is always smaller than 1/2, the step-size would keep decreasing, eventually leading to a quasi-null progress.

On the other hand, we see the robustness of the Median Success Rule with regards to the choice of the comparison percentile, as long as it remains within reasonable bounds (see above for the extreme cases). For the considered configurations, any choice of j_σ between the 20%-tile and 40%-tile results in no worse than half the optimal convergence rate. The convergence rate graphs become less and less flat as the dimension d increases, with the optimal comparison percentile decreasing and going to its limit value for $d \to \infty$ of around the 27%-tile (see (3.59)).

The results on the sphere show a good behavior of the algorithm (convergence rate) and robustness with regards to the choice of the comparison percentile while also allowing to define its bounds ($[0.2\lambda, 0.4\lambda]$).

3.4.2.3 The Ridge Function

The third and final function we consider is the ridge function:

$$f^{\text{ridge}}(\mathbf{x}) = [\mathbf{x}]_1 + \beta \left(\sum_{i=2}^{d} ([\mathbf{x}]_i)^2 \right)^{\alpha/2} , \tag{3.60}$$

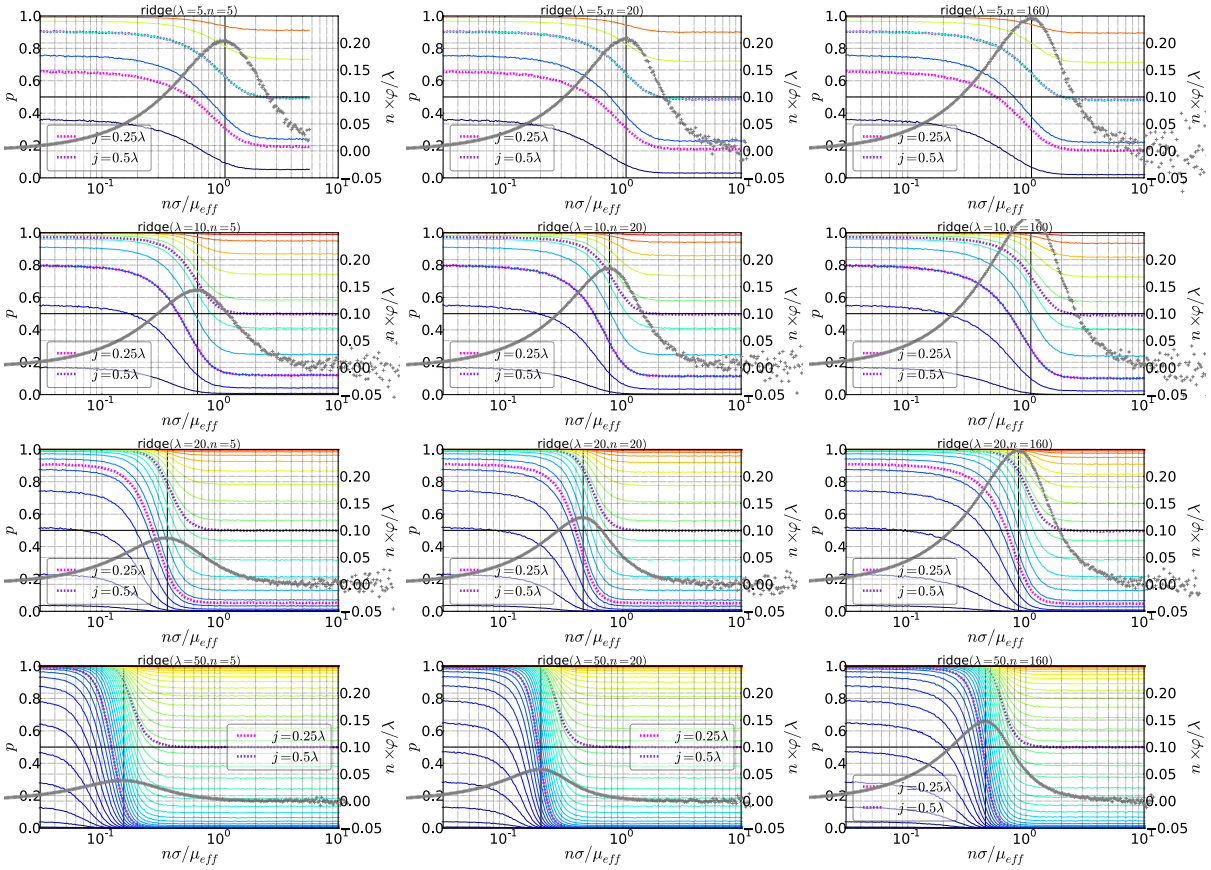

Figure 3.7: Median success probabilities for all $j_\sigma = 1, \ldots, \lambda$ and normalized progress rates (3.61) versus step-size on the ridge function; see caption of Figure 3.5 for additional details.

where α determines whether the ridge is *sharp* ($\alpha = 1$), *parabolic* ($\alpha = 2$) or has another shape. We consider the case of $\alpha = 4$ as it insures a finite optimal step-size (the condition for this being $\alpha > 2$ [Arnold and MacLeod, 2008]); $\beta = 1$ in this chapter.

The ridge function (3.60) consists of two σ-wise (σ the step-size) opposite parts: the quadratic part, $\beta \left(\sum_{i=2}^{d} ([\mathbf{x}]_i)^2 \right)^{\alpha/2}$ and the linear part, $[\mathbf{x}]_1$. The first part, similar to a sphere function treated above, has a finite optimal solution and requires a step-size that converges to zero. The second part, however, is similar to the linear function and has no finite optimum. It requires a non zero step-size. A stationary state, when reached (see [Ait Elhara et al., 2013] and [Arnold and MacLeod, 2008] for details), maintains a distance to the ridge which is constant in expectation (constant value of the quadratic part) whilst progress is achieved towards the direction of $[\mathbf{x}]_1$. This stationary state is achieved with a constant step-size that the algorithms need to find.

Figure 3.7 is similar to Figure 3.5 with the differences being: (i) first, the objective function is the ridge function (3.60) instead of the sphere function (3.4). (ii) Second, the step-size is no longer normalized by the distance to the optimum since such distance

can not be finite, so we do not plot using σ^* on the x-axis but simply against the actual step-size of the algorithm σ. (iii) Finally, the performance measure which is the progress rate instead of the convergence rate. Since the ridge function has no finite optimum, algorithms are not expected to converge to any particular finite solution. In addition, the desired behavior on the ridge function is to be in the stationary state (with the quadratic term being constant in expectation) and make progress parallel to the ridge (the first linear term) as expressed in the following equation:

$$\varphi_t = \frac{1}{t - t_0}([\mathbf{x}_t]_1 - [\mathbf{x}_{t_0}]_1) \ , \tag{3.61}$$

where t_0 is the iteration count when the stationary state is first assumed to be met. In the experiments of Figure 3.7, we set t_0 (the burn-in time) to half the chosen budget of t, with $t = 4 \times 10^4$ iterations. In addition, and in order to increase to likelihood of being in the stationary state at time t_0, the initial solution of the algorithm $\mathbf{x}_0$ is set close to the stationary state. Its first coordinate $[\mathbf{x}_0]_0 = 0$ and its quadratic term $\|[\mathbf{x}_0]_{2\ldots d}\| = R$, with R defined in Equation (12) of [Arnold and MacLeod, 2008] and $[\mathbf{x}]_{i\ldots j}, i \leq j$ the vector $([\mathbf{x}]_i, [\mathbf{x}]_{i+1}, \ldots, [\mathbf{x}]_{j-1}, [\mathbf{x}]_j)$ that we note so for simplicity and convenience.

The first thing we notice that is strikingly different from the sphere function (Figure 3.5) is that the success probabilities are weakly, if at all, dependent on dimension; and this, for the different population sizes that are considered. These success probabilities seem to only depend on the population size λ with similar success probabilities for relative indexes (j_σ/λ) especially for larger values of j_σ. This can easily be seen by comparing the success probabilities of the same highlighted dashed lines (purple and pink for median and first quartile respectively) or of the same shades of blue/red across different λ configurations. Smaller comparison indexes seem to attract to lower success probabilities as λ increases and the median success probability graphs are generally steeper around the crossing point $(\sigma, 0.5)$ with increasing population size. We also notice that as the step-size increases, the success probability for $j_\sigma = \lambda/2$ seems to converge to $1/2$ on all considered dimensions and population sizes, which means that successive medians are, in average, on the same fitness level-set for large enough values of the step-size. However, we notice that these step-size values lead to a null progress on the first coordinate (progress values are depicted in the right y-axis). This suggests that the linear part of the function plays a major role in defining the relative fitness of a solution within an offspring. We remind that we saw a similar dimension-independent behavior on the linear function (3.38) which we brought down to a single variable function (3.39) without loss of generality. In fact, assuming we are in the stationary state, the function can be seen as being, ultimately, a two variable function, the first variable represents the linear term $[\mathbf{x}]_1$ and the second represents the quadratic term $\|[\mathbf{x}]_{2\ldots d}\|$. Progress is only measured in the first variable while selection and success probability is computed on both. We also notice that contrarily to the sphere function, we see hardly any divergence, even with the largest step-size values. In fact, the progress rate plots seem symmetric when taking the step-size σ in a log-scale.

The median success probabilities of our j_σ's of interest seem to be similar to the ones observed on the sphere function. We confirm this by looking into the progress rate VS

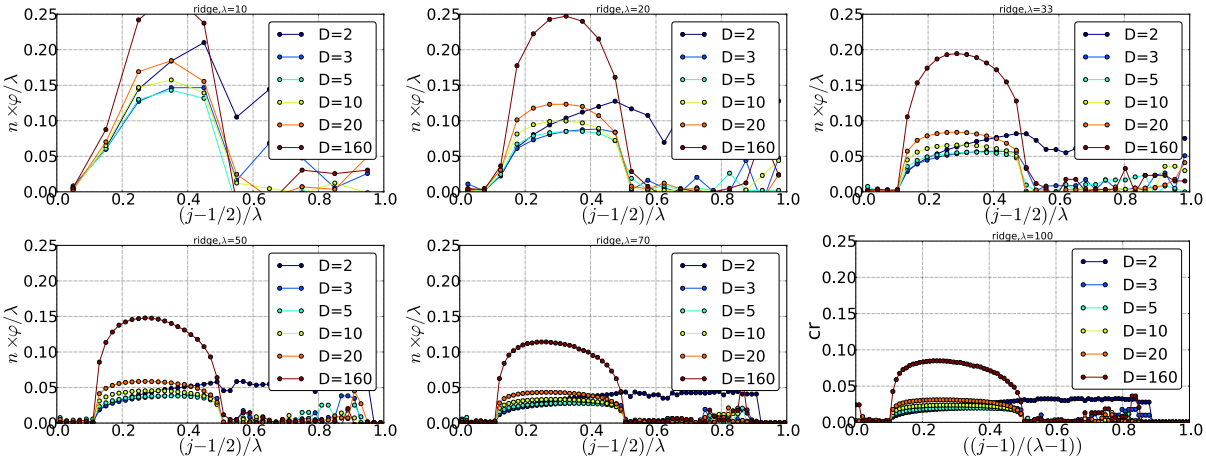

Figure 3.8: Normalized progress rates at $\hat{\sigma}_{1/2}(j_\sigma)$ versus fitness comparison quantile ($(j_\sigma - 1/2)/\lambda$ or $(j_\sigma - 1)/(\lambda - 1)$), for different values of λ and d on the ridge function (3.60). The half median-success step-size $\hat{\sigma}_{1/2}(j_\sigma)$ is the same as in Figure 3.6 replacing the sphere function by the ridge function.

j_σ in Figure 3.6.

In Figure 3.8 and similarly to Figure 3.6, the progress corresponding to each $\hat{\sigma}_{1/2}(j_\sigma)$ is looked into for the different values of λ, d and j_σ that are considered. We are interested in the values of j_σ that we might consider reasonable in the sens that they result in the algorithm seeking a step-size that leads to good progress rates.

The clear out-layers in this figure are the results for $d = 2$ (dark blue), $f^{\text{ridge}}((x_1, x_2)) = x_1 + x_2^4$, as we notice an almost constant progress rate for any choice of j_σ between the 10%-tile and 90%-tile. For other values of d, we see a behavior similar to what was observed on the sphere function with no progress when j_σ is larger than the index of the median or smaller that 0.10λ. The optimal values of j_σ are clearly within the $[0.20\lambda, 0.40\lambda]$ interval. However, the graphs seem much flatter than on the sphere (but still un-flattening as dimension increases). This means the algorithm is less sensitive to the choice of j_σ with less relative progress-loss when deviating from the optimal choice.

3.4.2.4 Comparison Index Formula

We saw that, for the three functions that were considered, the choice of the comparison percentile is not highly critical; an acceptable behavior is observed for a wide enough interval.

Putting aside the $\mu = 1$ case, setting the lower bound of the comparison percentile to the 20%-tile seems a reasonable choice. As we have seen, the optimal j_σ is generally negatively correlated with the search-space dimension. This can be seen by comparing the optimal values of j_σ in Figure 3.6 for increasing dimensions. Also, the linearity in λ is well established for several weighting scheme and number of parent choices; see for example Figure 3.4.

Figure 3.9 investigate further dependencies of the optimal j_σ, or equivalently the opti-

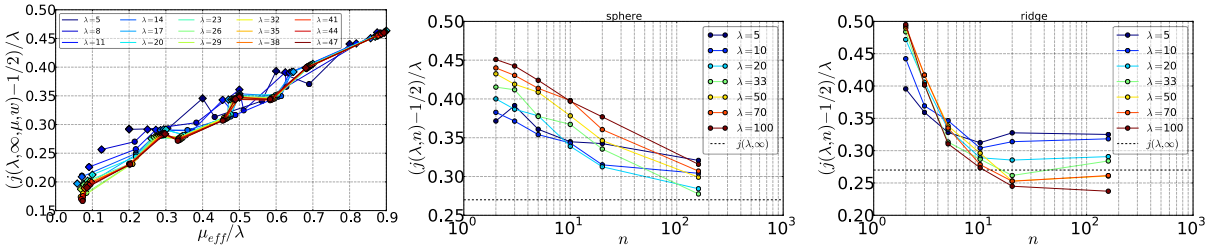

Figure 3.9: Optimal comparison quantile evolution with problem dimension and population size. This index is set to the value whose corresponding target step-size ($\hat{\sigma}_{1/2}(j_\sigma)$) maximizes the performance measure on the given function (convergence rate (3.5) on the sphere function and progress rate (3.61) on the ridge function). **Left:** Asymptotic sphere function. Two weighting schemes are considered, "•"s for optimal weights (3.13) and "◇"s for intermediate weights. Different values of μ are considered in order to obtain the different values of μ_{eff}/λ. For example, in the intermediate case, $\mu_{\text{eff}}/\lambda = \mu/\lambda$ since $\mu_{\text{eff}} = \mu$ (its maximal value). The optimal comparison index is plotted against μ_{eff}/λ. **Middle and Right:** Finite dimension (the dimension is noted n in these plots) sphere (middle) and ridge (right) functions. Optimal weights with $\mu = \lfloor \lambda/2 \rfloor$. The plots show the scaling with dimension of the comparison quantile. The dashed horizontal line shows the limit on the asymptotic sphere function estimated using (3.59).

mal comparison quantile $(j_\sigma - 1/2)/\lambda$ on the problem dimension (middle and right plots) and on the weighting scheme, and more specifically on the variance effective selection mass μ_{eff} (left plot).

The leftmost plot shows a positive linear correlation with μ_{eff}/λ of the optimal choice of j_σ with values approaching 0.5 as μ_{eff}/λ increases. The smaller values of μ_{eff}/λ are obtained when μ is relatively small and the intermediate weights ($w_i = 1/\mu, \forall i$) produce the largest possible $\mu_{\text{eff}} = \mu$ (values for optimal weights stop at around $\mu_{\text{eff}}/\lambda = 0.7$ while they reach 0.9 for intermediate weights). We see in the other two plots that the correlation with the dimension is negative, and we have already seen that the limit, when the dimension goes to infinity, of the optimal comparison index percentile is larger than one (see for example Figure 3.3 and (3.59)) on the sphere function. We see a similar behavior on the ridge function (rightmost plot) with the stationary values being reached relatively earlier. We also note that the actual values of the comparison percentile are similar on the two functions. This suggests a scaling which is roughly inversely proportional to the dimension.

Taking all this into consideration we set the value of j_σ that will be used to benchmark the algorithm (Section 3.5), as follows:

$$j_\sigma = (1 + \mu_{\text{eff}}/\lambda + 1/d) \times 20\%\text{-tile} \ . \tag{3.62}$$

The formula in (3.62) allows to have the value of j_σ well within the interval $[20\%, 40\%]$-tile that was emphasized in Figure 3.6 and Figure 3.8 since we suppose $d \geq 2$ and $\mu \leq \lambda$.

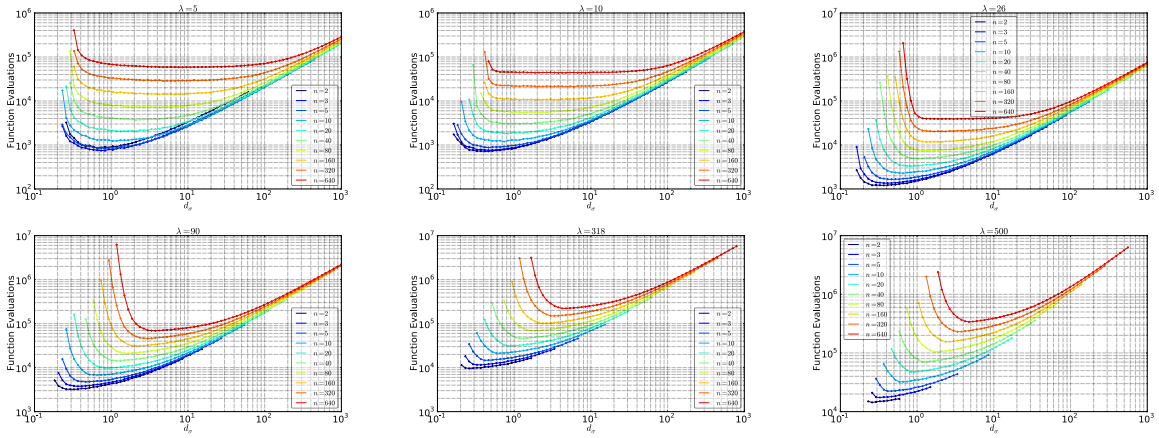

Figure 3.10: Average number of function evaluations (y-axis) of the MSR-ES (no covariance matrix adaptation) algorithm to reach a target value of 10^{-8} on the sphere function for different values of the damping parameter d_σ (x-axis, see Algorithm 1), dimension (noted n) and population size. The number of parents μ is set to $\lfloor \lambda/2 \rfloor$ and optimal weights are used (3.13). The initial step-size is $\sigma_0 = 2$ and the starting point $\mathbf{e}_1$ with an optimum at $\mathbf{0}_n$. The data for values of d_σ that do not produce 100% success are not plotted. The maximal number of function evaluations is $10^4 \times n$ and each data point is the average of $3 + \lfloor 200/n \rfloor$ runs.

3.4.3 Damping parameter

The last remaining parameter to tune for the Median Success Rule (Algorithm 1) is the *damping parameter* d_σ. This parameter allows to limit the amount of change on the step-size over a single iteration and intervenes in (3.28) by dividing the step-size update coefficient s_{t+1}. Since at any time-step t, $0 \leq s_t \leq 1$, the ratio between successive step-sizes satisfies: $\sigma_{t+1}/\sigma_t \in [\exp(1/d_\sigma)^{-1}, \exp(1/d_\sigma)]$.

In Figure 3.10, the average number of function evaluations needed to reach a target value of 10^{-8} of MSR-ES is plotted against different values of the damping parameter for different values of the population size and the problem dimension (see caption for details).

We see non-symmetric (in a log-scaled d_σ) uni-modal graphs. The optimal damping values, that is values with the lowest average number of function evaluations, seem to depend monotonously on the problem dimension. Larger dimensions require a larger damping value. On the other hand, the population size dictates the flatness of the graph, with smaller values resulting in flatter graphs.

Similarly to the reasoning used in [Brockhoff et al., 2010], we want to have damping values that are far enough from the unstable region (region where divergence is observed). Decreasing the damping quickly results in diverging behavior of the algorithm which can be seen from the lack of data points plotted (any value of the damping that results in at least one failure is omitted in the plot). Conversely, larger than optimal damping values see decreasing performance (larger numbers of function evaluations) but the algorithm

still manages to converge to the optimal solution. This behavior of the damping is similar and highly tied to that of the step-size on the sphere function. While smaller step-sizes result in slower convergence rates that tend to 0 as the step-size tends to 0, larger ones quickly lead to a diverging behavior. This can be seen in the convergence rate graphs in Figure 3.5.

We set the default value of the damping as follows:

$$d_\sigma = 2 - 2/d \ . \tag{3.63}$$

In (3.63), the value of the damping does increase with dimension but remains upper bounded by 2. Thus, on a single iteration, the largest step-size (increase) factor is $\sqrt{e} \simeq 1.649$ while the smallest (decrease) factor is $1/\sqrt{e} \simeq= 0.606$ where e designates the base of the natural logarithm.

3.5 Benchmarking

In this section, we benchmark the MSR-CMA-ES algorithm on the COCO noiseless testbed [Hansen et al., 2009] (see Tables 5.1, 5.2, 5.3 and 5.4 without the normalization term $\gamma(d)$ for the definitions of the functions and Section 2.3.1 for a brief description of the problem classes). We call MSR-CMA-ES the evolution strategy we obtain when using the Median Success Rule described in Algorithm 1 to adapt the step-size instead of the Cumulative Step-size Adaptation (CSA) in the Covariance Matrix Adaptation Evolution Strategy (CMA-ES, see Section 2.1.4.1) [Hansen and Ostermeier, 2001]. Similarly, CSA-CMA-ES will denote the original CMA-ES algorithm with CSA. Note that MSR-CMA-ES was also later benchmarked against CSA-CMA-ES and TPA-CMA-ES in [Atamna, 2015].

3.5.1 Parameter Configuration

The damping parameter of MSR-CMA-ES was set to $d_\sigma = 2$; CSA-CMA-ES uses its default damping value

$$d_\sigma = 1 + 2\max\left(0, \sqrt{\frac{\mu_{\text{eff}} - 1}{d - 1}} - 1\right) + \frac{\sqrt{\mu_{\text{eff}}}}{2\left(\sqrt{d} + \sqrt{\mu_{\text{eff}}}\right)} \ . \tag{3.64}$$

The value of d_σ in (3.64) is generally close to 1.

The initial step-size on both algorithms was set to $\sigma_0 = 2$. This is about a $1/4^{\text{th}}$ of the width of the region of interest since the optimal solutions of the problems in COCO are in $[-4, 4]^d$. A smaller value of the initial step-size ($\sigma_0 = 0.5$) was tested for both algorithms and resulted in an overall worse performance. The algorithms were allowed up to 7 restarts (within the allocated budget) with a doubling population size upon each restart (similar to [Auger and Hansen, 2005b]).

The comparison index of MSR is chosen following (3.62). The population size λ, the number of parents μ and the weights are chosen the same as in Table 4.3. The rest of the parameters are taken with their default values that can be found in the CMA-ES Python implementation CMA-ES (`https://pypi.python.org/pypi/cma`).

3.5.2 Result Discussion

Figure 3.11 compares the scaling of the two algorithms, CSA-CMA-ES in blue and MSR-CMA-ES in red, with the problem dimension on each of the 24 objective functions of COCO.

The artificial best algorithm (brown lines in Figure 3.11) is an aggregation of the best results observed on each function, for each dimension and on each target value in the BBOB-2009-2009 workshop [Hansen et al., 2010b]. It amounts to a portfolio algorithm (an algorithm which solves a problem by *choosing* one of the algorithms that compose it) that uses all the algorithms proposed in the aforementioned workshop, and on each (function,dimension,target value) triplets, chooses the best algorithm to solve it.

Statistically significant differences of performance are signaled by a black star in the marker of the better-performing algorithm; e.g., CSA-CMA-ES is significantly better than MSR-CMA-ES on the sphere function with dimension 2 (Figure 3.11).

We first see an overall similar performance of the two algorithms. On several functions ($f_4, f_5, f_7, f_{15}, f_{18}, f_{21}, f_{22}$), we see no significantly different performance from one of the algorithms on any dimension. On others (f_8, f_{12}, f_{17}), the significant differences are only observed on the largest dimension, $d = 40$. However, a number of notable statistically significant differences in performance are observed.

On the Sphere (f_1) Discus (f_{11}) and Sharp Ridge (f_{13}) functions, CSA-CMA-ES has a better performance on smaller dimensions but MSR-CMA-ES seems to scale better, outperforming CSA-CMA-ES on the larger dimensions. MSR-CMA-ES shows what seems to be a sub-linear scaling in d on the sphere function. The difference in performance is, however, relatively small despite it being significantly different.

As expected, both algorithms perform in a similar fashion on the rotated (f_{10}) and axes-aligned non-rotated (f_2) ellipsoid functions thanks to the covariance matrix adaptation. Some statistically significant ERT values are observed but the factor is at most 1.5. A similar behavior is seen on the Rosenbrock functions (f_8 and f_9). However, the plots suggest that the covariance matrix adaptation does not have the same effect on the non convex-quadratic Rastrigin functions (f_3, f_{15}). The two methods perform similarly on the rotated version (f_{15}) while MSR-CMA-ES has a clearly better performance for $d \geq 10$ on the non-rotated version f_3. This function being highly multi-modal (around 10^d local optima [Hansen et al., 2009]), it requires a number of restarts to be solved by CMA-ES and its variants. MSR-CMA-ES seems to allow more restarts, with the increasing population size allowing the algorithm more chances of finding a solution within the target value. We also see a fairly similar performance of the two algorithms on functions f_{12} and f_{17} with CSA-CMA-ES doing better mainly on dimension 40, but with a speedup which is still small.

On f_{16}, f_{19} and f_{20}, CSA-CMA-ES performs significantly better, with a scaling of the ERT in the problem dimension clearly lower than that of MSR-CMA-ES. On the other hand, on the attractive sector (f_6), MSR-CMA-ES performs similarly to CSA-CMA-ES on dimensions up to 5, then scales much higher in the dimension as the latter becomes larger. In f_6, the volume of the subspace with better fitness values decreases exponentially in the dimension; which affects the performance of success-based strategies. The Sum of

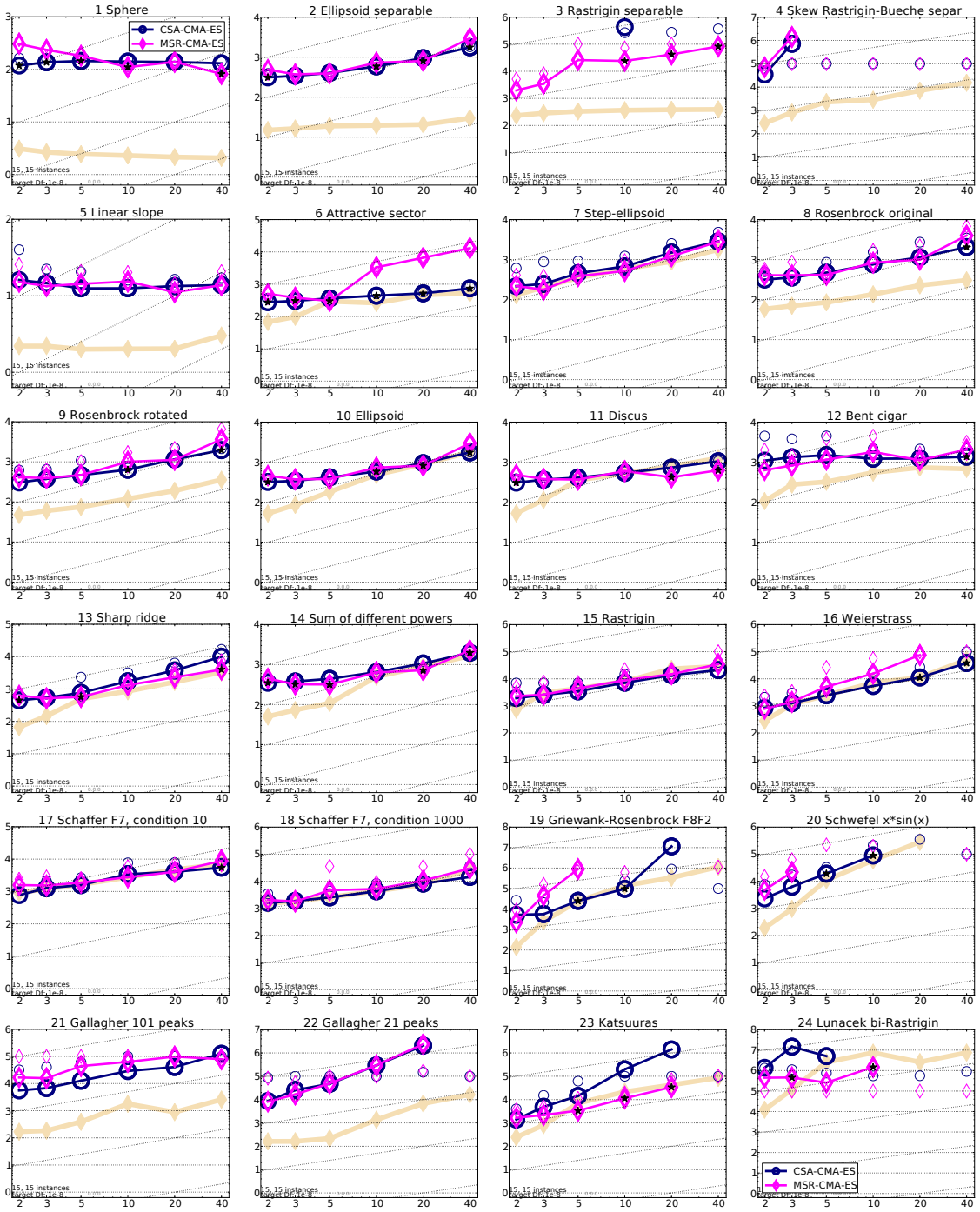

Figure 3.11: Expected running time divided by dimension versus dimension of CSA-CMA-ES and MSR-CMA-ES on the 24 functions of the COCO noiseless test-bed [Hansen et al., 2009]. Shown are the scaling figures obtained using the post-processing tools of COCO (`https://github.com/numbbo/coco`). The x-axes show the problem dimensions, while the y-axes show the corresponding Expected Running Time ERT of each algorithm. The brown solid lines show the performances of the *artificial* best algorithm. Thin "$\circ$"'s and "$\bigtriangledown$"'s indicate maximal numbers of function evaluations conducted by the respective algorithms while the thicker ones are for the actual ERTs. Horizontal dashed lines depict a linear scaling while slanted ones are for quadratic scaling.

Different Powers function (f_{14}) is similar to the Ellipsoid without being convex-quadratic. On this function, the performance of CSA-CMA-ES and MSR-CMA-ES is similar, with no algorithm dominating the other and fairly close ERT values.

Finally, on the other highly multi-modal (more than 10^d local optima) functions f_{23} and f_{24}, MSR-CMA-ES seems to give a dominating performance in comparison to CSA-CMA-ES. On f_{23}, MSR-CMA-ES scales better in d and manages to improve on the best BBOB-2009 performance on dimensions 5 and 10. The ratio between the ERTs of the two algorithms increases with the dimension, reaching a value larger than 10 on dimension 20. On the other hand, f_{24} seems to be an overall harder problem with the best BBOB-2009 algorithm managing to reach the final target value with high ERTs on dimensions larger than 3. Both methods see a number of failed runs (the plotted ERTs are above the budget used) but MSR-CMA-ES seems to manage some successful runs on dimensions up to 10 with what seem to be statistically meaningful results on dimensions 3 and 10. Note that the budget used in these experiments was lower than that needed in the best BBOB-2009 for $d \geq 3$.

To sum up, we see that MSR-CMA-ES performs well overall in comparison to CSA-CMA-ES. CSA-CMA-ES uses directly the covariance matrix adapted by CMA-ES in assuming an isotropic search space which MSR-CMA-ES does not. The generally smaller step-sizes and faster decrease of the step-size allows MSR-CMA-ES to reach stagnation points and restart quicker than CSA-CMA-ES. This leads to a better performance on some multi-modal functions, especially with the increasing population size.

Overall, MSR-CMA-ES seems a viable option as a step-size adaptation mechanism, showing reasonable performance on a wide range of functions and having no clear dysfunctionalities when coupled with the covariance matrix adaptation evolution strategy. As we will see in Chapter 4, by not assuming a particular shape of the search space or a fully adapted covariance matrix, and having less dependence on the assumed problem dimension d, the Median Success Rule solves some of the shortcomings of CSA on certain classes of problems, in addition to scaling better and thus being more appropriate for large-scale use.

3.6 Conclusion

In this chapter, we introduced a new success-based step-size adaptation mechanism for evolution strategies. At an iteration t, the Median Success Rule relies on comparing the offspring generate at this iteration to the j_σ^{th} best individual of the previous iteration $(t-1)$. The step-size is then adapted depending on the number of successes resulting from this comparison, where the success of an individual is defined as it having a better fitness than that of the j_σ^{th} best individual at $t-1$. The case of non-integer j_σ is handled using a weighted aggregation over the closest integer indexes. The success or not of the median at t determines whether the step-size will be increased (success) or decreased (failure), thus the name Median Success Rule. This correlation between median success and step-size change is obtained by having a target success rate of $1/2$ and computing the success rate as the proportion of successful offspring at t.

Theoretical derivations and empirical simulations allowed to establish the optimal and acceptable values of the main parameter of this strategy, namely the comparison index j_σ, on three functions that require different step-size adaptation regimes to be solved efficiently: (i) the linear function requires an ever increasing step-size, (ii) the sphere function a decreasing one proportionally to the distance of the mean solution to the optimum and finally (iii) the ridge function that requires the algorithm, once a stationary state is reached, to keep a constant steps-size. Interestingly, even though the optimal comparison index differs from one function to another, a wide range of values resulted in the algorithm not loosing too much of its efficiency in comparison to the optimal case (see Figure 3.6 and Figure 3.8 for example). In addition, these good performance intervals are similar across the functions. This allowed the establishment of a formula of the comparison percentile (comparison index normalized by the population size) that depend on the problem dimension and the weighting scheme used for the recombination (3.62). We also tuned the other parameters of the strategy in order to have, in the end, a fully functional step-size adaptation mechanism.

We, then, used MSR as the step-size adaptation mechanism of a $(\mu/\mu_w, \lambda)$-CMA-ES and compared the performance of the resulting algorithm, MSR-CMA-ES, to that of the default Cumulative Step-size Adaptation based CMA-ES (that we note CSA-CMA-ES to differentiate) on the noiseless COCO test-bed. MSR-CMA-ES showed comparatively good performance and only fell short on performance on a few functions. It also performed better than CSA-CMA-ES on some functions mainly because of its quicker step-size decrease and thus quicker restart on multi-modal functions.

After establishing the feasibility of MSR we see that it is a viable replacement for CSA that, in addition, comes with less restrictions. In fact, it does not directly rely on a particular class of sample distributions so it can, *a priori*, be used on any population based algorithm without necessarily normal distributed offspring (for example, in a re-sampling constraint handling algorithm). In addition, contrarily to CSA, it does not need an isotropic search space to operate on, which is practical especially when facing large scale problems and expensive problems on which algorithms can generally not afford to learn full-models but only approximated ones. We will also see in Chapter 4 that MSR performs significantly better than CSA on certain classes of functions (low effective dimension functions and low epsilon-effective dimension functions) thanks to it having no assumptions on the effective dimension of a problem, that is on the number of dimensions that actually contribute (significantly) in defining the fitness.

However, MSR being a success-based rule, it requires a negative correlation between the success measure and the desired direction of change of the step-size (whether one wants to increase or decrease the step-size). In addition, the chosen comparison index needs to have limit success probabilities, for $\sigma \to 0$ and $\sigma \to \infty$ respectively larger and smaller than $1/2$ since otherwise, the method will endlessly increase or decrease the step-size.

Finally, we note that MSR preserves all the invariance properties originally present in the Covariance Matrix Adaptation mechanism of CMA-ES. This includes invariance to monotonic transformations of the fitness function (since its a rank-based mechanism)

and invariance to affine transformations of the search-space. In addition, it does not require any additional cost in function evaluations and its internal computational cost per iteration only depends linearly on the population size (no direct dependence on the problem dimension).

Chapter 4

Effective and ε-Effective Dimensions

This chapter deals with a particular class of problems that are often encountered in large-scale optimization: low effective dimension problems. These problems have a large number of parameters but can be solved efficiently by considering an appropriate subspace of lower dimension. In fact, a relatively small number of dimensions (not necessarily aligned with the coordinate system) define the whole fitness of each solution; this number of dimensions is what we call the *effective dimension of the problem*. We also propose a natural generalization to the notion of low effective dimension by allowing a small proportion of the fitness, determined by a multiplier ε, to come from all the variables. We call this generalized notion the epsilon-effective dimension, and thus we consider low epsilon-effective dimension problems. In addition, we construct low effective dimension and low epsilon-effective dimension problems from functions that are commonly used to test continuous optimization algorithms and benchmark large-scale optimization algorithms on them.

4.1 Introduction

One interesting question when dealing with large-scale problems is the *form* in which these problems appear in real-world scenarios. Answering this question would help us design realistic benchmarks to model these problems.

One kind of large scale problems that are encountered in the literature is that of problems having an effective dimension d_{eff} smaller than that of their domains of definition d. This means that optimizing the problems in a smaller, generally linear, subspace of its original search space might suffice to find satisfactory solutions. This property was observed in the domain of integral estimation [Moskowitz and Caflisch, 1996, Wang and Fang, 2003, Wang and Sloan, 2005] [Hutter, 2009, Bergstra and Bengio, 2012, Berthier and Teytaud, 2015]. Low effective dimensionality and similar properties can be encountered in a number of other domains. In Genetic Programming, this property is called *bloat* and consists in parts of a program being unused (and variables related to these parts having no impact on fitness) [Banzhaf and Langdon, 2002, Luke and Panait, 2006, Silva and Costa, 2004, Ekárt and Németh, 2002, De Jong et al., 2001, Bleuler et al., 2001].

Bloat is even shown to be useful and removing it can hinder the performance [Langdon and Poli, 1998, Soule, 2002]. We also find similar notions in reinforcement learning [Sutton, 1996, Ratitch and Precup, 2004, Kearns et al., 2002], evolution of trees [Zhang et al., 1997], Nash equilibria [St-Pierre et al., 2011] and SVM's [Girosi, 1998].

However, it remains a topic which is relatively under-studied and under-represented in the context of continuous optimization, especially in black-box settings. There are a number of problems that exhibit this low effective dimensionality. For example, automatic configuration of the hyper parameters of state-of-the-art algorithms for solving SAT (Boolean Satisfiability) problems and Mixed Integer Programs [Hutter, 2009] showed the presence of a number of inconsequential parameters, especially when the parameter configuration space is high-dimensional. In [Bergstra and Bengio, 2012], it was shown that hyper-parameter optimization for neural networks and deep belief networks is, on most data sets, a low effective dimension problem. However, which dimensions are effective may vary from one data set to the other. The paper also shows a superior performance, on the hyper-parameter optimization problem, of random search over grid search because of this low effective dimensionality. [Berthier and Teytaud, 2015] studies the robustness of comparison-based algorithms when dealing with low effective dimension (called codimension in the paper) problems. The low effective dimension problems used in this paper are obtained by adding useless variables to problems from the BBOB-2009 test suite.

And yet, in addition to the theoretical interest that the study of this class of problems in a continuous black-box setting and design of appropriate algorithms offer, such a study will open a number of possibilities for broader real-world applications. In fact, algorithms designed for low effective dimensionality might turn out to be a good alternative to the *classical* (classical in the sense that full effective dimensionality is assumed) large-scale optimization algorithms in a number of scenarios. For example, this can be the case when the dimension of the problem is exceedingly large and sub-space optimization seems inevitable. Another scenario is when the quality of a solution is highly tied to the time needed to find it (optimality is, then, not a predominant criterion). In such a case, exploring the *most effective subspaces* (ones where the fitness varies the most) is a good approach to allow a rapid improvement over the initial solution. One can, for example, sample a number of directions with a given step-size and construct a sub-space of those directions where the fitness varies the most. The embedding approach to solving low effective dimension problems, allows, as it was the case in [Wang et al., 2013] with Bayesian Optimization and is the case in this chapter with CMA-ES, to take advantage of highly efficient algorithms that are otherwise unsuited for large-scale optimization (due to their complexity or to the poor scaling of their performance with the dimension of the search-space in which they operate).

Our intuition is that many large-scale problems could actually be of low effective dimensionality. This can, for example, be due to an over-parametrization in the modelling of a problem. It is then important to construct low effective dimension benchmarks for large-scale optimization in order to assess large-scale optimization algorithms on this particular class of problems. Chapter 5 will deal with the design of a benchmarking test suite for large-scale continuous optimization. However, it will solely, in its current

version, rely on full effective dimension problems. The proposed test suite extends on a pre-existing benchmark designed for smaller dimensions while trying to preserve as much of the properties of this benchmark (including full effective dimensionality) as possible. The introduction of low (ε-)effective dimension suites or problem classes is left for future work.

A natural generalization of low effective dimension functions is functions where not all, but a large portion of the fitness is determined by a few variables/linear combinations of variables. One main contribution of this chapter is to formally define such a class of problems with what we call *low epsilon-effective dimension*. In addition, we propose a rigorous way of building low epsilon-effective dimension test functions from functions that are widely used in continuous black-box optimization, extending from the construction method we use for low effective dimension problems.

A low effective dimension problems can be solved by optimizing in the subspace (of dimension d_{eff}) in which the fitness is determined that we will refer to as *the effective subspace*. However, being in a black-box scenario, this effective subspace and even its dimension/size are *a priori* unknown to the algorithm. The latter has only access to the overall search space in $\mathbb{R}^d$. Ideally, an algorithm would embed the search space into an optimization sub-space (of dimension d_{ss}) that coincides with the effective sub-space ($d_{\text{ss}} = d_{\text{eff}}$), thus reducing its complexity by searching instead of the whole search-space, in a sub-space of reduced dimension $d_{\text{eff}} = d_{\text{ss}} < d$. This approach of embedding into a lower-dimensioned sub-space was already proposed and implemented in the Random EMbedding Bayesian Optimization (REMBO) algorithm [Wang et al., 2013].

The embedding based approach allows to tackle large-scale problems as long as their effective dimension is low. In fact, the complexity of the algorithm is defined in terms of the dimension of the optimization sub-space, d_{ss}, instead of the dimension of the whole search space d. The optimization sub-space dimension needs to be at least as large as the effective dimension in order to insure full exploration of the fitness space.

As previously mentioned, once the embedded space defined, one can use well established low to medium dimension optimization algorithms, such as CMA-ES (see Section 2.1.4), to optimize in this low dimensional sub-space.

In Section 4.2, we define the notions of low effective dimension and low epsilon-effective dimension. We also prove two results allowing to rigorously construct low effective and low epsilon-effective dimension problems. In Section 4.3, the embedding based CMA-ES variant (SS-CMA-ES) is introduced in two versions, RSS-CMA-ES and OSS-CMA-ES and the difference between the two is investigated. We empirically compare the performance of SS-CMA-ES to LM-CMA-ES, VD-CMA-ES and CMA-ES for which different step-size adaptation mechanisms are considered on low effective and low epsilon-effective dimension problems in Section 4.4. We conclude in Section 4.5.

4.2 Function-Class Definition

Definition 1 (Effective Dimension, [Wang et al., 2013])**.** *A function f^{Low}, $f^{\text{Low}} : \mathbb{R}^d \to \mathbb{R}$ has effective dimension d_{eff} when there exists a linear subspace $\mathcal{T} \subseteq \mathbb{R}^d$ of dimension d_{eff}*

such that for any $\mathbf{x}_\top \in \mathcal{T}$ *and for any* $\mathbf{x}_\perp \in \mathcal{T}^\perp$*, where* $\mathcal{T}^\perp$ *is the orthogonal complement of* $\mathcal{T}$*, we have:* $f^{\text{Low}}(\mathbf{x}) = f^{\text{Low}}(\mathbf{x}_\top + \mathbf{x}_\perp) = f^{\text{Low}}(\mathbf{x}_\top)$.

In Definition 1, the effective dimension does not need to be defined by a subset of the variables of the problem, but only by a linear subspace of the search-space. The case where a few variables define the fitness is a special case where the basis of the subspace is canonical.

Let $f : \mathbb{R}^{d_{\text{eff}}} \to \mathbb{R}$ be a function, and let us construct f^{Low}, a function defined on $\mathbb{R}^d \to \mathbb{R}$ with an effective dimension $d_{\text{eff}} \leq d$. One straightforward way of doing this construction is to define the function f^{Low} as a call to the function f on variables $\mathbf{z} \in \mathbb{R}^{d_{\text{eff}}}$ that will be defined for each solution $\mathbf{x}$ in the domain of definition of f^{Low}:

$$f^{\text{Low}}(\mathbf{x}) = f(\mathbf{z}) \ , \tag{4.1}$$

where $\mathbf{z} = \mathbf{B}^T\mathbf{x}$ and $\mathbf{B} \in \mathbb{R}^{d \times d_{\text{eff}}}$ is a full-rank matrix that allows to map points from $\mathbb{R}^d$ to a subspace of dimension d_{eff} spanned by its columns.

We introduce, in Lemma 2, a way of constructing low effective dimension functions from convex-quadratic ones without changing the eigenvalues of the convex-quadratic function's Hessian matrix [1]:

“

Lemma 2. *Let* $f : \mathbf{z} \in \mathbb{R}^{d_{\text{eff}}} \mapsto \mathbf{z}^T\mathbf{H}\mathbf{z}$*, be a convex-quadratic function*[2] *with* $\mathbf{H} \in \mathbb{R}^{d_{\text{eff}} \times d_{\text{eff}}}$ *its symmetric positive-definite Hessian matrix. Let* $\mathbf{B} \in \mathbb{R}^{d \times d_{\text{eff}}}$ *be a full column-rank matrix whose column vectors are orthonormal (orthogonal with vectors normalized to* 1*). The Hessian matrix of the low effective dimension function*

$$f^{\text{Low}} : \mathbf{x} \in \mathbb{R}^d \mapsto f(\mathbf{B}^T\mathbf{x}) \in \mathbb{R} \tag{4.2}$$

constructed from f *using* $\mathbf{B}$ *(similarly to (4.1)) has the same non-zero eigenvalues as* $\mathbf{H}$.

Proof. We investigate the eigenvalues of $\mathbf{B}\mathbf{H}\mathbf{B}^T$*. Let* $\lambda_1, \lambda_2, \ldots, \lambda_{d_{\text{eff}}}$ *be the ordered eigenvalues of* $\mathbf{H}$*. Since* $\mathbf{H}$ *is positive semi-definite, there exists an orthogonal matrix* $\mathbf{P} \in \mathbb{R}^{d_{\text{eff}} \times d_{\text{eff}}}$ *and a diagonal matrix* $\mathbf{\Lambda} \in \mathbb{R}^{d_{\text{eff}} \times d_{\text{eff}}}$ *such that* $\mathbf{H} = \mathbf{P}\mathbf{\Lambda}\mathbf{P}^T$*. Let* $\mathbf{p}_i$ *be the* i^{th} *column vector of* $\mathbf{P}$ *that is an eigenvector of* $\mathbf{H}$ *associated to the eigenvalue* λ_i*. Let us define* $\mathbf{u}_i = \mathbf{B}\mathbf{p}_i$*. Then, and because* $\mathbf{B}^T\mathbf{B} = \mathbf{I}_{d_{\text{eff}}}$ *(the identity matrix in dimension* d_{eff}*), we have* $\mathbf{B}^T\mathbf{u}_i = \mathbf{p}_i$*. Thus*

$$\mathbf{B}\mathbf{H}\mathbf{B}^T\mathbf{u}_i = \mathbf{B}\mathbf{P}\mathbf{\Lambda}\mathbf{P}^T\mathbf{B}^T\mathbf{u}_i = \mathbf{B}\mathbf{P}\mathbf{\Lambda}\mathbf{P}^T\mathbf{p}_i = \mathbf{B}\mathbf{P}\mathbf{\Lambda}\mathbf{e}_i = \lambda_i\mathbf{B}\mathbf{P}\mathbf{e}_i = \lambda_i\mathbf{B}\mathbf{p}_i = \lambda_i\mathbf{u}_i \ , \tag{4.3}$$

where $\mathbf{e}_i$ *the* i^{th} *canonical-base vector in* $\mathbb{R}^{d_{\text{eff}}}$*. Then,* $\mathbf{u}_i$ *is an eigenvector of* $\mathbf{B}\mathbf{H}\mathbf{B}^T$ *with eigenvalue* λ_i *and we have thus proven that* $\mathbf{H}$ *and* $\mathbf{B}\mathbf{H}\mathbf{B}^T$ *have similar non-zero eigenvalues.* ” □

[1] This Lemma and its proof are taken from a draft-version of a paper co-authored with Anne Auger and Nikolaus Hansen

[2] We assume WLOG that the optimum is in zero.

Another definition of effective dimension can be found in the literature (e.g., [Caflisch et al., 1997, Owen, 2002, Surkov, 2004, Tezuka, 2005, Wang and Sloan, 2005, Asotsky et al., 2006]) where, in Definition 1, the fitness of $\mathbf{x}_\top$ needs only to be approximately that of $\mathbf{x}_\top + \mathbf{x}_\perp$ not equal to it. Instead, in this chapter, we generalize the notion from Definition 1 to that of *epsilon-effective dimension* which will be, in some point, similar to this relaxed definition.

We remind that the oscillation $\omega_f(S)$ of a function f on a set S is defined as:

$$\omega_f(S) = \sup_{\mathbf{x}\in S}(f(\mathbf{x})) - \inf_{\mathbf{x}\in S}(f(\mathbf{x})) \ . \tag{4.4}$$

Let us note $\mathbf{Ball}_d(\mathbf{x}, \delta)$ the closed ball of radius δ centered on $\mathbf{x} \in \mathbb{R}^d$ (in dimension d).

Definition 2 (Epsilon-Effective Dimension)**.** *A function $f^{\varepsilon\mathrm{L}} : \mathbb{R}^d \to \mathbb{R}$ has epsilon-effective dimension d_{eff} when there exist: (i) a d_{eff}-effective dimensional function $f^{\mathrm{Low}} : \mathbb{R}^d \to \mathbb{R}$, (ii) a function $g : \mathbb{R}^d \to \mathbb{R}$ and (iii) a constant $\varepsilon \geq 0$ such that for any $\mathbf{x} \in \mathbb{R}^d$:*

$$f^{\varepsilon\mathrm{L}}(\mathbf{x}) = f^{\mathrm{Low}}(\mathbf{x}) + \varepsilon \times g(\mathbf{x}) \ , \tag{4.5}$$

and for any $(\mathbf{x}^*, \delta) \in \mathbb{R}^d \times \mathbb{R}^*$ ($\mathbb{R}^*$ the set of real numbers excluding zero) with $\mathbf{x}_{\text{opt}} \in \mathbf{Ball}_d(\mathbf{x}^*, \delta)$:

$$\omega_g(\mathbf{Ball}_d(\mathbf{x}^*, \delta)) \leq \omega_{f^{\mathrm{Low}}}(\mathbf{Ball}_d(\mathbf{x}^*, \delta)) \ , \tag{4.6}$$

where $\mathbf{x}_{\text{opt}} = \arg\min f^{\varepsilon\mathrm{L}}(\mathbf{x})$.

In (4.6), we insure that on any ball that contains the optimum, the oscillation in the non-effective part of the function $f^{\varepsilon\mathrm{L}}$ (represented by $\varepsilon \times g$ in (4.5)) is upper bounded by the oscillation in the effective part (represented by f^{Low}). Thus, when $\varepsilon < 1$, the fitness varies more, on this ball, because of f^{Low} than because of g.

As previously mentioned when $\varepsilon = 0$, Definition 2 coincides with Definition 1 so the function becomes a *low effective dimension function.* since said function can be written as (4.5) with $\varepsilon = 0$ (g can be replaced by any function, a constant function for example and then (4.6) remains satisfied).

In a similar fashion to Lemma 2, the following proposition[3] provides a way of constructing low *epsilon*-effective dimension functions from a low-effective dimension function.

“

Proposition 1. *Let g be a normalized sphere function in dimension d:*

$$g : \mathbf{x} \in \mathbb{R}^d \mapsto \frac{1}{d}\sum_{i=1}^{d} x_i^2 \ , \tag{4.7}$$

[3]This Proposition and its proof are also taken from the draft-version of the paper co-authored with Anne Auger and Nikolaus Hansen

and $f^{\text{Low}} : \mathbf{x} \in \mathbb{R}^d \mapsto f^{\text{Low}}(\mathbf{x}) = f(\mathbf{B}^T\mathbf{x})$ *be a* d_{eff}*-effective dimension function with* $f : x \in \mathbb{R}^{d_{\text{eff}}} \mapsto \frac{1}{d_{\text{eff}}}\mathbf{x}^T\mathbf{H}\mathbf{x}$ *a convex-quadratic function normalized by* d_{eff}*. Let* $\mathbf{B} \in \mathbb{R}^{d\times d_{\text{eff}}}$ *be a full column-rank matrix whose column vectors are orthonormal. The function*

$$\begin{aligned} f^{\varepsilon\text{L}} :&\mathbb{R}^d \to \mathbb{R}, \\ &\mathbf{x} \mapsto f^{\text{Low}}(\mathbf{x}) + \varepsilon \times g(\mathbf{x}) \ , \end{aligned} \tag{4.8}$$

has epsilon-effective dimension d_{eff} *if and only if* $\lambda_{\max}(\mathbf{H}) \geq \frac{4d_{\text{eff}}}{d}$ *, where* $\lambda_{\max}(\mathbf{H})$ *is the largest eigenvalue of* $\mathbf{H}$*.*

Proof. Since in Definition 2, the oscillation is investigated on a ball that contains the minimum, the infimum of our functions on the ball is zero (value at the minimum) such that the oscillation equals to the supremum of f *on the ball. We have to control the oscillation of* g *given that of* $f^{\text{Low}}(\mathbf{x}) = \frac{1}{d_{\text{eff}}}\mathbf{x}^T\mathbf{B}\mathbf{H}\mathbf{B}^T\mathbf{x}$*. We can assume without loss of generality that* $\mathbf{B}\mathbf{H}\mathbf{B}^T$ *is diagonal (because we can consider the coordinate system where the matrix* $\mathbf{B}\mathbf{H}\mathbf{B}^T$ *is diagonal otherwise using the isotropy of the sphere function and of the oscillation conditions on balls for the euclidean norm). Let us consider the worst case scenario for (4.6), that is find* $\mathbf{x}^*$ *that realizes the smallest oscillation in* f^{Low} *and the largest oscillation in* g*. This worst case scenario is achieved when* $\mathbf{x}^*$ *has zero coordinates in the effective space and maximal (within the ball* $\widehat{\mathbf{Ball}}(\mathbf{x}^*, \delta)$*) coordinates in the non-effective space as this maximizes the oscillation in* g *and minimizes it in* f^{Low}*. Then*

$$\omega_g(\widehat{\mathbf{Ball}}(\mathbf{x}^*, \delta)) = \frac{4\delta^2}{d} \ , \tag{4.9}$$

as $\mathbf{x}^*$ *is at distance* δ *from the optimum* $\mathbf{x}_{\text{opt}}$ *while the supremum is* 2δ *away from the optimum (*$\mathbf{x}^*$ *is the symmetry point between* $\mathbf{x}_{\text{opt}}$ *and the supremum). Since* $f^{\text{Low}}(\mathbf{x}) = \frac{1}{d_{\text{eff}}}\sum_i \lambda_i x_i^2$ *where* λ_i *are the eigenvalues of* $\mathbf{H}$ *by assuming, without loss of generality, that the first* d_{eff} *coordinates correspond to the effective coordinates, otherwise the writing of* f^{Low} *as a sum is more cumbersome (this can be obtained by a permutation which is, itself, an orthogonal transformation that preserves eigenvalues). We have*

$$\omega_{f^{\text{Low}}}(\widehat{\mathbf{Ball}}(\mathbf{x}^*, \delta)) = \lambda_{\max}(\mathbf{H})\delta^2/d_{\text{eff}} \ , \tag{4.10}$$

where $\lambda_{\max}(\mathbf{H})$ *is the largest eigenvalue of* $\mathbf{H}$*. Since the effective coordinates are zero, the supremum of* f^{Low} *on* $\widehat{\mathbf{Ball}}(\mathbf{x}^*, \delta)$ *is obtained when the coordinate with the largest eigenvalue is maximized, which on* $\widehat{\mathbf{Ball}}(\mathbf{x}^*, \delta)$ *makes it equal* δ*. Hence (4.6) is satisfied if and only if* $\lambda_{\max}(\mathbf{H})\frac{\delta^2}{d_{\text{eff}}} \geq \frac{4\delta^2}{d}$*, that is* $\lambda_{\max}(\mathbf{H}) \geq \frac{4d_{\text{eff}}}{d}$*.* □

”

We use this proposition in Section 4.4.1 to construct epsilon-low effective dimension functions.

4.3 SS-CMA-ES

We now propose the Sub-Space Covariance Matrix Adaptation Evolution Strategy (SS-CMA-ES) that relies on an approach similar to [Wang et al., 2013] to solve low effective dimension and low epsilon-effective dimension problems by taking advantage of the low effective dimension property. SS-CMA-ES relies on the following theorem from [Wang et al., 2013]:

Theorem 1. *Given a function* $f^{\text{Low}} : \mathbb{R}^d \to \mathbb{R}$ *with effective dimension* d_{eff} *and a matrix* $\mathbf{A} \in \mathbb{R}^{d \times d_{\text{ss}}}$ *whose entries are independent standard-normally distributed, with* $d_{\text{ss}} \geq d_{\text{eff}}$*; we have with probability* 1*:*

$$\forall \mathbf{x} \in \mathbb{R}^d, \exists \mathbf{y} \in \mathbb{R}^{d_{\text{ss}}} : f^{\text{Low}}(\mathbf{x}) = f^{\text{Low}}(\mathbf{A}\mathbf{y}) \ . \tag{4.11}$$

We call d_{ss} the optimization-subspace dimension while d_{eff} and d are, respectively, the effective dimension and the problem/global dimension.

Theorem 1 ensures that for low effective dimension functions, an optimal solution $\mathbf{x}_{\text{opt}}$ can be found by performing the search in the subspace spanned by the column vectors of the matrix $\mathbf{A}$ as there exists $\mathbf{z}_{\text{opt}} \in \mathbb{R}^{d_{\text{ss}}}$ satisfying $f(\mathbf{x}_{\text{opt}}) = f(\mathbf{A}\mathbf{z}_{\text{opt}})$, with $\mathbf{x}_{\text{opt}}$ the optimal solution of f^{Low}.

We expect embedding based approaches to still remain effective when the definition of a low effective dimension problem is relaxed and extended to allow for a relatively small change in the fitness to come from the non effective dimensions as in (4.5) when $\varepsilon > 0$, namely the *low epsilon-effective dimension* problems (Definition 2).

The simple idea behind SS-CMA-ES is to use CMA-ES [Hansen and Ostermeier, 2001] (see Section 2.1.4) to search in the embedded subspace spanned by the columns of $\mathbf{A}$. CMA-ES replaces the Bayesian Optimization algorithm used in REMBO [Wang et al., 2013] and has the advantage of not requiring the search-space to be bounded. It is also a well developed algorithm with state of the art performance on several continuous optimization benchmarks (see for example [Hansen et al., 2010b]).

4.3.1 RSS-CMA-ES and OSS-CMA-ES

We consider two variants of SS-CMA-ES depending on the nature of its embedding matrix. In the first variant RSS-CMA-ES, for Random matrix Sub-Space Covariance Matrix Adaptation, the matrix $\mathbf{A} \in \mathbb{R}^{d \times d_{\text{ss}}}$, has its entries independently distributed following a normal distribution with mean 0 and variance $1/d$. The length of the vectors are normalized to have similar expected lengths across dimensions (these expected lengths are in the interval $[\frac{\sqrt{d}}{\sqrt{d+1}}, 1]$ [Chandrasekaran et al., 2012]).

On the other hand in OSS-CMA-ES, for Orthonormal matrix Sub-Space Covariance Matrix Adaptation, additionally, the column vectors of $\mathbf{A}$ are ortho-normalized, using the Gram-Schmidt process. In this case, all vectors have length one and are mutually orthogonal.

We will empirically investigate the effect from orthogonalizing the vectors of the embedding matrix $\mathbf{A}$.

The pseudo-code of SS-CMA-ES is shown in Algorithm 2. The core of the embedding based approach is that the algorithm searches only in a d_{ss}-dimensional subspace instead of the original, d dimensioned one. So all the original-algorithm complexities are expressed in d_{ss} instead of d.

Algorithm 2 Sub-Space Covariance Matrix Adaptation Evolution Strategy (SS-CMA-ES)

d: problem dimension.
f: objective function.
d_{ss}: Optimization-subspace dimension.
$\mathbf{x}_{\text{start}}$: initial solution.
σ: step-size.
$\mathbf{C}$: $d_{\text{ss}} \times d_{\text{ss}}$ covariance matrix, initially $\mathbf{I}_{d_{\text{ss}}}$.
$\mathbf{D}_{\text{CMA}}$: $d_{\text{ss}} \times d_{\text{ss}}$ diagonal matrix containing the square roots of the eigenvalues of the covariance matrix $\mathbf{C}$.
$\mathbf{B}_{\text{CMA}}$: $d_{\text{ss}} \times d_{\text{ss}}$ orthogonal matrix containing the normalized eigenvectors of $\mathbf{C}$.
λ: population size of the algorithm.
$\mathbf{A}$: $d \times d_{\text{ss}}$ embedding matrix.
doUpdateBD: is true if and only if the singular value decomposition of $\mathbf{C}$ into $\mathbf{B}_{\text{CMA}}\mathbf{D}_{\text{CMA}}\mathbf{D}_{\text{CMA}}^T\mathbf{B}_{\text{CMA}}^T$ is to take place in the current iteration. Initially True.

1: $\mathbf{y} \leftarrow \mathbf{0}_{d_{\text{ss}}}$
2: Generate $\mathbf{A}$
3: **while** (No stopping criterion is met) **do**
4: set doUpdateBD depending on the iteration counter
5: **if** doUpdateBD **then**
6: $\mathbf{D}_{\text{CMA}}, \mathbf{B}_{\text{CMA}} \leftarrow \text{SingularValueDecomposition}(\mathbf{C})$
7: **end if**
8: $\mathbf{y}^1, \ldots, \mathbf{y}^\lambda \sim \mathbf{y} + \sigma\mathbf{B}_{\text{CMA}}\mathbf{D}_{\text{CMA}}\mathcal{N}(\mathbf{0}_{d_{\text{ss}}}, \mathbf{I}_{d_{\text{ss}}})$
9: **for** $1 \leq i \leq \lambda$ **do**
10: $\mathbf{x}^i \leftarrow \mathbf{A}\mathbf{y}^i + \mathbf{x}_{\text{start}}$
11: $f^i \leftarrow f(\mathbf{x}^i)$
12: **end for**
13: Sort $\mathbf{y}^1, \ldots, \mathbf{y}^\lambda$ according to $f^1, \ldots, f^\lambda$
14: Adapt $\mathbf{y}$, σ and $\mathbf{C}$ given $\mathbf{y}^1, \ldots \mathbf{y}^\lambda$ and $f^1, \ldots, f^\lambda$
15: **end while**

Algorithm 2 optimizes over the coordinates of $\mathbf{y} \in \mathbb{R}^{d_{\text{ss}}}$ and evaluates the fitness of each $\mathbf{y}$ depending on the solution $\mathbf{x}$ in $\mathbb{R}^d$ that corresponds to it in the original search space since the problem f is defined in $\mathbb{R}^d$. Thus the sampled solutions (Line 8) are generated in the optimization subspace. The embedding matrix $\mathbf{A}$ is then used to compute $\mathbf{x}$ from $\mathbf{y}$ by taking into account the initial solution $\mathbf{x}_{\text{start}}$ (Line 10).

The only difference between RSS-CMA-ES and OSS-CMA-ES is in Line 2, and whether the sampled embedding matrix $\mathbf{A}$ is orthogonalized or not.

4.3.2 Complexities

We will now investigate the complexity of Algorithm 2 in terms of cost per function evaluation, number of function evaluations needed given certain scalings in the search-space dimension and in terms of computational (CPU) time.

4.3.2.1 In Number of Function Evaluations

When comparing to a standard CMA-ES (operating on a same dimension d_{ss}), SS-CMA-ES has an added cost, per functions evaluation or per offspring, that comes from Line 10 in Algorithm 2.

For each offspring $\mathbf{y}^i$, computing its corresponding $\mathbf{x}^i$ costs $d \times d_{\text{ss}}$ multiplications (multiplication of a $d \times d_{\text{ss}}$ matrix with a vector of size d_{ss}). This cost is to be added to the innate internal complexity of CMA-ES. The number of operations needed to sample each solution is in $O(d_{\text{ss}}^2)$ (Line 8 in Algorithm 2), updating the covariance matrix (Line 14) is in $O(d_{\text{ss}}^2)$ and factorizing it in $O(d_{\text{ss}}^3)$ (Line 6). The latter cubic complexity is brought down to quadratic ($O(d_{\text{ss}}^2)$) by applying the covariance matrix update only once every a linear number of generations [Hansen and Ostermeier, 2001, Ros and Hansen, 2008]. In Algorithm 2, this is controlled by setting the parameter *doUpdateBD* in Line 4 making it trigger in the desired iterations. Since $d_{\text{ss}} \leq d$, the overall complexity per function evaluation is in $O(d \times d_{\text{ss}})$. Note that in our case, the $O_\forall$, $O_\exists$ and $\hat{O}$ definitions of asymptotic behavior when multiple variables are involved (in our case, d and d_{ss}) from [Howell, 2008] all apply.

Let us consider that the algorithm needs a linear (respectively quadratic), in the search space dimension d_{ss}, number of function evaluations in order to reach a given target value. Then, SS-CMA-ES would use in the order of $O(d \times d_{\text{ss}}^2)$ (respectively $O(d \times d_{\text{ss}}^3)$) function evaluations to reach this target. In comparison, a CMA-ES operating on the original $\mathbb{R}^d$ search space would need in the order of $O(d^3)$ (respectively $O(d^4)$) operations, assuming the same scalings in numbers of function evaluations.

These complexities are summarized in Table 4.1 where we suppose that, in a search-space of dimension n, the algorithms need either linear ($O(n)$) or quadratic ($O(n^2)$) times in terms of number of function evaluations to *find* (within a given numeric precision) the optimum.

One important thing to note in Table 4.1 is that these complexities (in search space dimension) are expected to be relevant on low effective dimension problems thanks to Theorem 1 which states that searching in the d_{ss} dimensioned sub-space is sufficient to find the optimum. For full effective dimension problems, one can not expect embedding based algorithms to have these same linear or quadratic complexities in their optimization subspace dimension since, in theory, the full search space is to be explored.

4.3.2.2 CPU Time

Regarding the CPU timing of SS-CMA-ES, the left plot of Figure 4.1 shows, for a given problem dimension d and effective dimension d_{eff}, the evolution of the average CPU times

algorithm	RSS-CMA-ES	OSS-CMA-ES	CMA-ES
initialization	$O(d \times d_{\text{ss}})$	$O(d \times d_{\text{ss}}^2)$	$O(d)$
cost per f-evaluation	$O(d \times d_{\text{ss}})$	$O(d \times d_{\text{ss}})$	$O(d^2)$
scaling in # f-evaluations	$O(d_{\text{ss}})\|O(d_{\text{ss}}^2)$	$O(d_{\text{ss}})\|O(d_{\text{ss}}^2)$	$O(d)\|O(d^2)$
overall cost (linear|quadratic)	$O(d \times d_{\text{ss}}^2)\|O(d \times d_{\text{ss}}^3)$	$O(d \times d_{\text{ss}}^2)\|O(d \times d_{\text{ss}}^3)$	$O(d^3)\|O(d^4)$

Table 4.1: Estimated internal complexities of RSS-CMA-ES and OSS-CMA-ES in comparison to CMA-ES on low effective-dimension problems. Shown are (from top to bottom): cost of initialization, cost per function evaluation, linear and quadratic scalings in numbers of function evaluations and expected overall cost when considering these two scalings.

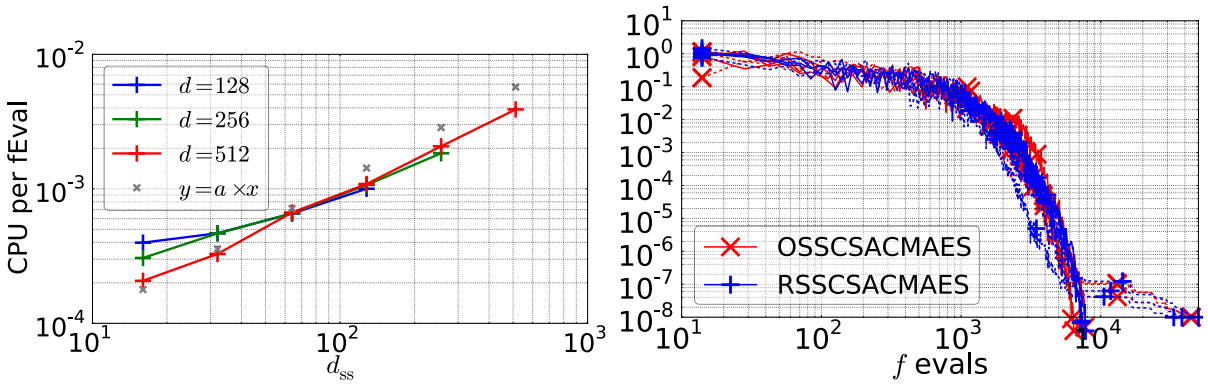

Figure 4.1: **Left:** Average CPU time per function evolution versus optimization-subspace dimension d_{ss} on the $f^{\varepsilon\text{LRosen}}$ function (see Section 4.4.1). Three values of the problem dimension d are considered with $d_{\text{eff}} = 16$. The CPUs are averaged over $100 \times D$ function evaluations each. **Right:** Evolution of the best fitness on the current iteration for RSS-CMA-ES and OSS-CMA-ES on $f^{\varepsilon\text{LSphere}}$. The problems dimension is $d = 100$, the effective dimension $d_{\text{eff}} = 9$, the optimization subspace dimension $d_{\text{ss}} = 36$ and two values of ε are considered: $\varepsilon = 0$ (solid lines) and $\varepsilon = 10^{-8}$ (dashed lines).

per function evaluation of SS-CMA-ES for different values of d_{ss}. The rightmost data points (d_{ss}=d) allow to compare to the standard CMA-ES algorithm.

As expected from the complexity study, we see a near linear scaling (compare with the gray "+"s) of the average CPU time per function evaluation with d_{ss}. We also see similar CPU times for the different values of the problem dimension d. This allows the use of SS-CMA-ES for problems with large dimension as long as the value of d_{ss} is small enough. Since d_{ss} needs only to be as large as the effective dimension d_{eff} (Theorem 1), SS-CMA-ES is reasonable, in practice, as a method of solving low effective dimension problems, even when the problem dimension d is large.

4.3.3 Conditioning of the Embedding Matrix A

One difference between RSS-CMA-ES and OSS-CMA-ES is the conditioning of the problem that is passed to the optimization algorithm (in our case, CMA-ES). A priori, the use of an orthonormal matrix should better preserve the eigenvalues, and thus the conditioning, of the original problem. That is the original problem defined in $\mathbb{R}^d$ in contrast to the subspace problem defined in $\mathbb{R}^{d_{\text{ss}}}$ that the optimization algorithm *sees*.

In this section, we quantify the change in conditioning observed on a low effective dimension problem when applying a *random* matrix or an *orthonormal* matrix $\mathbf{A}$, which represent, respectively, the RSS-CMA-ES and OSS-CMA-ES variants of SS-CMA-ES. We define the *conditioning* κ of a symmetric, positive semi-definite matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$ as the ratio between its largest and smallest strictly positive eigenvalues:

$$\kappa(\mathbf{M}) = \frac{|\lambda_1(M)|}{|\lambda_m(M)|} \ , \tag{4.12}$$

where λ_i is the i^{th} largest, strictly positive, eigenvalues of $\mathbf{M}$, and m satisfying $\lambda_1 \geq , \cdots \geq \lambda_m > 0$; and when $m < n$, $\lambda_{m+1}, \ldots, \lambda_n = 0$.

Now, if we consider that $\mathbf{M} = \mathbf{A}\mathbf{A}^T$ and $\widehat{\mathbf{M}} = \mathbf{A}^T\mathbf{A}$ with $\mathbf{A} \in \mathbb{R}^{n\times m}$ and $m < n$, then both matrices, $\mathbf{M}$ and $\widehat{\mathbf{M}}$, have the same m strictly positive eigenvalues with $m = \text{rank}(\mathbf{A})$. Consequently, and given the definition of conditioning in (4.12), $\kappa(M) = \kappa(\widehat{M})$.

When considering RSS-CMA-ES, the matrix $\mathbf{A}^T\mathbf{A}$ is the same as the one defined in [Edelman, 1989] as $\mathbf{W}(d_{\text{ss}}, d)$. For this matrix, the limits of the largest eigenvalue λ_1 and the smallest strictly positive eigenvalue $\lambda_{m=d_{\text{ss}}}$, when both d and d_{ss} tend to infinity with $d_{\text{ss}}/d \to r \in [0, 1]$, are given in Propositions 6.1 and 5.1 of [Edelman, 1989]:

$$\frac{1}{d}\lambda_1 \overset{a.s.}{\to} (1 + \sqrt{r})^2 \ , \tag{4.13}$$

$$\frac{1}{d}\lambda_{d_{\text{ss}}} \overset{a.s.}{\to} (1 - \sqrt{r})^2 \ . \tag{4.14}$$

Thus, the conditioning, for the same limits, satisfies:

$$\kappa(\mathbf{M}) \overset{a.s.}{\to} \frac{(1 + \sqrt{r})^2}{(1 - \sqrt{r})^2} \ . \tag{4.15}$$

Numerical simulations shown in Figure 4.2 show consistent results with (4.15).

In Figure 4.2, we show $\kappa(\mathbf{A}^T\mathbf{A})$ with fixed d and varying d_{ss} (left plot) and when both are varied while satisfying:

$$d_{\text{ss}} = 4\lfloor 3\log_2 d - 10 \rfloor \ , \tag{4.16}$$

(right plot) which will be the default formula for the optimization-subspace dimension through this chapter (see Section 4.4.2.2). The results are only shown for $\mathbf{A}$ *random* since it is not difficult to show that for *orthonormal* $\mathbf{A}$, the conditioning of $\mathbf{A}$ equals to 1 since in this case, $\mathbf{A}^T\mathbf{A} = \mathbf{I}_{d_{\text{ss}}}$ (the column vectors of *orthonormal* $\mathbf{A}$ have norm 1 and are mutually orthogonal).

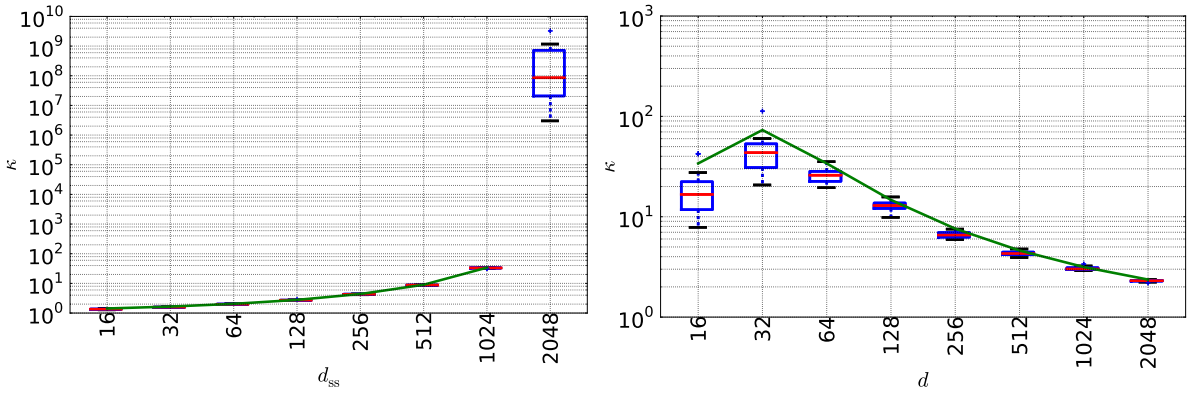

Figure 4.2: Conditioning of $\mathbf{A}^T\mathbf{A}$ with $\mathbf{A} \in \mathbb{R}^{d \times d_{\mathrm{ss}}}$ a *random* matrix for different values of d and d_{ss}. The solid green lines show the corresponding (with the same value of $r = d_{\mathrm{ss}}/d$) asymptotic value when both d and d_{ss} tend to infinity (4.15). **Left:** $d = 2048$. **Right:** $d_{\mathrm{ss}} = 4\lfloor 3\log_2 d - 10 \rfloor$. The results are obtained using 19 data points and show the median (red line), the upper and lower quartiles (box limits), the lowest and highest data points still within $1.5 \times$ IQD (the Inter-Quartile Distance) of the lower quartiles and higher quartile respectively (whiskers) and eventual outliers outside of the whiskers (+). For the asymptotic case, when $d = d_{\mathrm{ss}}$, the conditioning tends to infinity as the smallest eigenvalue tends, almost surely, to 0 (see (4.14)), thus the absence of extension on the green line for the rightmost data point of the left plot.

The left plot shows that the asymptotic values remain accurate for relatively small values of d and d_{ss}. In the special case $d = d_{\mathrm{ss}}$, the limit when $d \to \infty$ of the conditioning is infinite (replace r with 1 in (4.15)) while it is high but remains finite for $d = 2048$. The observed conditionings are significantly larger than 1, increasing steadily as r increases. However, if we look into the values we are interested in ($d_{\mathrm{ss}} = 4\lfloor 3\log_2 d - 10 \rfloor$) on the right plot, this conditioning decreases as d increases. If we look into (4.15), this is justified by the fact that d_{ss}, as defined in (4.16), increases slower than d (we set d_{ss} in the logarithm of d), meaning that r, in this case, decreases as d increases:

$$r = \frac{4\lfloor 3\log_2 d - 10 \rfloor}{d} \ . \tag{4.17}$$

The reason we see a pic of the conditioning, for both asymptotic and empiric values, on the right plot is because in (4.17), r is not monotonous for smaller values of d, it is smaller for $d = 16$ than for $d = 32$, and is strictly decreasing after $d = 32$. Its limit when $d \to \infty$ is 0.

Let us consider SS-CMA-ES (Algorithm 2) on a low effective dimension problem generated using $\mathbf{B} \in \mathbb{R}^{d \times d_{\mathrm{eff}}}$ on a sphere function (f in (4.1) is a sphere function):

$$f^{\mathrm{LowSphere}}(\mathbf{x}) = f(\mathbf{z}) \ , \tag{4.18}$$

with $\mathbf{z} = \mathbf{B}^T\mathbf{x}$ and $f(\mathbf{z}) = \mathbf{z}^T\mathbf{z}$.

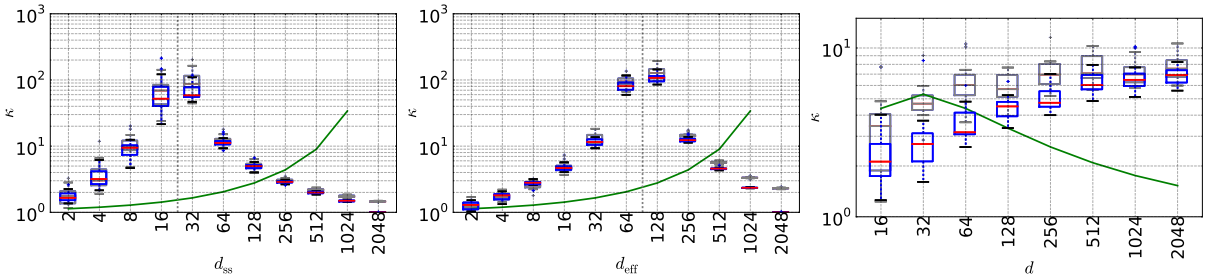

Figure 4.3: Overall conditioning on $f^{\text{LowSphere}}$, i.e., conditioning of $\mathbf{A}^T\mathbf{B}\mathbf{B}^T\mathbf{A}$ given the algorithm embedding matrix $\mathbf{A} \in \mathbb{R}^{d,d_{\text{ss}}}$ and effective-dimension lowering matrix $\mathbf{B} \in \mathbb{R}^{d,d_{\text{eff}}}$. See caption of Figure 4.2 for the description of the box plots. The gray boxes are for $\mathbf{A}$ *random* while the blue ones are for $\mathbf{A}$ *orthonormal*. The solid green lines show the asymptotic values as defined in (4.15). **Left:** $d = 2048$, $d_{\text{eff}} = \lfloor 3\log_2 d - 10 \rfloor = 23$ and d_{ss} varying on the x-axis. **Middle:** $d = 2048$, d_{eff} varying on the x-axis and $d_{\text{ss}} = 4\lfloor 3\log_2 d - 10 \rfloor = 92$. **Right:** d, $d_{\text{eff}} = \lfloor 3\log_2 d - 10 \rfloor$ and $d_{\text{ss}} = 4d_{\text{eff}} = 4\lfloor 3\log_2 d - 10 \rfloor$. The expressions for d_{ss} and d_{eff} will be the default ones used for $\mathbf{A}$ and $\mathbf{B}$ respectively.

Since in SS-CMA-ES $\mathbf{x} = \mathbf{A}\mathbf{y} + \mathbf{x}_{\text{start}}$ (Line 10 of Algorithm 2), $\mathbf{z} = \mathbf{B}^T(\mathbf{A}\mathbf{y} + \mathbf{x}_{\text{start}})$, thus the function that the CMA-ES algorithm *sees* and optimizes is:

$$\begin{aligned} f^{\text{SS}} &: \mathbb{R}^{d_{\text{ss}}} \to \mathbb{R} \\ f^{\text{SS}}(\mathbf{y}) &= (\mathbf{A}\mathbf{y} + \mathbf{x}_{\text{start}})^T\mathbf{B}\mathbf{B}^T(\mathbf{A}\mathbf{y} + \mathbf{x}_{\text{start}}) \\ &= \|\mathbf{B}^T(\mathbf{A}\mathbf{y} + \mathbf{x}_{\text{start}})\|^2 \\ &= \|\mathbf{B}^T\mathbf{A}\mathbf{y} + \mathbf{B}^T\mathbf{x}_{\text{start}}\|^2 \\ &= \mathbf{y}^T\mathbf{A}^T\mathbf{B}\mathbf{B}^T\mathbf{A}\mathbf{y} + 2\mathbf{x}_{\text{start}}^T\mathbf{B}\mathbf{B}^T\mathbf{A}\mathbf{y} + \mathbf{x}_{\text{start}}^T\mathbf{B}\mathbf{B}^T\mathbf{x}_{\text{start}} \ . \end{aligned} \tag{4.19}$$

Then, one should look at the conditioning of $\mathbf{A}^T\mathbf{B}\mathbf{B}^T\mathbf{A}$ for both cases: $\mathbf{A}$ *random* and $\mathbf{A}$ *orthonormal*, since the solution the algorithm varies and optimizes is $\mathbf{y}$ not $\mathbf{x}$.

Figure 4.3 shows estimates of $\kappa(\mathbf{A}^T\mathbf{B}\mathbf{B}^T\mathbf{A})$ from 19 independent pairs of the two matrices $\mathbf{A}$ and $\mathbf{B}$ considering *random* and *orthonormal* $\mathbf{A}$ matrices. The matrices $\mathbf{B}$ are orthonormal. We see that both types of $\mathbf{A}$ matrices (and thus RSS-CMA-ES and OSS-CMA-ES) result in similar conditionings on the low effective dimension sphere function. The left and middle plots show rather similar conditioning values between RSS-CMA-ES and OSS-CMA-ES for small values of d_{ss} and d_{eff} with a difference appearing as these two values increase (larger conditioning values for RSS-CMA-ES). The final values settle, when $d_{\text{ss}} = d$ or $d_{\text{eff}} = d$, at 1 for OSS-CMA-ES while staying at a larger value for RSS-CMA-ES.

Looking into the right plots that show the results for the values of d_{ss} and d_{eff} that will be used by default, we only see some significant differences on relatively small values of d. For larger values, the conditioning values seem to stabilize around the same value (around 7 for $d = 2048$).

Even though individually (not considering the low effective dimension function), OSS-CMA-ES generates smaller conditioning values (equal to 1 since the vectors of $\mathbf{A}$ are

orthonormal while larger for RSS-CMA-ES, see Figure 4.2), the application of $\mathbf{B}$ in order to have a low effective dimension problem results in there being no relevant difference, between the two variants, in the conditionings of the problems that the internal CMA-ES algorithm tackles (defined in $\mathbf{y}$). This suggests similar performance for RSS-CMA-ES and OSS-CMA-ES on the low effective dimension problems. Note that this result does not apply directly to the case of full effective dimension but low epsilon-effective dimension (Definition 2). This will be further investigated in the following sections.

4.4 Performance Assessment

We compare the performances of RSS-CMA-ES and OSS-CMA-ES to those of two *standard* (not designed for low effective dimension optimization) large scale optimization algorithms that, too, are based on CMA-ES: the Limited Memory CMA-ES (LM-CMA-ES) [Loshchilov, 2014] and VD-CMA-ES [Akimoto et al., 2014]. Details of these algorithms can be found in Section 2.2.1.3. In addition, we also compare to the default CMA-ES algorithm [Hansen and Ostermeier, 2001] for which we consider different step-size adaptation mechanisms: (i) Cumulative Step-size Adaptation (CSA, see Section 2.1.4.1), (ii) Two Point Adaptation (TPA) [Hansen et al., 2014] and (iii) Median Success Rule (MSR, see Chapter 3) [Ait Elhara et al., 2013]. These three step-size adaptation mechanisms will also be used in SS-CMA-ES (in the internal CMA-ES optimizer) and a variant of VD-CMA-ES with TPA step-size adaptation is also considered.

TPA generates two points in the line between the actual mean solution and its predecessor. These two points are then ranked in the current population. The ranking provides the signal that will be used to adapt the step-size.

The performance comparison is carried on low epsilon-effective dimension problems including the special case of $\varepsilon = 0$ which translates into low effective dimension problems.

4.4.1 Test Functions

In this section, we provide the expressions of the different test functions used in this chapter. In what follows, $\mathbf{x} \in \mathbb{R}^d$ is a solution in the original space, $\mathbf{z} \in \mathbb{R}^{d_{\text{eff}}}$ with $\mathbf{z} = \mathbf{B}^T\mathbf{x}$ is its counterpart in the effective subspace. As in Proposition 1, the function g is chosen to be the normalized sphere function (4.7).

The functions in Table 4.2 are meant to replace the function f in Proposition 1 used to build the f^{Low} part of (4.5) via $f^{\text{Low}}(\mathbf{x}) = f(\mathbf{B}^T\mathbf{x})$ (as in (4.1)) and the resulting function will be named accordingly. For example, $f^{\varepsilon\text{LSphere}}$ is the low epsilon-effective dimension sphere function where, in (4.5), we replace f^{Low} by $f^{\text{LowSphere}}$, $f^{\text{LowSphere}}$ being the function constructed similarly to (4.1) where f is a normalized sphere function (taken form Table 4.2).

Since the tested algorithms are invariant to translations of the search space (that would change $\mathbf{x}_{\text{opt}}$), we keep all the problems centered at zero. Thus, on the Rosenbrock function which, originally, has its optimum in the vector of 1's ($\mathbf{1}_d$), a translation is

$$
\begin{aligned}
f^{\text{LowCigar}}(\mathbf{x}) &= \frac{1}{d_{\text{eff}}}\left(z_1^2 + 10^6 \sum_{i=2}^{d_{\text{eff}}} z_i^2\right) \\
f^{\text{LowCigtab}}(\mathbf{x}) &= \frac{1}{d_{\text{eff}}}\left(z_1^2 + 10^4 \sum_{i=2}^{d_{\text{eff}}-1} z_i^2 + 10^8 z_{d_{\text{eff}}}^2\right) \\
f^{\text{LowDiffpow}}(\mathbf{x}) &= \sqrt{\frac{1}{d_{\text{eff}}} \sum_{i=1}^{d_{\text{eff}}} |z_i|^{\left(2+4\times\frac{i-1}{d_{\text{eff}}-1}\right)}} \\
f^{\text{LowElli}}(\mathbf{x}) &= \frac{1}{d_{\text{eff}}} \sum_{i=1}^{d_{\text{eff}}} 10^{\left(6\frac{i-1}{d_{\text{eff}}-1}\right)} z_i^2 \\
f^{\text{LowRosen}}(\mathbf{x}) &= \frac{1}{d_{\text{eff}}-1} \sum_{i=1}^{d_{\text{eff}}-1} \left(100\left(\hat{z}_i^2 - \hat{z}_{i+1}\right)^2 + \left(1-\hat{z}_i^2\right)\right) \\
f^{\text{LowSphere}}(\mathbf{x}) &= \frac{1}{d_{\text{eff}}} \sum_{i=1}^{d_{\text{eff}}} z_i^2 \\
f^{\text{LowTablet}}(\mathbf{x}) &= \frac{1}{d_{\text{eff}}}\left(10^6 z_1^2 + \sum_{i=2}^{d_{\text{eff}}} z_i^2\right) \\
f^{\text{LowTwoaxes}}(\mathbf{x}) &= \frac{1}{d_{\text{eff}}}\left(10^6 \sum_{i=1}^{\lfloor d_{\text{eff}}/2 \rfloor} z_i^2 + \sum_{i=\lfloor d_{\text{eff}}/2 \rfloor+1}^{d_{\text{eff}}} z_i^2\right)
\end{aligned}
$$

Table 4.2: Table of the low effective dimension functions that are used to compare the large-scale optimization algorithms. The solution $\mathbf{z} \in \mathbb{R}^{d_{\text{eff}}}$ in the effective space is obtained via $\mathbf{z} = \mathbf{B}^T\mathbf{x}$, with $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{B} \in \mathbb{R}^{d\times d_{\text{eff}}}$ an orthonormal matrix. For the Rosenbrock function, using $\hat{\mathbf{z}} = \mathbf{1}_{d_{\text{eff}}} - \mathbf{z}$ allows to have the optimum of f^{LowRosen} at $\mathbf{z} = \mathbf{0}_{d_{\text{eff}}}$, thus coinciding with that of g (the optimum of g is $\mathbf{0}_d$ and its image in the effective space is $\mathbf{0}_{d_{\text{eff}}}$).

applied to the search space, by defining the function on $\hat{\mathbf{z}} = \mathbf{1}_d - \mathbf{z}$, such that the optimum of the shifted problem is in $\mathbf{0}_d$.

As a reminder, Proposition 1 prescribes that the matrices $\mathbf{H}$ of the functions $f(\mathbf{z}) = \frac{1}{d_{\text{eff}}}\mathbf{z}^T\mathbf{H}\mathbf{z}$ used to build f^{Low} need to have eigenvalues that satisfy $\lambda_{\max}(\mathbf{H}) \geq 4d_{\text{eff}}/d$ in order for the corresponding $f^{\varepsilon\text{L}}$ functions to have d_{eff} epsilon-effective dimension. Among the convex-quadratic functions in Table 4.2 (the Rosenbrock and the Sum of Different Powers functions are not convex quadratic), the sphere function has, originally (meaning prior to the normalization by d_{eff}), the smallest largest eigenvalue $\lambda_{\max} = 1$. Hence the condition on $\lambda_{\max}$ (Proposition 1) is satisfied whenever $d_{\text{eff}} \leq d/4$.

In this chapter, we consider the problem dimensions $d \in 2^i, i \in \{4 \cdots 11\}$ and set the corresponding effective dimensions to:

$$
d_{\text{eff}} = \lfloor 3\log_2 d - 10 \rfloor \ , \tag{4.20}
$$

thus, the condition $d_{\text{eff}} \leq d/4$ is satisfied.

The internal complexity (in number of operations) associated to these low epsilon-effective dimension functions is of the order of $d_{\text{eff}} \times d$ while it is linear for the original (without the effective dimension reduction via $\mathbf{B}$) functions. The highest order of complexity comes from the matrix-vector multiplication in $\mathbf{z} = \mathbf{B}^T\mathbf{x}$ and is directly affected by the effective dimension d_{eff}, and how it scales with the problem dimension d. In our case, the choice of d_{eff} in (4.20) leads to a complexity in $O(d\log d)$.

λ	μ	w_i	μ_{eff}
$4 + \lfloor 3 \log d \rfloor$	$\lfloor \lambda/2 \rfloor$	$\frac{\log(\frac{\lambda+1}{2}) - \log(i)}{\mu \log(\frac{\lambda+1}{2}) - \sum_{j=1}^{\mu} \log(j)}$	$\frac{1}{\sum_{j=1}^{\mu} w_j^2}$

Table 4.3: Default parameter values common to all the considered evolution strategies. In the columns: λ is the population size, μ the number of parents, w_i the weight associated to the i^{th} best individual and μ_{eff} the variance effective selection mass.

	c_c	c_s	d_s
CSA	$\frac{4 + \mu_{\text{eff}}/d}{d + 4 + 2\mu_{\text{eff}}/d}$	$\frac{\mu_{\text{eff}} + 2}{d + \mu_{\text{eff}} + 3}$	$1 + 2 \times \zeta(d, \mu_{\text{eff}}) + c_s$
TPA		0.3	$4 - 3.6/\sqrt{d}$
MSR			$2 - 2/d$
VD-CSA		$\frac{\sqrt{\mu_{\text{eff}}}}{2(\sqrt{d} + \sqrt{\mu_{\text{eff}}})}$	$1 + 2 \times \zeta(d, \mu_{\text{eff}}) + c_s$
VD-TPA		0.3	$4 - 3.6/\sqrt{d}$
LM	$1/\lambda$		1

Table 4.4: Default parameter values, follow up to Table 4.3. In d_s, $\zeta(d, \mu_{\text{eff}}) = \max\left(0, \sqrt{\frac{\mu_{\text{eff}}-1}{d+1}} - 1\right)$. The entries of the leftmost columns are to be suffixed by CMA-ES (TPA-CMA-ES, VD-CSA-CMA-ES...etc.). The parameter values for SS-CMA-ES are the same as for default CMA-ES (first three rows) replacing d by d_{ss} (the dimension of the algorithm's search-space).

4.4.2 Parameter and Experimental Settings

Regarding the parameters used by the different algorithms, we start by describing the parameters common to all of them in Table 4.3. The population size is noted λ and the number of parents μ. The weights in the recombination procedure are noted $w_i, i \in [1, \mu]$. In addition, the initial step-size is set to $\sigma_0 = 2$ and the starting point, in the $\mathbb{R}^d$ search space, is set to $\mathbf{x}_{\text{start}} = \mathbf{1}_d$. Thus all algorithms start from the same point which is at a Euclidean distance $\sqrt{d}$ from the optimum.

Then, we show the parameters that are not set to the same values for the different algorithms or step-size adaption mechanisms in Table 4.4. The embedding based algorithm uses the same parameter configuration as CMA-ES. Replacing d by d_{ss}. In fact, these parameters are defined in the search space dimension, and SS-CMA-ES searches in a d_{ss}-dimensioned subspace.

In addition to the parameters in Tables 4.3 and 4.4, we have the comparison index of MSR [Ait Elhara et al., 2013] set to $j = 0.3\lambda$, the target success rate of PSR (Population Success Rule), that is used to adapt the step-size in LM-CMA-ES [Loshchilov, 2014]), set to $z^* = 0.25$ and the number of vector that are stored to represent the covariance matrix equals the population size ($m = \lambda = 4 + \lfloor 3 \log d \rfloor$). Finally, on SS-CMA-ES, we

set:

$$d_{\text{ss}} = 4 \times d_{\text{eff}} = 4 \times \lfloor 3 \log_2 d - 10 \rfloor \ . \tag{4.21}$$

4.4.2.1 Performance Measure

The measure we use to assess the the performance of these stochastic algorithms and how they scale with the problem dimension d is a variation of the estimated Expected Running Time (ERT) [Auger and Hansen, 2005a]:

$$\hat{\text{ERT}} = \frac{\text{AVG}_{\text{success}}}{p_s} \ , \tag{4.22}$$

where $\text{AVG}_{\text{success}}$ is the average number of function evaluations of the successful runs and p_s the success rate (number of successful runs divided by total number of runs). A successful run is a run in which the algorithm evaluates at least one solution which has a fitness better than a given target fitness.

The original expression that can be found in [Auger and Hansen, 2005a] takes into account the average number of function evaluations of the unsuccessful runs too. However, since we lack a solid control on the stopping criteria of the different algorithms, we opted for the formulation in (4.22).

If all runs are unsuccessful, the measure in (4.22) is not defined in which case the ERT is set to infinity (no data point is shown). This might happen for example if the chosen budget or the number of runs are too small or simply because the algorithm does not converge to the defined target value within a reasonable number of function evaluations.

The target values were set to correspond to a certain fitness ($f^{\text{eff}}_{\text{target}}$) in the effective part of the function that we are most interested in and a fitness on the non-effective part (obtained on g) no worse than that of the initial point[4]:

$$f_{\text{target}} = f^{\text{eff}}_{\text{target}} + \varepsilon \times g(\mathbf{x}_{\text{start}}) \ , \tag{4.23}$$

where, in our case, $f^{\text{eff}}_{\text{target}} = 10^{-8}$ will be used for all the ERT based experiments. The idea is to encourage optimization in the effective space without completely disregarding the non-effective part when $\varepsilon > 0$. The case $\varepsilon = 0$ allows to test algorithms on problems where only the effective part matters (low effective dimension problems).

4.4.2.2 Sub-Space dimension d_{ss}, Effective Dimension d_{eff} and ε

Given a problem with low effective dimension $d_{\text{eff}} < d$, the embedding based algorithm, SS-CMA-ES, needs to have an optimization-subspace dimension d_{ss} at least equal to d_{eff} (Theorem 1). This is due to the fact that at least one target solution (a solution with a fitness better than the target value) needs to be in the optimization subspace spanned by $\mathbf{A}$. The latter condition is satisfied via Theorem 1 only when $d_{\text{ss}} \geq d_{\text{eff}}$.

[4] Actually, since the d_{eff} effective dimensions that are optimized are also involved in computing g, the target value should be lower. However, the resulting target being, at most, a factor of 2 lower, we choose to retain this simpler formulation.

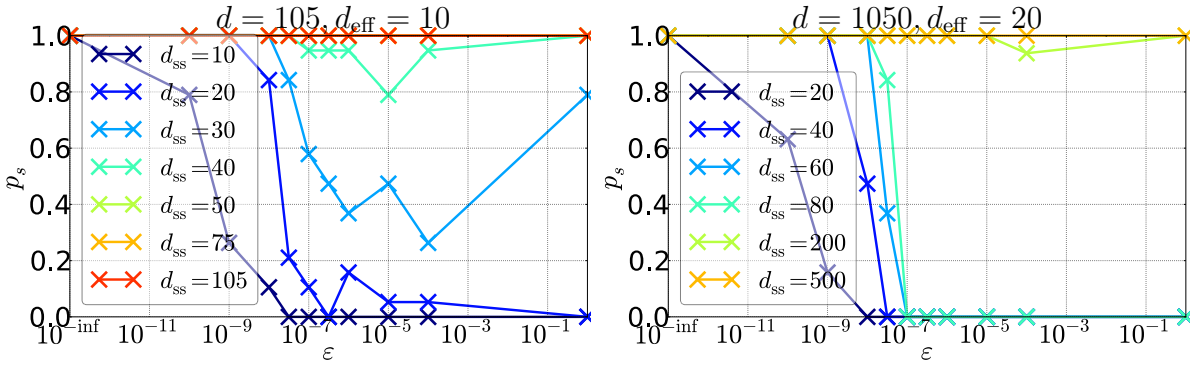

Figure 4.4: Success probability of SS-CSA-CMA-ES on $f^{\varepsilon\text{LSphere}}$ versus different values of ε and d_{ss}. **Left:** $d = 105$, $d_{\text{eff}} = 10$; **Right:** $d = 1050$, $d_{\text{eff}} = 20$. The target values are chosen following (4.23) with $f^{\text{eff}}_{\text{target}} = 10^{-8}$. A total of 19 runs is used for each data point. The leftmost data points, labeled $10^{-\text{inf}}$, are for $\varepsilon = 0$.

Even though in a completely black-box setting the effective dimension is an unknown parameter, we chose to set its values, and more specifically those of d_{ss}, such that the probability of having the optimum in the optimization subspace is at least 0.5. When $\varepsilon = 0$ in (4.5) and $d_{\text{ss}} \geq d_{\text{eff}}$, Theorem 1 insures that this is satisfied. However, for $\varepsilon > 0$, this result is no longer guaranteed since the theorem does not address the case of low epsilon-effective dimension. Thus we choose to set the value of d_{eff} as in (4.20) and choose ε and d_{ss} accordingly.

Figure 4.4 shows the success probabilities of SS-CSA-CMA-ES on $f^{\varepsilon\text{LSphere}}$ for different values of d_{ss} and ε. We see a clear, and expected, monotony in d_{ss}. Regarding ε, we see on the right plots a change of phase for half of the tested values of d_{ss} around $\varepsilon = 10^{-7.5}$ with success probabilities going from 1 to 0. We end up setting $d_{\text{ss}} = 4d_{\text{eff}}$. For ε, in addition to the case $\varepsilon = 0$ and low effective dimension, we choose the values $\varepsilon = 10^{-8}$ and $\varepsilon = 10^{-7.5}$. The target values are, then, respectively, 10^{-8}, 2×10^{-8} and approximately 4×10^{-8}. Note that we make the assumption here that given an optimization subspace, the algorithm manages to find its optimal solution. In other words, we assume that if a target solution is in the optimization subspace, SS-CSA-CMA-ES finds it.

4.4.3 Stopping Criteria on SS-CMA-ES

Using the default stopping criteria of CMA-ES when running RSS-CMA-ES and OSS-CMA-ES resulted, on some occasions, in the algorithms stopping prematurely. This can be seen in Figure 4.5 where the best fitness of the current iteration is plotted against the number of function evaluations of SS-CMA-ES on $f^{\varepsilon\text{LSphere}}$ and $f^{\varepsilon\text{LElli}}$ with $\varepsilon = 10^{-8}$.

We see on both functions that when using the default stopping criteria ($\times$), and in some cases, the algorithm stops sooner than when using the relaxed stopping criteria one ($+$). Furthermore, the final fitness values reached by the relaxed version are, in the cases where we see an improvement, significantly better than those of the default version. In our cases, the relaxation even allows the algorithm to be successful on all runs for both

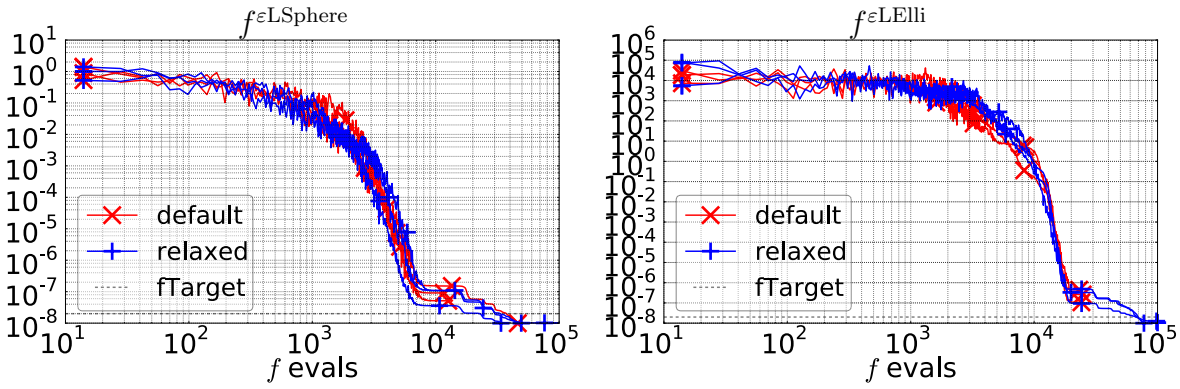

Figure 4.5: Single runs of SS-CMA-ES on $f^{\varepsilon\text{LSphere}}$ and $f^{\varepsilon\text{LElli}}$ with $d = 100$ and $\varepsilon = 10^{-8}$. $\times$: default stopping criteria of CMA-ES; $+$ relaxed stopping criteria as described in Section 4.4.3. The gray dashed line corresponds to the target fitness value that would be obtained using (4.23).

problems when the default version managed to do so only once on $f^{\varepsilon\text{LSphere}}$ (successes is achieved when the fitness passes below the gray dashed line that shows the target value as defined in (4.23)). The default stopping criteria seem to sometimes prevent the algorithm from finding the optimal value in its optimization subspace. So relaxing the stopping criteria seems to be a better, and more efficient, solution than simply restarting the algorithm. Especially if the restarts are done from scratch, thus disregarding the information gathered form the previous runs.

Here, the disabled stopping criteria in question are (range designated the difference between the smallest and largest values within a set):

- the smallest range of the function values within a same generation (*tolfun*),
- the maximal number of iterations with no improvement (*tolstagnation*),
- the minimal range of the best fitnesses over a certain number of iterations (*tolfunhist*).

These disabled stopping criteria are neither used by LM-CMA-ES nor by VD-CMA-ES. Both only stop when reaching the provided target value, exhausting their budget or generating extremely small step-size (10^{-40}).

The original source codes that were used can be found: at PyPI cma 1.1.06 (`https://pypi.python.org/pypi/cma/1.1.06`) for CMA-ES, lmcma (`https://sites.google.com/site/lmcmaeses/`) for LM-CMA-ES and vdcma (`https://sites.google.com/site/youheiakimotospage/pdf/vdcma.m`) for VD-CMA-ES.

Note also that the code of CMA-ES was modified to make it able to handle covariance matrices with larger condition numbers than the original code as several runs on the low epsilon-effective dimension functions stopped because of numerical problems. This is especially the case for low epsilon-effective dimension problems as these have a condition number that is, at least, equal to ε^{-1}. A standard version of $f^{\varepsilon\text{LSphere}}$ (no normalization

by the dimensions) has a condition number ε^{-1} and for $f^{\varepsilon\text{LElli}}$ this value is multiplied by the inherent condition number of the ellipsoid (10^6 in our case) [5].

4.4.4 Single Runs

In order to compare the different algorithms, we start by looking into single runs on some low effective and low epsilon-effective dimension problems.

We are first interested in the behaviors of OSS-CMA-ES and RSS-CMA-ES and whether these two algorithms, as suggested by the results in Section 4.3.3 (solving problems with similar conditioning values), have the same performance, and thus need to be merged into a single algorithm SS-CMA-ES.

We look into the evolution of the best fitness on a current iteration with the number of function evaluations. The right plot of Figure 4.1 compares OSS-CMA-ES to RSS-CMA-ES on $f^{\varepsilon\text{LSphere}}$.

We see no substantial difference in behavior between the two algorithms. The orthonormalization of the $\mathbf{A}$ matrix does not seem to have an effect on the behavior of SS-CMA-ES. Note that we also observe (but not show) this similar behavior on other functions and for other values of ε. This extends the results of Section 4.3.3, where we have seen that both variants deal with similar conditioning values on $f^{\text{LowSphere}}$ to low epsilon-effective dimension problems.

From this point on, we only consider RSS-CMA-ES (computationally cheaper since no orthogonalization is needed), and simply refer to it by SS-CMA-ES. When no step-size adaptation mechanism is mentioned, CSA is used by default.

In Figure 4.6, we see a typical run of SS-CMA-ES on the low epsilon-effective dimension sphere function $f^{\varepsilon\text{LSphere}}$.

We see, in parallel on the four plots, two phases in which the optimization process happens. In the first one (up to a little more than 4000 function evaluations), the fitness and the step-size decrease up to a certain point (around $f(\mathbf{x}) = 2.5 \times 10^{-8}$ and $\sigma = 3 \times 10^{-4}$, see top-left plot) at which point the fitness stagnates while the step-size keeps decreasing. The behavior up to the stagnation point is similar to what is observed in a typical convex-quadratic optimization with a well adapted covariance matrix.

We identify the start point of the second phase as the point where the step-size starts increasing. At this point, the fitness value remains at the same level while the step-size and the scaling of one of the axis of the non-effective dimensions increase. We also notice, in the bottom-left plot, that the d_{eff} effective axes are optimized during the first phase, decreasing as the algorithm converges to the stagnation point. These are the axes that have their scaling values decrease continuously over the course of the optimization. Because of the search space transformations that both of the matrices $\mathbf{A}$ and $\mathbf{B}$ apply (no axis parallel coordinate system), the increase on one of the axis translates into a simultaneous change on all the variables, in coordinate values and in standard deviations

[5] These values are given for the sake of illustration, for the actual values concerning the functions used in this chapter, these condition numbers are also multiplied by d/d_{eff} because of the normalizations by search-space dimension of f^{Low} and g.

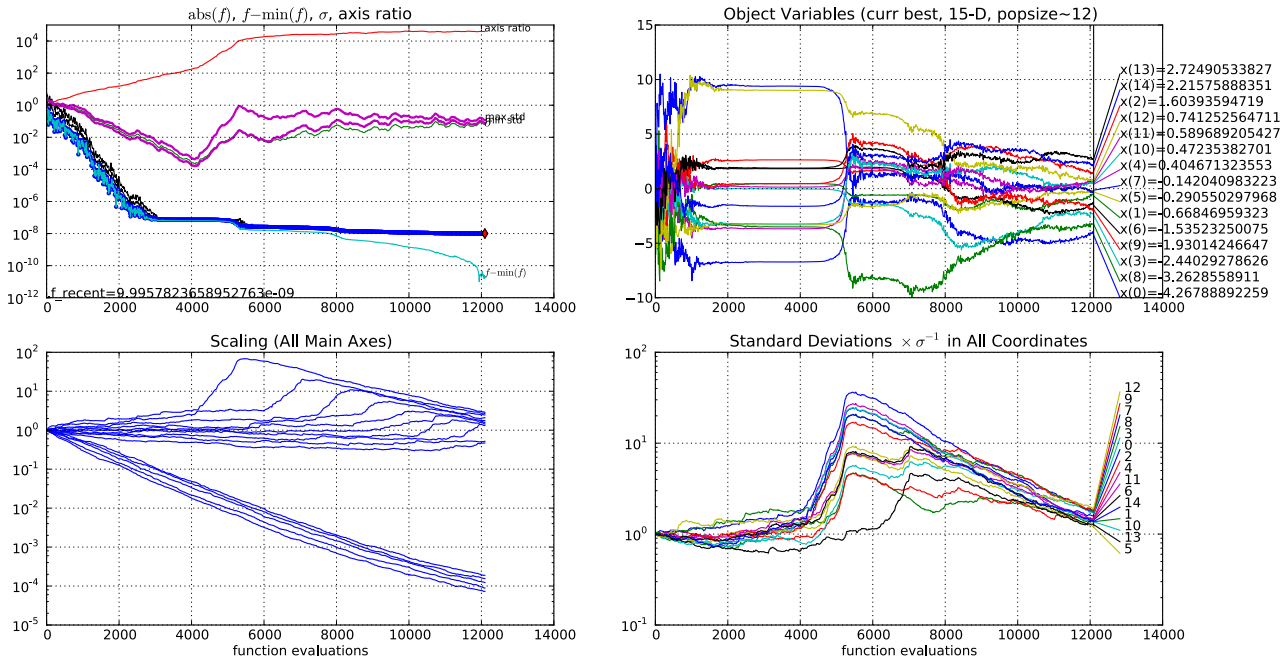

Figure 4.6: Single run of SS-CMA-ES on $f^{\varepsilon\text{LSphere}}$ with $\varepsilon = 10^{-8}$, $d = 50$, $d_{\text{eff}} = 5$ and $d_{\text{ss}} = 15$ showing the number of function evaluations in the x-axis. **Top-Left:** evolution of the fitness, the step-size and the largest axes ratio; **Top-Right:** coordinate values in the optimization subspace; **Bottom-Left:** scaling of the main axes of the covariance matrix; **Bottom-Right:** standard deviation in the optimization subspace coordinates normalized by the current step-size.

as seen in the rightmost plots (upper and lower) of Figure 4.6.

In this second phase, the fitness improvements coincide with *pic* values of the step-size and of the currently explored axis. The algorithm seems to optimize the non-effective space one dimension at a time by increasing the length of its corresponding axis up to a certain value (around 7×10^1 for the first non-effective axis). Once a non-effective axis is optimized (reached its scaling pic value), it follows a similar behavior to that of the effective dimensions and a new non-effective axis is addressed. The pic values of the following axes seem to decrease with each additional addressed non-effective axis. These pic values are upper bounded by those of the previously optimized non-effective axes. The algorithm seems faced with an increasingly difficult task of optimizing in a dimension while not deteriorating the fitness obtained from the optimization of the other dimensions, instead of the standard approach of optimizing variables all at once. A process similar to a line search which dependencies between optimization variables make tedious. This can be seen from the different stagnation values of the optimization subspace coordinates (top-right plot).

The increasing axis ratio (red line in upper left plot), even in the first phase, is due

to the lengths of the non-effective axes being very little changed in the first phase while those of the effective dimension decrease with the decrease in step-size and fitness value. In the second phase, the effective axis keep decreasing and the axis ratio is dictated by the first optimized non-effective axis (which eventually decreases at the same rate, leading to a stationary value of the axis ratio). Basically, the search-space is comprised of d_{eff} effective dimensions and $d_{\text{ss}} - d_{\text{eff}}$ non-effective ones.

We now come to the comparison between the different algorithms. Figure 4.7 shows the evolution of the best fitness of an iteration on problems of dimension $d = 512$ and effective/epsilon-effective dimension $d_{\text{eff}} = 17$ for $\varepsilon = 0$ and $\varepsilon = 10^{-8}$. As previously stated, in addition to comparing with VD-CMA-ES and LM-CMA-ES, we also consider different step-size adaptation mechanisms for CMA-ES and SS-CMA-ES, namely CSA, MSR and TPA. TPA is also applied to VD-CMA-ES and we note the resulting algorithm VD-TPA-CMA-ES. In the default VD-CMA-ES implementation, a modified, more computationally efficient, version of CSA is used (see Section 2.2.1.3).

First, we see that LM-CMA-ES and SS-CMA-ES seem to perform the best overall, with an advantage for LM-CMA-ES on both sphere functions that SS-CMA-ES manages to make up for with the use of TPA or MSR instead of CSA as step-size adaptation mechanisms. Another observation is the greatly improved performance of CMA-ES with the use of TPA (TPA-CMA-ES) and MSR (MSR-CMA-ES) in comparison to CSA (CSA-CMA-ES). The largest such improvement is observed on the two sphere functions (first row). An improvement is also observed for SS-CMA-ES but to a smaller extent. Using TPA in VD-CMA-ES (VD-TPA-CMA-ES), and in comparison to standard VD-CMA-ES, also results in an improvement of the performance on $f^{\varepsilon\text{LSphere}}$, where it actually produces the best results among the algorithms in this study for $\varepsilon = 0$, and on $f^{\varepsilon\text{LRosen}}$ but not on $f^{\varepsilon\text{LElli}}$. In the latter case ($f^{\varepsilon\text{LElli}}$), VD-TPA-CMA-ES stagnates at a relatively high function value (around 1) despite its quicker early stage optimization. Overall, on the low effective dimension functions ($\varepsilon = 0$), the algorithms, with the exception of VD-TPA-CMA-ES, show a converging behavior on the three functions.

The better results observed on CMA, and to a lesser extent on SS-CMA-ES, when using TPA and MSR instead of CSA to adapt the step-size are due to the search-space-dimension dependent comparison that is carried in CSA. In order to adapt the step-size, CSA compares the length of a d (and in the case of SS-CMA-ES, d_{ss}) dimensional vector to the expected length of the same vector had the selection been random. For low effective dimensions problems, only $d_{\text{eff}} < d$ dimensions (the effective dimensions or effective subspace) are relevant while the rest (the non-effective dimensions or non-effective subspace) provide nothing but noise. This results in a slower than desired step-size decrease. The smaller performance gap between CSA and TPA/MSR on SS-CMA-ES, in comparison to CMA-ES, is due to smaller differences between the search-space dimensions (in our case $d_{\text{ss}} = 4 \times d_{\text{eff}} < d$) and the effective dimension d_{eff}, reducing the relative effect of the noise from the non-effective space. This effect is not observed on LM-CMA-ES since it uses the Population Success Rule [Loshchilov, 2014] step-size adaptation mechanism which is similar to MSR. TPA, on the other hand, decreases the step-size faster which is generally the desired behavior.

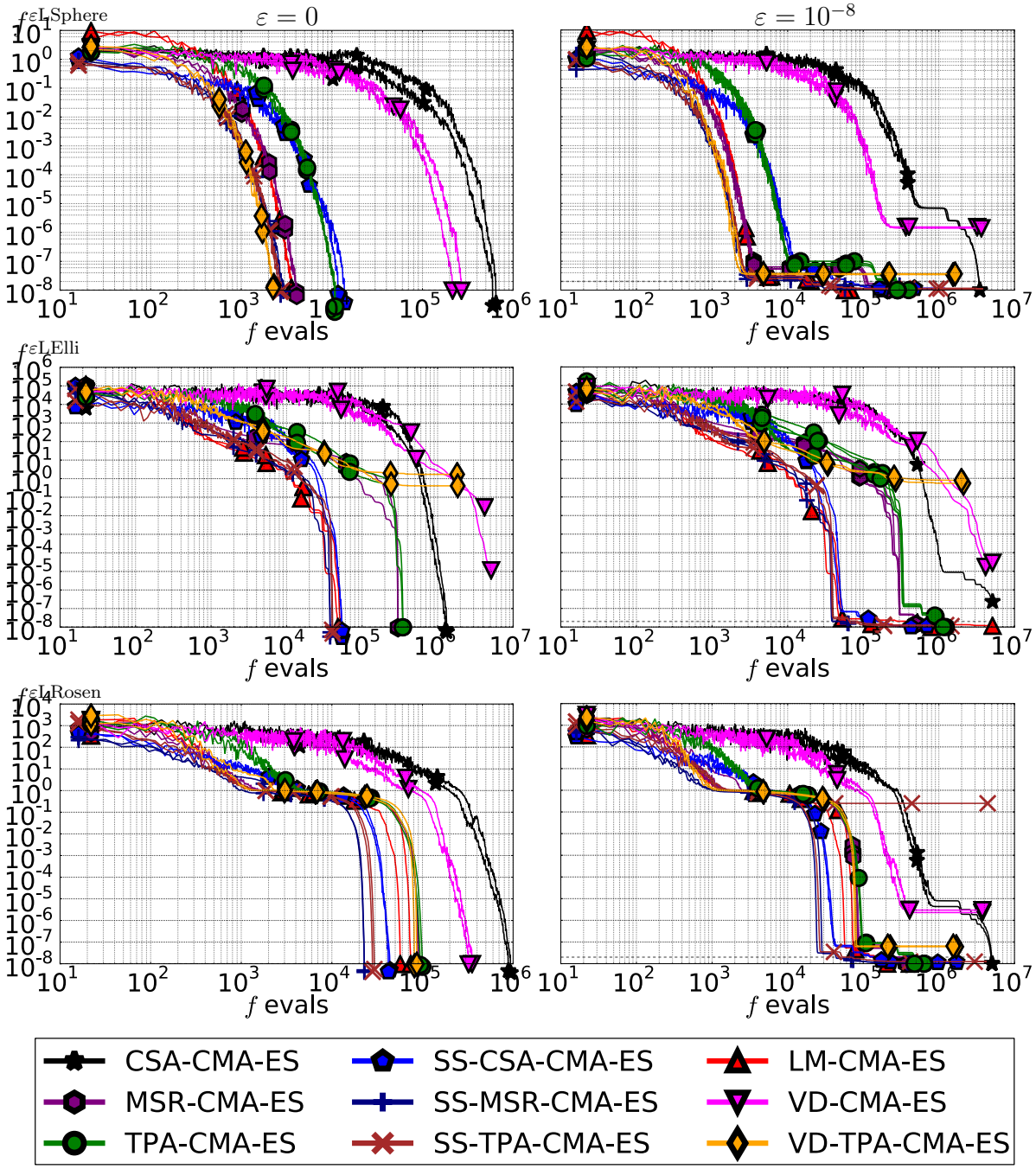

Figure 4.7: Single Runs (two for each) on (from **top** to **bottom**) $f^{\varepsilon\text{LSphere}}$, $f^{\varepsilon\text{LElli}}$ and $f^{\varepsilon\text{LRosen}}$. **Left:** $\varepsilon = 0$; **Right:** $\varepsilon = 10^{-8}$. Evolution of the intra-iteration best fitness on the y-axis for different evolution strategies. The dimension of the problems is $d = 512$, and the effective/epislon-effective dimension $d_{\text{eff}} = 17$. For SS-CMA-ES, the optimization subspace dimension is $d_{\text{ss}} = 4 \times d_{\text{eff}} = 68$. The horizontal dashed gray lines in the right plots show the target value as defined in (4.23).

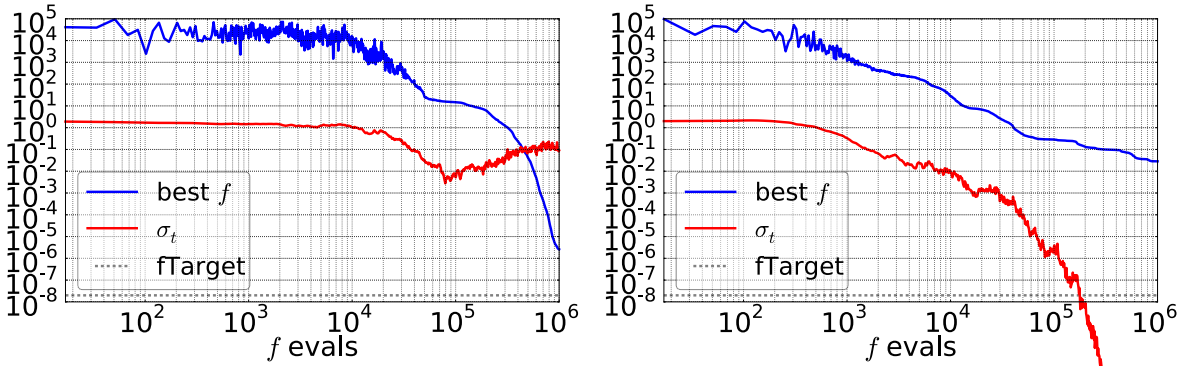

Figure 4.8: Single runs of: Left VD-CMA-ES; Right VD-TPA-CMA-ES on $f^{\varepsilon\text{LElli}}$ with $\varepsilon = 10^{-8}$, $d = 100$ and $d_{\text{eff}} = 9$.

However, when used in VD-TPA-CMA-ES, this results in the algorithm witnessing premature convergence on the ellipsoid functions with the step-size decreasing too rapidly in comparison to the fitness (see Figure 4.8).

When $\varepsilon = 10^{-8}$, two phases are generally observed. A first one where the plots are similar to those of $\varepsilon = 0$ and the second where the fitness improves by jumps followed by what look like plateaus resulting in an overall slow improvement. The second phase seems to start relatively close to the target value (4.23) for most algorithms (compare with the horizontal, gray, dashed lines). The exceptions being the two algorithms that use CSA (VD-CMA-ES and CSA-CMA-ES) for which the slowdown happens at earlier stages. CSA-CMA-ES still manages to reach the designated target values on $f^{\varepsilon\text{LSphere}}$ and $f^{\varepsilon\text{LRosen}}$ within the allocated budget, but at a significantly slower pace.

One important thing to notice is that the target value, as defined in (4.23), is, in all the observed cases, lower, fitness-wise, than the starting point of the second phase. In addition, with small enough step-sizes, and the effective parts being optimized to a certain extent, the variations in the non-effective space become relevant in defining the fitness. We see that LM-CMA-ES, TPA-CMA-ES, MSR-CMA-ES (except for a run on $f^{\varepsilon\text{LRosen}}$ which was stuck on a local optimum) and the different versions of SS-CMA-ES manage to reach the desired target values (4.23). VD-CMA-ES and VD-TPA-CMA-ES, on the other hand, can not solve the problems with $\varepsilon = 10^{-8}$. For $f^{\varepsilon\text{LElli}}$, VD-CMA-ES uses all the budget before reaching the target value, however, on the other functions and for VD-TPA-CMA-ES, the graphs seem to stagnate on higher fitness values. VD-TPA-CMA-ES reaches better stagnation points than VD-CMA-ES on $f^{\varepsilon\text{LSphere}}$ and $f^{\varepsilon\text{LRosen}}$.

In this case of low epsilon-effective dimension ($\varepsilon = 10^{-8}$), the target values (4.23) are only reached after doing some optimization on the non effective subspace. In the second phase, and in order to improve the overall fitness, the non-effective fitness needs to be improved. This must be done without penalizing (too much, ε defining the weight of each part) the already optimized effective space, which results in the slower improvement observed in this phase. The larger step-size values generated by CSA (for both CSA-CMA-ES and VD-CMA-ES) result in *seeing* the non effective part earlier. The random steps in the non effective space that are done in the first phase are of larger magnitude,

resulting in the second phase starting at higher non-effective fitness.

We see that LM-CMA-ES and the different versions of SS-CMA-ES manage to reach the target values on the low epsilon-effective dimension problems, even though the latter only operate on a sub-spaces of the search space, and the problems have full effective dimension. On the other hand, VD-CMA-ES, in its two versions, seems unable to learn an appropriate representation of the Hessian matrix that would allow it to optimize the non effective space.

4.4.5 Scaling with the Optimization Sub-Space Dimension d_{ss}

In this section, we look into the scaling of the performance (ERT) of SS-CMA-ES with the optimization subspace dimension d_{ss}. As seen in Section 4.4.2.2, the success probability (probability of reaching a given target values) is monotonous in d_{ss} on $f^{\varepsilon\text{LSphere}}$. However, and because of the internal complexity of SS-CMA-ES (see Figure 4.1, left plot), the CPU times per function evaluation scale linearly with d_{ss} which means d_{ss} needs to be controlled and ideally minimized.

Figure 4.9 shows the ERTs (normalized by d and d_{ss}) versus d_{ss} on low effective dimension and low epsilon-effective dimension functions given a fixed problem dimension $d = 512$ and effective/epsilon-effective dimension $d_{\text{eff}} = 17$.

First, we notice that the non-normalized ERTs grow with d_{ss} for values of d_{ss} larger than d_{eff}. This can be seen by comparing with the constant scaling lines (negative slope slanted lines). The speeds at which this growth is witnessed differ depending, not only on the function, but also on the step-size adaptation mechanism used.

SS-CSA-CMA-ES shows a scaling in d_{ss} which is larger than linear in all plots (compare with the horizontal lines). It even shows quadratic scaling on $f^{\varepsilon\text{LSphere}}$ when $\varepsilon = 10^{-8}$. In the low effective dimension configurations ($\varepsilon = 0$), SS-TPA-CMA-ES and SS-MSR-CMA-ES show less than linear scalings on $f^{\text{LowSphere}}$ and f^{LowRosen} and a linear scaling on f^{LowElli}. The scaling is even close to being constant for SS-MSR-CMA-ES on $f^{\text{LowSphere}}$. Overall, SS-MSR-CMA-ES and SS-TPA-CMA-ES have a similar scaling in d_{ss} on the tested functions; with a slight advantage for SS-MSR-CMA-ES that is most noticeable on $f^{\text{LowSphere}}$. The three algorithms suffer some failed runs on f^{LowRosen}, with a failure rate decreasing as the optimization subspace dimension d_{ss} increases (no failure among the 3×19 runs with $d_{\text{ss}} = d$). We note that, when $d_{\text{ss}} = d_{\text{eff}}$, the three algorithms show identical ERTs, the differences reported above are only seen when the value of d_{ss} becomes larger than d_{eff}.

When $\varepsilon = 10^{-8}$, the difference in scaling between SS-CSA-CMA-ES and the other two methods becomes smaller as the former seems to preserve a similar scaling to when $\varepsilon = 0$ while the latter end up with larger scalings that are, at best, linear (SS-MSR-CMA-ES on $f^{\varepsilon\text{LSphere}}$). We also see that, and contrarily to the case $\varepsilon = 0$, none of the methods manages to achieve a success when $d_{\text{ss}} = d_{\text{eff}}$.

The increase in ERTs with d_{ss} means that larger values of d_{ss}, and ultimately $d_{\text{ss}} = d$ (which translate into a default CMA-ES plus an orthogonal transformation of the search-space applied by $\mathbf{A}$), are not optimal, not only with regards to the CPU times as seen in

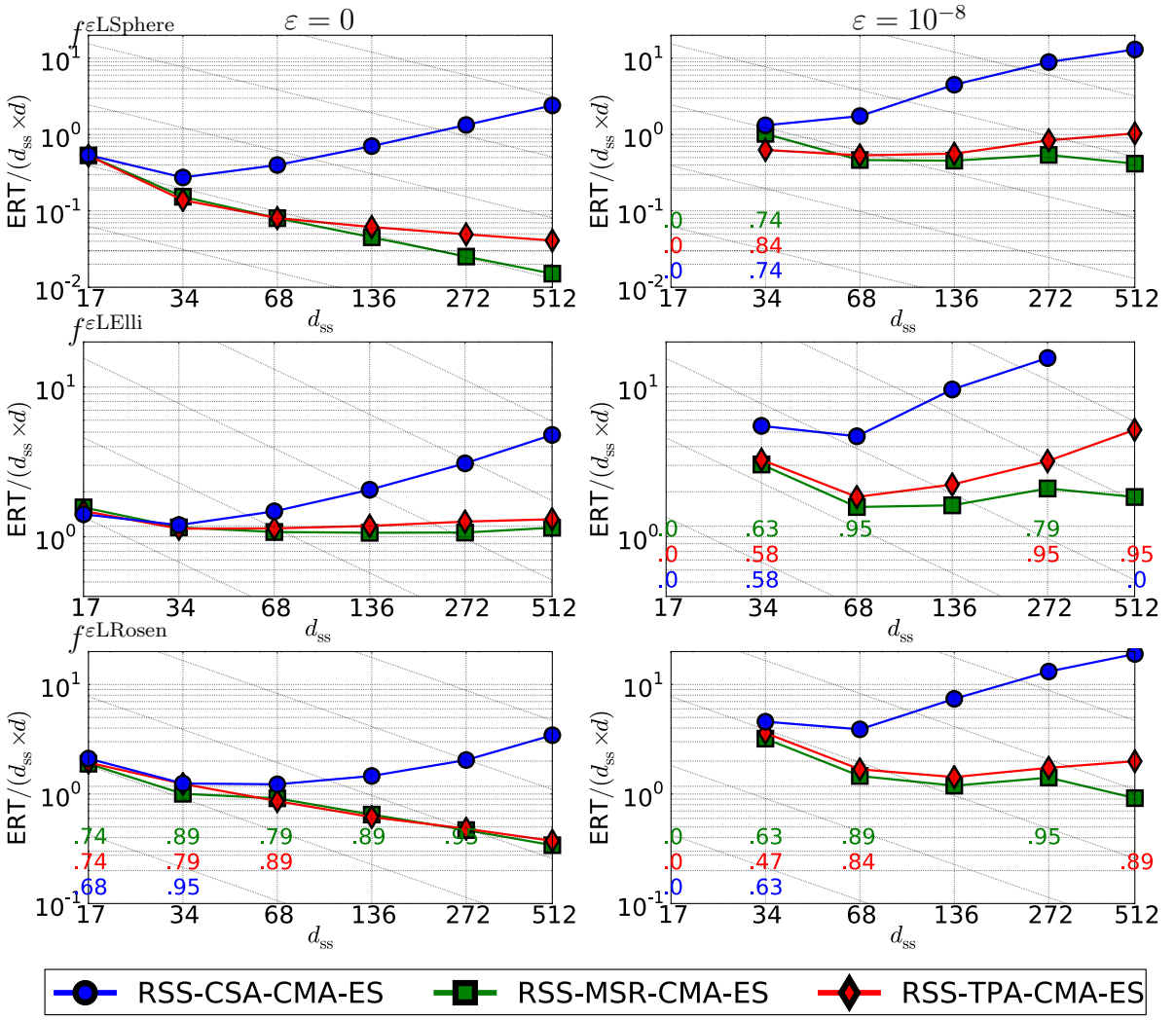

Figure 4.9: Scaling of $\text{ERT}/(d_{\text{ss}} \times d)$ with d_{ss} for SS-CMA-ES with different step-size adaptation mechanisms. From **top** to **bottom**: $f^{\varepsilon\text{LSphere}}$, $f^{\varepsilon\text{LElli}}$ and $f^{\varepsilon\text{LRosen}}$ all with $d = 512$ and $d_{\text{eff}} = 17$. **Left:** $\varepsilon = 0$; **Right:** $\varepsilon = 10^{-8}$. The budget is fixed to $10^4 \times d$, the target values were set following (4.23) and 19 runs are done for each data point. The numbers on the plots, coded by the algorithm colors, show the success rates when lower than 1. The slanted grid lines are for constant and quadratic scalings (with d_{ss}) while linear scaling is represented by the horizontal grid lines.

Figure 4.1 but also in costs in numbers of functions evaluations. This follows from the fact that the larger the search-space of the algorithm (in our case, $\mathbb{R}^{d_{\text{ss}}}$), the more evaluations needed to explore it. Knowing that budgets are generally defined with regards to the problems not the particular algorithms in play (so one would typically define a budget depending on the problem dimension d or even d_{eff} not the optimization space dimension d_{ss}), this further justifies the use of SS-CMA-ES to solve low effective and low epsilon-effective dimensions problems. However, the success rates of the algorithm increase with

d_{ss} (except when failures are due to exhausted budgets and a larger than linear scaling and a linear budget). In fact, the failure, when $\varepsilon = 10^{-8}$, that are not budget related are because of the absence of solutions with good enough fitness (considering the target values) in the optimization search space. The probability of this happening, given a fixed ε, decreases as the optimization search space dimension d_{ss} increases (see Figure 4.4). Failures on f^{LowRosen} are due to the algorithms being trapped in local optima and the absence of a restart strategy. Here again, the failure rates decrease with increasing d_{ss}.

The difference in performance between SS-CSA-CMA-ES and SS-MSR-CMA-ES and SS-TPA-CMA-ES is seen again on the scaling plots. As explained in Section 4.4.4, the larger the difference between the optimization space dimension d_{ss} and the effective dimension d_{eff} of the problem, the larger the effect of the non effective space. We even see that no difference of performance is observed when $d_{ss} = d_{\text{eff}}$ and $\varepsilon = 0$ as the optimization subspace dimension coincides with the effective dimension of the problem. The slower performance of CSA is also present when $\varepsilon = 10^{-8}$ because of the slower early stages of the optimization (see Figure 4.7) when the algorithms focus on the effective dimensions. The difference when compared to MSR and TPA is, however, smaller in this case since the non effective space eventually plays a role in the optimization process, and in the second phase of the optimization, the algorithms try to improve the fitness on a full effective-dimension problem.

When $\varepsilon = 10^{-8}$, the functions no longer have low effective dimension. Their effective dimensions become d but with a low epsilon-effective dimension. No algorithm solves the problems for $d_{ss} = d_{\text{eff}}$ since, as explained above, some optimization in the non effective space (in the second phase) seems needed to reach the target values. The effective space is of dimension d_{ss} so in order to optimize any part of the non-effective space, the algorithms need to have $d_{ss} > d_{\text{eff}}$.

The general picture shows that smaller d_{ss} leads to smaller ERT (in addition to the smaller CPU times). However, the failure rates goes the other way as smaller values of d_{ss} are the most prone to failure (if one excludes insufficient-budget caused failures). We saw this effect more clearly when $\varepsilon > 0$ since the lower bound on d_{ss} for the algorithm to be successful is larger. However, in a complete black-box setting (no information about the effective or epsilon-effective dimension of the problem is provided), the problem is the same regardless of ε since even for $\varepsilon = 0$, $d_{ss} < d_{\text{eff}}$ is bound to result in failed runs. One straightforward strategy in such a setting (unknown d_{eff}) is then to start with small values of d_{ss}, since such value cost little in terms of function evaluations and CPU time Then, if the configuration fails, increase d_{ss}, say by multiplying it by a constant factor (similar to what is done for the population size in [Auger and Hansen, 2005b]) upon each restart.

4.4.6 Scaling with the Problem Dimension d

In order to see how well the large-scale optimization algorithms considered in this chapter scale with the problem dimension, their performance was assessed for different values of d on low effective dimension functions ($\varepsilon = 0$) and low epsilon-effective dimension functions

(with $\varepsilon = 10^{-8}$ and $\varepsilon = 10^{-7.5}$). Since we saw similar results for SS-MSR-CMA-ES and SS-TPA-CMA-ES (except on the sphere functions) in the previous sections, we only consider the latter for both CMA-ES and SS-CMA-ES. We also do not consider CSA-CMA-ES since its performance is dominated by TPA-CMA-ES.

The evolution of the ERT's (normalized by d) versus d is shown in Figures 4.10 and 4.11.

First, note that ($d = 16$, $d_{\text{eff}} = 2$) translates into the functions f^{LowElli}, $f^{\text{LowTablet}}$, f^{LowCigar} and $f^{\text{LowTwoaxes}}$ having the same expression ($f^{\text{Low}}(\mathbf{z}) = (z_1^2 + 10^6 z_2^2)/d_{\text{eff}}$) and $f^{\text{LowCigtab}}$ being similar with a larger condition number (10^8 instead of 10^6). This explains the similarity of the results on the leftmost data points for all algorithms on these functions.

We see that VD-CMA-ES and VD-TPA-CMA-ES do not solve any of the problems with $\varepsilon > 0$ and VD-CMA-ES is dominated, when $\varepsilon = 0$, by the other algorithms. Among the low effective dimensions problems, VD-CMA-ES manages to solve, in addition to $f^{\text{LowSphere}}$, f^{LowCigar}, $f^{\text{LowCigtab}}$ and f^{LowRosen} with what appears to be a linear scaling in d. However, this scaling gets worse on f^{LowElli} and the ERTs are relatively large on $f^{\text{LowTablet}}$. We see no success on dimensions larger that 16 on $f^{\text{LowTwoaxes}}$.

VD-TPA-CMA-ES improves on the performance of VD-CMA-ES $f^{\text{LowSphere}}$ and f^{LowRosen}. Its scaling on $f^{\text{LowSphere}}$ is less than linear, showing, with SS-TPA-CMA-ES, the best overall performance on this function. A significant improvement is also observed for relatively larger dimensions on f^{LowCigar} and $f^{\text{LowCigtab}}$, with the less than linear scaling appearing again. However, on f^{LowElli} and $f^{\text{LowTablet}}$, VD-CMA-ES shows better results as we have already observed on f^{LowElli} in Figure 4.7.

The reason behind the failure of both VD version on low epsilon-effective dimension problems is that its restricted covariance matrix, $\mathbf{C} = \mathbf{D}(\mathbf{I} + \mathbf{v}\mathbf{v}^T)\mathbf{D}$, is unable to learn a good enough approximation of the Hessian matrices of the functions when the non effective part becomes involved. This can be seen for example on $f^{\varepsilon\text{LSphere}}$ where the Hessian is of the form $2(\mathbf{B}\mathbf{B}^T + \varepsilon\mathbf{I}_d)$ with $\mathbf{B}\mathbf{B}^T \neq \mathbf{I}_d$ that can not be modeled in the VD representation.

The same reason is behind the poor performance of VD-CMA-ES on f^{LowElli} and $f^{\text{LowTablet}}$. In fact, the application of $\mathbf{B}$ (a sort of rotation) results in the Hessian being non-representable in the VD framework. However, taking $\mathbf{B} = \mathbf{I}_d$ makes VD-CMA-ES solve both problems in an efficient manner. This inability to solve the rotated ellipsoid and tablet functions was already observed in the original VD-CMA-ES paper [Akimoto et al., 2014]. On the ellipsoid function, the defect of CSA on low effective dimension problems seems to actually have a positive effect on VD-CMA-ES, in comparison to VD-TPA-CMA-ES, by adapting larger step-sizes. This can be seen in Figure 4.8 where the evolution, on a single run, of both the fitness and the step-size on f^{LowElli} is compared for VD-CMA-ES and VD-TPA-CMA-ES. We see that VD-TPA-CMA-ES adapts smaller step-sizes, and seems to be subject to a positive feedback effect making the step-size converge to zero prematurely. Note that the experiments were stopped whenever the step-size reached a value smaller than 10^{-40}.

We see that TPA-CMA-ES, even though not a large-scale specific algorithm, shows reasonably good performance ERT-wise. It manages to solve most of the problems for

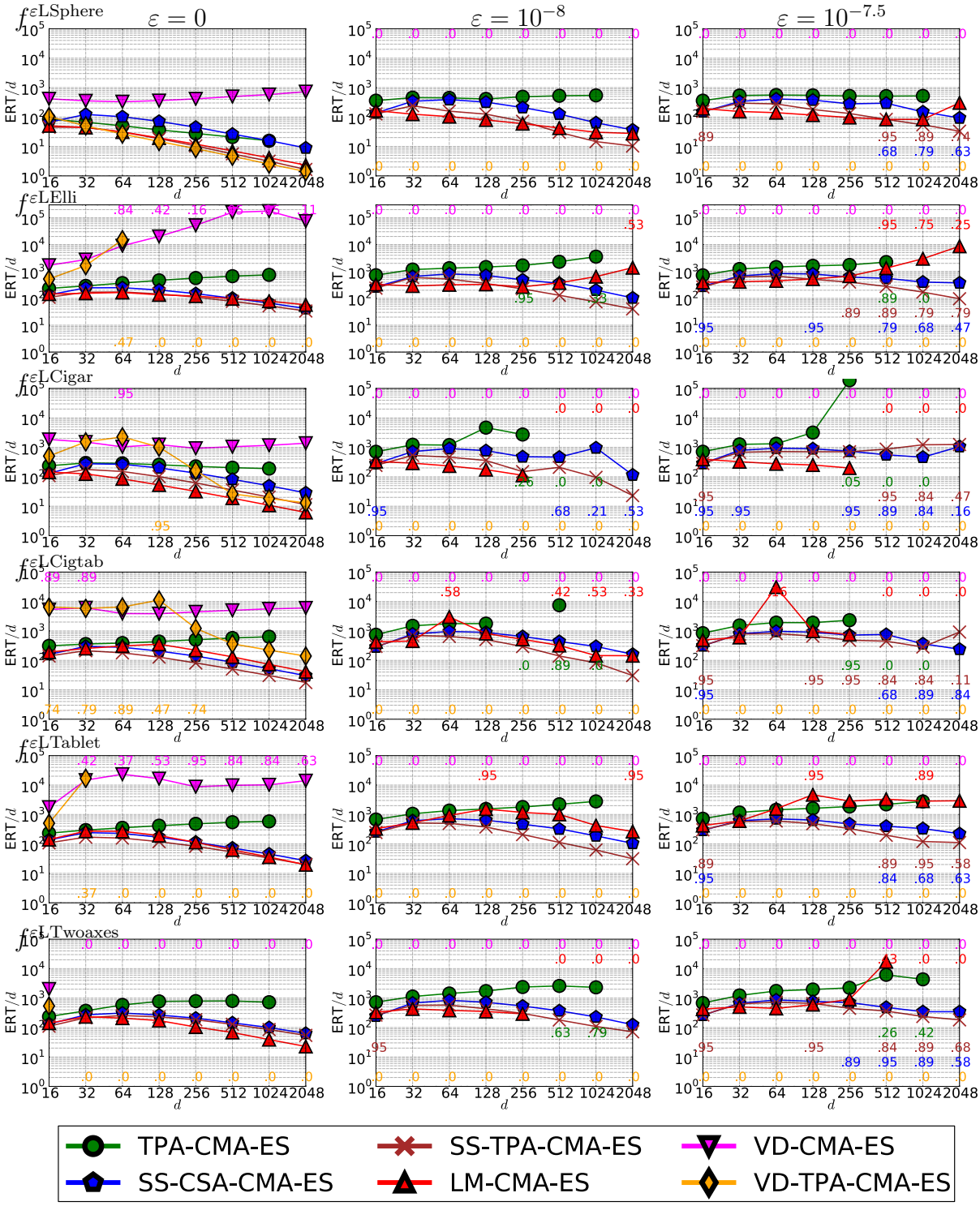

Figure 4.10: Expected Running Time (ERT) divided by problem dimension d versus d on low effective and low epsilon-effective dimension problems on quadratic f^{Low} functions (when replacing in (4.5)). From **left** to **right**: $\varepsilon = 0$, $\varepsilon = 10^{-8}$ and $\varepsilon = 10^{-7.5}$. From **top** to **bottom**: $f^{\varepsilon\text{LSphere}}$, $f^{\varepsilon\text{LElli}}$, $f^{\varepsilon\text{LCigar}}$, $f^{\varepsilon\text{LCigtab}}$, $f^{\varepsilon\text{LTablet}}$ and $f^{\varepsilon\text{LTwoaxes}}$. The budget is $10^4 \times d$ and the target values are chosen following (4.23). A total of 19 runs are used to compute each ERT. The numerical values seen on the plots, coded by the algorithm colors, show the success rates when lower than 1. No runs where done for TPA-CMA-ES on dimension 2048.

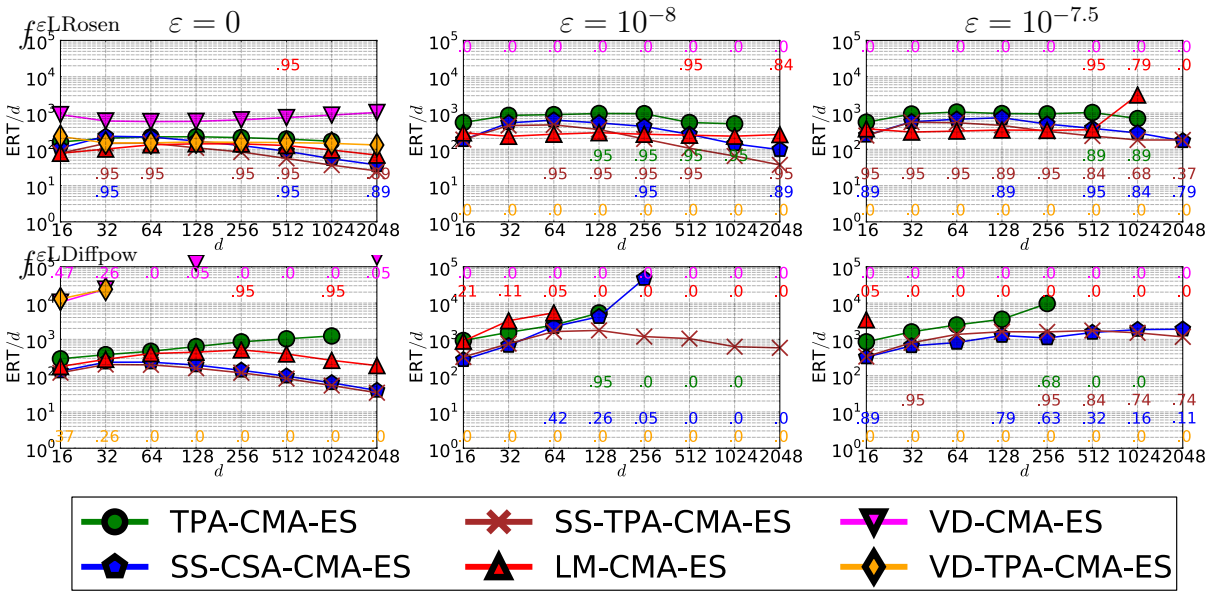

Figure 4.11: Follow up to Figure 4.10 on the non convex-quadratic f^{Low} functions. **Top**: f^{LowRosen}; **Bottom**: $f^{\text{LowDiffPow}}$. For details, see the caption of Figure 4.10.

the three values of ε that are considered and shows less than linear scaling on $f^{\text{LowSphere}}$ but not on any other function. However, it remains dominated by SS-TPA-CMA-ES and even SS-CSA-CMA-ES, and this, despite the CSA defect on low effective dimension problems from which the latter suffers. It also seems to struggle in its scaling, when $\varepsilon > 0$, on functions with a high number of short axes. This is particularly the case on $f^{\varepsilon\text{LCigar}}$ where the scaling becomes worse for $\varepsilon = 10^{-8}$ and $\varepsilon = 10^{-7.5}$ as soon as $d \geq 256$. Its main drawback remains the high internal complexity and CPU time which make it unusable for large scale problems. In our case, $d = 2048$ took too long running times to be included in the results.

Comparing the remaining algorithms, we see that SS-TPA-CMA-ES dominates (although sometimes weakly, in the sens that the differences may be non significant) SS-CSA-CMA-ES on all the functions.

On the low effective dimension functions ($\varepsilon = 0$), LM-CMA-ES and SS-CMA-ES do not show a clear dominance relationship. SS-CMA-ES has better performance on $f^{\text{LowCigtab}}$ and $f^{\text{LowDiffpow}}$ while LM-CMA-ES performs better on f^{LowCigar} and $f^{\text{LowTwoaxes}}$. Both algorithms show less than linear scaling with the problems dimension d on all the functions. These results suggest both methods take into account the low effective dimensions of the problems, and scale accordingly. On SS-CMA-ES, this is to be expected since the algorithm actually operates on a lower dimension d_{ss} which is of the same order than d_{eff} (see (4.21)). LM-CMA-ES on the other hand seems to take advantage of the low effective dimension of the problem in an indirect way. Remember that it does not suffer from the CSA effect explained above thanks to the use of a success based step-size adaptation mechanism. Also, it uses a number of vectors, m, which is in our case in the same order (logarithmic in d) but always larger than d_{eff} to estimate the covariance

matrix.

We see that overall, SS-CMA-ES, and in comparison with LM-CMA-ES, seems to better preserve the scaling that it has for low effective dimensions problems into low epsilon-effective dimensions. This is with the exception of $f^{\varepsilon\text{LCigar}}$ where it becomes linear for $\varepsilon = 10^{-7.5}$. It is also the case for $f^{\varepsilon\text{LDiffpow}}$ as the less than linear scaling is lost and SS-CSA-CMA-ES sees a large increase in ERT for the case $\varepsilon = 10^{-8}$. The most visible example of the better scaling preservation of SS-CMA-ES (if we, again, disregard problems with no successful run) is $f^{\varepsilon\text{LElli}}$ on which LM-CMA-ES sees an increase in the scaling that becomes larger than linear, with a more pronounced effect for larger values of d. On some functions, the difficulty of the problem for LM-CMA-ES seems to increase around $d_{\text{ss}} = 512$ with larger ERTs ($f^{\varepsilon\text{LElli}}$) or sudden failures to solve the problems ($f^{\varepsilon\text{LCigar}}$ and $f^{\varepsilon\text{LTowaxes}}$) due to premature convergence of the step-size (the smallest accepted value is set to 10^{-40}).

This decrease in performance of LM-CMA-ES might be due, as for VD-CMA-ES, to the algorithm being unable to have its restricted covariance matrix representation *match enough* the Hessian of the problems on the second phases (improvement of the fitness by optimizing in the non-effective space) when $\varepsilon > 0$ to actually solve these problems. LM-CMA-ES has overall good results (except on $f^{\varepsilon\text{LDiffpow}}$) and seems dominated by SS-TPA-CMA-ES on functions with a relatively high number of important variables for $\varepsilon = 0$. Here, we mean by important variables the variables in f^{Low} (4.5) with relatively, and in comparison to the other variables in f^{Low}, high associated eigenvalues, not the effective variables of the global function $f^{\varepsilon\text{L}}$. One reason for this is it being unable to learn enough important directions because of the limited number of vectors it uses to estimate the covariance matrix.

For $\varepsilon > 0$, the algorithms require more function evaluations to reach the target values. Even when the target values are larger/easier for larger values of ε (4.23), we have seen in Figure 4.7 that optimization in the non-effective space is needed in order to reach these target values. In addition, this non-effective space optimizations are slower, not only because of the larger number of variables that come into play (all dimensions are effective) in comparison to a low effective function (first phase of the optimization where the non effective spaces are *quasi-invisible* to the algorithm) and the added conditioning (because of ε and the normalizations by d_{eff} and d), but also because of the time needed to re-adapt the representation of the problem since the algorithms see what is basically a new function. We also see larger failure rates of both versions of SS-CMA-ES because of fit enough solutions (given the target value) being absent from the optimization subspace. In fact, we have already seen in Figure 4.4 that the success probabilities decrease with increasing ε.

Some peculiar results are observed when f^{Low} is the non convex-quadratic Sum of Different Powers function (bottom plots of Figure 4.11). In fact, only SS-TPA-CMA-ES solves $f^{\varepsilon\text{LDiffpow}}$ with $\varepsilon = 10^{-8}$ within the allocated budget. In addition, the case $\varepsilon = 10^{-7.5}$ seems less difficult than the case $\varepsilon = 10^{-8}$ for both SS-CSA-CMA-ES and TPA-CMA-ES; the most prominent effect is seen on the former which shows a clearly better scaling of performance when $\varepsilon = 10^{-7.5}$ than when $\varepsilon = 10^{-8}$. This seems to be a result of

the way the target value is chosen (4.23). A higher value of ε means a larger target value (the target value of the case $\varepsilon = 10^{-7.5}$ is more than double that of the case $\varepsilon = 10^{-8}$). When this increase in the target value is not offset by a similar or larger increase in difficulty, the problem with the higher value of ε ends up being easier. We suspect this to be the case on the larger dimensions of $f^{\varepsilon\text{LDiffpow}}$ (smaller ratios between d_{eff} and d). The closer one gets to the optimum, the higher the differences in sensitivity between the variables of this function become (similar to an increase in the condition number if the function were convex-quadratic); thus limiting the impact of a higher value of ε (which also impacts these sensitivities). To backup our assumption that this is a result of the way the target values are chosen, we did some single runs, on this function, of SS-CSA-CMA-ES in dimension 256 with $\varepsilon = 10^{-8}$ and with $\varepsilon = 10^{-7.5}$. The single runs, indeed, show that the algorithm manages to reach lower fitness values ($f(\mathbf{x}^{\text{best}}) \simeq 2.683 \times 10^{-8}$) in the case $\varepsilon = 10^{-8}$ than in the case $\varepsilon = 10^{-7.5}$ ($f(\mathbf{x}^{\text{best}}) \simeq 4.157 \times 10^{-8}$) within the same budget. However, these fitness values reach the target value of the latter scenario ($f_{\text{target}} \simeq 4.162 \times 10^{-8}$) but not that of the former scenario ($f_{\text{target}} \simeq 2 \times 10^{-8}$).

4.5 Discussion

In this chapter, we start by introducing the notions of effective and of epsilon-effective dimensions and showing a way of constructing low effective dimension and low epsilon-effective dimension problems from widely used benchmark functions. The notion of epsilon-effective dimensionality appears as a natural extension of the notion of effective dimensionality. However, to the best of our knowledge, this is the first time this notion is formally expressed for large-scale continuous optimization and used to construct a test set of low epsilon-effective dimension functions. Even though the formalism might seem non-trivial, this appears to be a reasonable way to define functions of low epsilon-effective dimension, confirmed by empirical results. This paves the way for the design of low (ε-)effective dimension problems for benchmarking large-scale continuous optimization algorithms, particularly those algorithms that are designed to handle this class of problems. The following chapter of this thesis will present a different approach to large-scale benchmarking that consists in extending an already existing benchmark for lower dimensions to larger ones. There, the main purpose is to circumvent the large-scale constraints that make the problems of the original benchmark unpractical for use in a large-scale setting while preserving most of the properties of these original problems, including their effective dimensionalities (in our case, full effective dimensionalities).

We, then, introduce SS-CMA-ES, an embedding based algorithm built around the CMA-ES algorithm to optimize low effective dimension problems. Two variants were proposed depending on whether the search subspace spanning matrix is orthogonalized or not. However, the study on the conditioning of the problems and the empirical comparison of the perforamnce of the two variants strongly suggest that orthonormalization does not improve the performance on this specific set of functions.

We also saw that *standard* large scale optimization algorithms deal with these newly introduced problems differently. LM-CMA-ES shows very good performance in compar-

ison to VD-CMA-ES and even to SS-CMA-ES, even though the latter was specifically designed to deal with this kind of problems. However, SS-CMA-ES shows a better robustness with regards to the extension to full effective dimensions and low epsilon-effective dimensions. Another important result observed in this chapter is the effect of non-effective dimensions on the Cumulative Step-size Adaptation (CSA) step-size adaptation mechanism. The non-effective dimensions provide noise that makes CSA adapt the step-size more slowly, expecting to deal with a full effective dimension problem. This, however, is not the case for other step-size adaptation techniques such as Two Point Adaptation (TPA) and the Median Success Rule (MSR) which do not depend directly on the problem dimension.

In most of this chapter, d_{ss} was set to four times the effective dimension of the problem d_{eff}, and we have seen that this value of d_{ss} suffices to see good performance on most of the problems, even when $\varepsilon = 10^{-8}$. However, in a black-box setting, one is not expected to have, beforehand, information on the effective dimension of a problem. In addition, the choice of d_{ss} in this chapter was specifically done to guarantee a certain success-rate of the algorithm in order to assess the performances of SS-CMA-ES. For SS-CMA-ES to be competitive in real world applications, a self-adapting d_{ss} must be introduced. A straightforward solution would be to start with relatively low values of d_{ss} and then increase it, for example by, multiplying d_{ss} by a certain constant factor on each restart. The advantage of this approach is that, as we have seen in Section 4.4.5, low values of d_{ss} result in small running times and higher failure rates, meaning that the budget *wasted* on failed runs because of too small d_{ss} might, in many cases, be smaller than the budget used by standard CMA-ES to solve a low effective dimension problem.

Chapter 5

The COCO Large Scale Suite

In this chapter, we are interested in the extension of a low/medium dimension benchmarking suite to a large-scale one while preserving as much of the properties of this suite and its problems as possible. The suite in question is the BBOB-2009 test suite. The resulting benchmark problems will have, contrarily to the ones presented in the previous chapter, full effective dimensionality. The design of a low (ε-)effective dimension test suite will be left for future work. We propose a sparse and well-scaling orthogonal transformation that can be used as a replacement of full orthogonal (rotation) matrices that are generally used in benchamrking as a generic tool to introduce non-separability and coordinate system-independence. This proposed transformation can be computed in linear time in the problem dimension d and relies on the use of two permutation matrices on either side of an orthogonal block-diagonal matrix. The resulting matrix retains most of the properties of a full orthogonal matrix and allows us to define a practically usable large-scale test-bed based on the BBOB-2009 noiseless test-bed. After including the sparse transformation in the COCO framework, we benchmark three large scale algorithms and analyze their performance.

5.1 The BBOB-2009 Testbed

The 24 problems defined in BBOB-2009 rely on the use of raw (base) functions to which transformations are applied. The notion of *raw function* designates, generally, functions that are defined on the simplest of bases (the canonical base).

Let us consider the Ellipsoid function which is a convex-quadratic function with a diagonal Hessian matrix and eigenvalues uniformly distributed on the log-scale in the interval $[1, 10^6]$. The raw version of the function, on a canonical base, reads:

$$f^{\text{Elli}}(\mathbf{x}) = \sum_{i=1}^{d} 10^{6\frac{i-1}{d-1}} x_i^2 \ , \tag{5.1}$$

where we retain d as the problem dimension. On the other hand, one can consider a different coordinate system that is defined, for example, as a rotation of the canonical

base using a matrix $\mathbf{R}$. Then the corresponding *transformed* (as contrast to a raw function) Ellipsoid function would be:

$$f^{\text{Elli}}(\mathbf{x}) = \sum_{i=1}^{d} 10^{6\frac{i-1}{d-1}} z_i^2 \quad , \tag{5.2}$$

where $\mathbf{z} = \mathbf{R}\mathbf{x}$ and $\mathbf{R}$ an orthonormal matrix.

The raw functions used in benchamrking present, generally, a number of properties that algorithms might be able to exploit. Looking at (5.2), we see that the raw ellipsoid function is separable while its transformed counterpart (5.1) is not (assuming $\mathbf{R}$ is not diagonal). In addition, the function is an even function on the vector level ($f(\mathbf{x}) = f(-\mathbf{x}), \forall \mathbf{x} \in \mathbb{R}^d$) and on the coordinate level ($f(\mathbf{x}) = f(\tilde{\mathbf{x}})$, where $\tilde{\mathbf{x}}$ is the same as $\mathbf{x}$ on all but one coordinate i where $x_i = -\tilde{x}_i$). Furthermore, the optimal solution to (5.1) is $\mathbf{x}_{\text{opt}} = \mathbf{0}_d$ and its corresponding fitness is $f_{\text{opt}} = 0$.

One of the purposes of the transformations used in BBOB-2009 is to *hide* such properties. In addition, parameterized transformations allow to generate multiple instances of a same problem by changing the values of the transformation parameters. The latter is done by setting as seed of the Random Number Generator (RNG) a value that is directly dependent on the instance number. In BBOB-2009, and on most functions, the *initial* seed is: $f_i + 10000 \times i_i$, where f_i is the function index ($1 \leq f_i \leq 24$) and i_i the instance index (in theory, unbounded). Different seeds are sometimes used for the different transformations within a same problem, but these seeds remain deterministically tied to the *initial* seed.

Ideally, instances of a same problem possess the same properties and similar levels of *difficulty*, assuming the notion of difficulty is properly defined. For example, taking different orthonormal matrices $\mathbf{R}$ in (5.2) results in as many different instances of the same, rotated ellipsoid, problem. The instances of (5.2) have the same condition number (since $\mathbf{R}$ is an orthonormal matrix, see Lemma 2), so the difficulty related to the condition number is the same across all instances of this problem. However, the level of non-separability of the resulting problem depends on the structure of $\mathbf{R}$ (full matrix, sparse matrix, band matrix, block-diagonal matrix...).

5.1.1 The BBOB-2009 Transformations

As previously mentioned, the 24 problems of BBOB-2009 are obtained by applying series of transformations on a number of raw functions with relatively simple closed formulae. We now present these transformations and, for each, briefly describe its purpose. We note $f^{\text{new}} : \mathbb{R}^d \to \mathbb{R}^d$ the transformed version of a function f.

5.1.1.1 Shift of Parameter and Fitness Spaces

Looking into the functions defined in [Hansen et al., 2009] and found in Table 5.1 and Table 5.2, we see that most of them have optimal values $\mathbf{x}_{\text{opt}} = \mathbf{0}_d$ and $f_{\text{opt}} = 0$. Such a property can be exploited by algorithms and is artificial in that it does not reflect what

one expects to see in a real-world problem. For example, and on the spherical functions, knowing the distance to the optimum of each solution allows to optimally set the step-size of single parent evolution strategies ((1+1)-ES and (1,λ)-ES)[Auger and Hansen, 2011]. In addition, modeling a real world problem such that its optimal parameter and fitness values are centered in zero suggests these values are known beforehand, reducing the need to resort to the use of black-box optimization algorithms and removing the black-box property of the problem itself.

The optimal solution $\mathbf{x}_{\text{opt}}$ and its fitness f_{opt} are generated randomly and are used such that they become the optimal values of the transformed function:

$$f^{\text{new}}(\mathbf{x}) = f(\mathbf{z}) + f_{\text{opt}} \quad , \tag{5.3}$$

where $\mathbf{z} = \mathbf{x} - \mathbf{x}_{\text{opt}}$. When an algorithm queries a solution $\mathbf{x}$, this solution is first transformed into $\mathbf{z} = \mathbf{x} - \mathbf{x}_{\text{opt}}$, then applied to f. Thus, the optimal solution the algorithm needs to find is $\mathbf{x}_{\text{opt}}$ (since $\arg\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}) = \mathbf{0}_d$). On the other hand, f and f^{new} have the same optimum $\mathbf{x}_{\text{opt}}$ and only differ in the fitness associated to this optimal solution: 0 for the former and f_{opt} for the latter.

A penalization function, f_{pen} is added to the fitness on a number of functions to insure that the optimum is within the domain of interest $[-5, 5]^d$ by penalizing solutions outside of it:

$$f_{\text{pen}}(\mathbf{x}) = \sum_{i=1}^{d} \max\left(0, |x_i| - 5\right)^2 \quad . \tag{5.4}$$

5.1.1.2 Linear Transformations

Linear transformations of the form

$$f^{\text{new}}(\mathbf{x}) = f(\mathbf{M}\mathbf{x}) \quad , \tag{5.5}$$

where depending on the composition and shape (sparsity) of $\mathbf{M} \in \mathbb{R}^{d\times d}$, the transformation is used in order to either introduce non-separability and coordinate system independence or control the condition number of a problem.

Diagonal matrices $\mathbf{M}$ are used to control the condition number of problems, and are noted Λ^α, where the i^{th} diagonal element equals $\alpha^{\frac{1}{2}\frac{i-1}{d-1}}$. On the other hand, the non-separability is introduced using full orthogonal matrices $\mathbf{M}$ in (5.5), noted $\mathbf{R}$ and $\mathbf{Q}$ (some problems require two different orthogonal matrices) in [Hansen et al., 2009] generated using the Gram-Schmidt process on matrices with standard normally distributed entries.

5.1.1.3 Non-Linear Transformations

Two non-linear transformations are defined in [Hansen et al., 2009] and used in order to reduce the regularity and symmetry of the problems.

The raw functions are defined as relatively simple functions of $\mathbf{x}$, or linear transformations of $\mathbf{x}$. The function T^{osz} (5.6) is used to prevent such linearity and regularity in $\mathbf{x}$ by applying a non-linear oscillation:

$$T^{\text{osz}}(x) = \text{sign}(x) \exp(\hat{x} + 0.049(\sin(c_1\hat{x}) + \sin(c_2\hat{x}))) \ , \tag{5.6}$$

where:

$$\hat{x} = \begin{cases} \log(|x|) & \text{if } x \neq 0 \\ 0 & \text{otherwise} \end{cases} \ , \ (c_1, c_2, \text{sign}(x)) = \begin{cases} (10, 7.9, 1) & \text{if } x > 0 \\ (5.5, 3.1, 0) & \text{if } x = 0 \\ (5.5, 3.1, -1) & \text{otherwise} \end{cases} \ , \tag{5.7}$$

On the other hand, T^{asy} introduces a non-symmetry between the positive and negative values of its parameters:

$$T^{\text{asy}}_{\beta}(x) = \begin{cases} x^{1+\beta\frac{i-1}{d-1}\sqrt{x}}, & \text{if } x > 0 \\ x, & \text{otherwise} \end{cases} \ . \tag{5.8}$$

In COCO, T^{asy} is used both on solutions $\mathbf{x} \in \mathbb{R}^d$ and on fitness values that are in $\mathbb{R}$. When used on a solution $\mathbf{x} \in \mathbb{R}^d$, it is applied on each coordinate of that solution, so $T^{\text{asy}}_{\beta}(\mathbf{x}) = \mathbf{x}^{\text{new}} \in \mathbb{R}^d$, where $x_i^{\text{new}} = T^{\text{asy}}_{\beta}(x_i), \forall 1 \leq i \leq d$.

The Rastrigin problem of COCO (f_{15}) is a good example of a problem where all the transformations defined in this section appear. The definition of the raw Rastrigin function, that we note $f^{\text{Rastrigin}}_{\text{raw}}$, can be found in Table 5.1. The transformed Rastrigin function that defines problem f_{15} is:

$$f_{15}(\mathbf{x}) = f^{\text{Rastrigin}}_{\text{raw}}(\mathbf{z}) + f_{\text{opt}} \ , \tag{5.9}$$

where $\mathbf{z} = \mathbf{R}\Lambda^{10}\mathbf{Q}T^{\text{asy}}_{0.2}\left(T^{\text{osz}}\left(\mathbf{R}\left(\mathbf{x} - \mathbf{x}_{\text{opt}}\right)\right)\right)$ and $\mathbf{R}$ and $\mathbf{Q}$ two $d \times d$ orthogonal matrices.

5.2 The Large-Scale Extension

In this chapter, we want to design a computationally reasonable COCO large scale suite based on the BBOB-2009 test-bed [Hansen et al., 2009]. Among the transformations presented in Section 5.1.1, only the application of an orthogonal matrix $\mathbf{R}$ or $\mathbf{Q}$ does not scale well in dimension. The rest scale linearly in d. Thus, iand n order to define the large scale test-bed, we need to replace the full orthogonal matrices with more efficient transformations.

We want the introduced transformation matrix to retain some of the properties of a full-orthogonal matrix without having the shortcoming of quadratic time and space complexities. Thus we define three main criteria that it needs to satisfy in order to be both a reasonable replacement and be applicable in a large-scale setting: (i) a cost criterion, (ii) a non-separability criterion and (iii) an orthogonality criterion.

1. **Cost**: In a large scale setting, both the memory needed to store the matrix and the time complexity needed to apply it to each solution need to scale well with the dimension of the problem. Generally, large-scale algorithms are identified as such thanks to a cost that scales well in the problem dimension. So, to remain usable in practice, the function evaluation of the test-bed should not be a bottleneck cost-wise. Ideally, the cost of applying the transformation should be linear or close to linear ($d^{1+\varepsilon}$ with $\varepsilon \ll 1$), or at most in $d \log(d)$.

2. **Non-separability**: The main purpose of using the orthogonal matrices is to introduce a generic way of generating non-separable problems from separable ones. This is particularly practical since many of the raw functions are separable (see Tables 5.1 and 5.2). Even though we can not reproduce the same *level of non-separability* as a full matrix with a sparse one, we still want the transformed problem to remain non solvable by algorithms that exploits separability, partial separability or particular shapes of the Hessian matrix (when it exists); or, at least, make it costly for algorithms to do so. Ideally, we want to have a parameter/set of parameters that allows to control the level of non-separability of the transformed problem in comparison to the original, non-transformed, problem.

3. **Orthogonality**: Since the matrices $\mathbf{R}$ and $\mathbf{Q}$ are orthogonal, they preserve the eigenvalues and condition number (when they exist) of the original problem (see Lemma 2). Thus, these matrices are not used to change these features of the function, the latter is done via another diagonal matrix Λ^α that can be directly applicable in large-scale because of its linear complexity.

The replacement transformations in our case are *permuted orthogonal block-diagonal matrices* whose costs, in time and space, scale linearly in the problem dimension d and that comply with the above defined criteria.

5.2.1 The Core Transformation Matrix

Before settling on using orthogonal block-diagonal matrices as the core transformation matrix, other sparse matrices were considered. First, we know from the criteria that we defined that diagonal matrices can not work. Not only do they not introduce non-separability, they are simply a special case of the conditioning matrix Λ^α with $\alpha = 1$. Such a matrix being the identity matrix in d ($\mathbf{I}_d$) it has a neutral effect on the problem. One idea of orthogonal sparse matrices is the one used in Chapter 4 in order to define low effective dimension problems. The idea consists in using rank-deficient matrices with a majority of the columns being $\mathbf{0}_d$ vectors. However, in the context of this chapter, we wish to retain as much of the original properties of the noiseless BBOB-2009 test-bed as possible, including the full effective dimensionality. Thus, the suite we define in this chapter will consist of full effective dimension problems.

Another idea was to consider band matrices which are matrices that have all their non-zero elements constrained in a fixed number of the diagonals of the matrix. Let $\mathbf{B}$

be a matrix of size $d \times d$. We say that $\mathbf{B}$ is a band matrix with a lower bandwidth of k_1 and an upper bandwidth of k_2 when for any i, j, $1 \leq i, j \leq d$:

$$i - j > k_1 \text{ or } j - i > k_2 \implies b_{i,j} = 0 \ , \tag{5.10}$$

where $b_{i,j}$ designates the element at the i^{th} row and j^{th} column of $\mathbf{B}$. A tri-diagonal matrix has all its entries at 0 that are not on the main diagonal or the two diagonals adjacent to the main diagonal.

However, band matrices can not be orthogonal unless the band width is 0, and thus are diagonal matrices (contradicts Property 2). To see this, we take the simple example of a 3×3 tri-band matrix:

$$\mathbf{B} = \begin{pmatrix} b_{1,1} & b_{1,2} & 0 \\ b_{2,1} & b_{2,2} & b_{2,3} \\ 0 & b_{3,2} & b_{3,3} \end{pmatrix} \ . \tag{5.11}$$

For $\mathbf{B}$ to be orthogonal, the following system of equations needs to hold:

$$\begin{cases} b_{1,1}b_{1,2} + b_{2,1}b_{2,2} = 0 \\ b_{2,2}b_{2,3} + b_{3,2}b_{3,3} = 0 \\ b_{2,1}b_{2,3} = 0 \end{cases} \ . \tag{5.12}$$

By setting (because of the third equality) either $b_{2,1} = 0$ or $b_{2,3} = 0$, we can propagate the zeros to $b_{1,2}$ or $b_{3,2}$ respectively and end up with $b_{i,j} = 0, \forall i \neq j$ since the band matrix is expected to be full that is all the elements within the band are different from 0 (or replace the $\implies$ in (5.10) by an $\iff$).

This proof can be generalized to any $d \times d, d \geq 3$ tri-diagonal matrix. In fact, setting $b_{2,1} = 0$ means that $b_{i+1,i} = 0, \forall i, 1 \leq i \leq d-1$ in order for the band matrix to remain full. For the same reason, having $b_{1,2} = 0$ means that $b_{i,i+1} = 0, \forall i, 1 \leq i \leq d-1$. As with the 3×3 matrix, we obtain the same result with a similar reasoning by setting $b_{2,3} = 0$ instead $b_{1,2} = 0$ (we have a symmetry around the diagonal).

For a generalized band matrix with k_1 lower bandwidth and k_2 upper bandwidth, when $k_1 + k_2 + 1 \leq d$, the same propagation principle can be used to show that for the matrix to be full-band and orthogonal, we necessarily need $k_1 = k_2 = 0$ (diagonal matrix). Since $k_1 + k_2 + 1 \leq d$, we have, in our system of equations resulting from the orthogonality of the matrix (similar to (5.12)) at least one equation with only one product in the left-hand size:

$$b_{k_1+1,1}b_{k_1+1,j} = 0 \ , \tag{5.13}$$

with $j \leq d$. This equations allows to eliminate either the diagonal $b_{i+k_1,i}$ or the diagonal $b_{i,i+j-k_1}$. Once this is done, another diagonal can be eliminated which is involved in, respectively, either of the following equations:

$$b_{k_1,1}b_{k_1,j-1} + b_{k_1+1,1}b_{k_1+1,j-1} = 0 \ , \tag{5.14}$$

where $b_{k_1+1,1} = 0$, or

$$b_{k_1+2,2}b_{k_1+2,j} + b_{k_1+1,2}b_{k_1+1,j} = 0 \ , \tag{5.15}$$

where $b_{k_1+1,j} = 0$. The process can, then, be carried on until only the diagonal is left.

A special case of (non-full) band matrices that can be non-diagonal and still orthogonal are *block-diagonal matrices*. A block-diagonal matrix $\mathbf{B}$ is a matrix of the form:

$$\mathbf{B} = \begin{pmatrix} \mathbf{B}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_2 & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{B}_{n_b} \end{pmatrix} , \tag{5.16}$$

where $n_b \geq 1$ is the number of blocks and $\mathbf{B}_i, 1 \leq i \leq n_b$ are square matrices of sizes $s_i \times s_i$ satisfying $s_i \geq 1$ and $\sum_{i=1}^{n_b} s_i = d$. The $\mathbf{0}$ designate matrices whose entries are all 0's and that are of the appropriate size in (5.16) (depending on the s_i's).

Property 1 relies upon the number of non-zero entries of $\mathbf{B}$:

$$\sum_{i=1}^{d} \sum_{j=1}^{d} \mathbb{1}_{b_{i,j} \neq 0} = \sum_{i=1}^{n_b} s_i^2 , \tag{5.17}$$

where $\mathbb{1}_{b_{i,j} \neq 0}$ is the indicator function that equals 1 when $b_{i,j} \neq 0$ and 0 otherwise. If we simplify and consider blocks of equal sizes s, bar possibly the last block (in case s does not divide d evenly), then $n_b = \lceil d/s \rceil$. We end up with at most $d \times s$ non-zero entries. Then, in order to satisfy Property 1 with a number of non-zero elements linear in d, s needs to be independent of d. In the case of different block-sizes, the same reasoning can be applied to the largest block-size instead of s. That is we require $\max_{1 \leq i \leq n_b}(s_i)$ to be independent of d.

Concerning Property 2, we only need the matrices $\mathbf{B}_i$ in (5.16) to be non-diagonal in order for the functions that are originally separable to become non-separable since non zero off-diagonal elements introduce correlations between the variables.

For Property 3, given the shape of $\mathbf{B}$ (block-diagonal), it is necessary and sufficient to have orthogonal blocks $\mathbf{B}_i$ for $\mathbf{B}$ to be orthogonal. We generate the matrices $\mathbf{B}_i$ the same way the matrices $\mathbf{R}$ and $\mathbf{Q}$ are generated in [Hansen et al., 2009]. Thus, we end up with each matrix $\mathbf{B}_i$ uniformly distributed in the set of orthogonal matrices of the same size (the orthogonal group $\mathrm{O}(s_i)$). As in [Hansen et al., 2009] and in Section 5.1.1, we first generate square matrices with entries i.i.d. standard normally distributed. Then we apply the Gram-Schmidt process to orthogonalize these matrices. The resulting $\mathbf{B}_i$, and thus $\mathbf{B}$, are orthogonal and satisfy Property 3.

The orthogonal block-diagonal matrice $\mathbf{B}$ is the core of the orthogonal transformation matrix of our large scale test-suite that is to replace full orthogonal matrices. The description above allows us to identify its two parameters:

- d, defines the size of the matrix which coincides with the problem dimension,
- $\{s_1, \ldots, s_{n_b}\}$, the list of block sizes corresponding to, respectively, $\{\mathbf{B}_1, \ldots, \mathbf{B}_{n_b}\}$, where n_b is the number of blocks.

Since the problem dimension is defined by the problem, only the block-sizes remain as a free parameter that needs to bet set.

5.2.2 The Permutations

The block-diagonal matrices defined in Section 5.2.1 introduce dependencies between the variables of each block. However, originally independent variables that do not fall into the same block remain independent. In addition, given the way these blocks are structured, they apply exclusively to adjacent variables. So, if we only use a block-diagonal matrix $\mathbf{B}$, the indexes of the variables would have a direct impact with regards to the application of $\mathbf{B}$.

Such a property can be exploited by algorithms. One way of doing so would, then, be to learn a restricted block-diagonal matrix where the major learning effort would be on the free parameter of the transformation, namely, the block-sizes $(s_i)_{1\leq i\leq n_b}$. In fact, and as we will see further in this chapter, even algorithms that learn an exclusively diagonal matrix such as sep-CMA-ES [Ros and Hansen, 2008] can manage to solve block-diagonal dependencies under some block condition-number conditions.

Ideally, to have a realistic representation of a black-box scenario, the problems we define should not have properties that can be easily exploited by algorithm designers even when these properties are known.

We apply two permutation matrices on both sides of $\mathbf{B}$ and the result completes the transformation matrix as follows:

$$\mathbf{R} = \mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}} \ , \tag{5.18}$$

where $\mathbf{B} \in \mathbb{R}^{d\times d}$ is an orthogonal block-diagonal matrix and $\mathbf{P}_{\text{right}}, \mathbf{P}_{\text{left}} \in \{0,1\}^{d\times d}$ are the two permutation matrices; that is, matrices that have one and only one non-zero element (equal to 1) on each row and each column. A permutation matrix can also be seen as an identity matrix $\mathbf{I}_d$ whose rows/columns were permuted.

The purpose behind applying these two permutations is to *hide* the block-diagonal structure of $\mathbf{B}$ and make it even harder to exploit while retaining the properties granted by the block-diagonal matrix. The additional cost of applying a permutation matrix is linear in the size of this matrix. As we will see in Section 5.6.3, we can simply substitute the permutation matrices by vectors and use these vectors for indexing the variables. The non-separability dictated by Property 2 is satisfied since the number of non-zero entries in $\mathbf{B}$ and in $\mathbf{R}$ is the same, meaning that $\mathbf{R}$ can not be diagonal, thus it necessarily introduces dependencies between the variables. Finally, Property 3 remains satisfied as permutation matrices are orthogonal, and the product of orthogonal matrices is an orthogonal matrix. So $\mathbf{R}$ in (5.18) is an orthogonal matrix that preserves the eigenvalues and condition number of the functions it is applied on (when existent).

The effect of a permutation matrix on another matrix depends on the order of the matrices in the product. In (5.18), $\mathbf{P}_{\text{left}}$ shuffles the rows of $\mathbf{B}$ while $\mathbf{P}_{\text{right}}$ shuffles its columns. In order to better understand the effect of $\mathbf{P}_{\text{left}}$ and $\mathbf{P}_{\text{right}}$ on a function, we suppose a separable quadratic function f and transform it using $\mathbf{R} = \mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}$ into $f_{\mathbf{R}}$, that is define [Ait Elhara et al., 2016]:

$$f_{\mathbf{R}}(\mathbf{x}) = f(\mathbf{z}) = \mathbf{z}^{\mathsf{T}}\mathbf{D}\mathbf{z} \ , \tag{5.19}$$

where $\mathbf{D}$ is a diagonal matrix and $\mathbf{z} = \mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}\mathbf{x}$. By replacing $\mathbf{z}$ in (5.19), we obtain

$$\begin{aligned} f_{\mathbf{R}}(\mathbf{x}) &= (\mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}\mathbf{x})^{\mathsf{T}}\,\mathbf{D}\mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}\mathbf{x} \\ &= \mathbf{x}^{\mathsf{T}}\mathbf{P}_{\text{right}}^{\mathsf{T}}\mathbf{B}^{\mathsf{T}}\mathbf{P}_{\text{left}}^{\mathsf{T}}\mathbf{D}\mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}\mathbf{x} \\ &= \tilde{\mathbf{x}}^{\mathsf{T}}\mathbf{B}^{\mathsf{T}}\tilde{\mathbf{D}}\mathbf{B}\tilde{\mathbf{x}} \\ &= \tilde{\mathbf{z}}^{\mathsf{T}}\mathbf{D}\tilde{\mathbf{z}} \\ &= f(\tilde{\mathbf{z}}) \ , \end{aligned} \tag{5.20}$$

where $\tilde{\mathbf{x}} = \mathbf{P}_{\text{right}}\mathbf{x}$, $\tilde{\mathbf{D}} = \mathbf{P}_{\text{left}}^{\mathsf{T}}\mathbf{D}\mathbf{P}_{\text{left}}$ and $\tilde{\mathbf{z}} = \mathbf{B}\tilde{\mathbf{x}}$. Thus, if we look into the equivalent problem that was transformed by only $\mathbf{B}$ (without the permutations): $\mathbf{x}^{\mathsf{T}}\mathbf{B}^{\mathsf{T}}\mathbf{D}\mathbf{B}\mathbf{x}$, we see that the permutations simply replace $\mathbf{x}$ by $\tilde{\mathbf{x}} = \mathbf{P}_{\text{right}}\mathbf{x}$ and $\mathbf{D}$ by $\tilde{\mathbf{D}} = \mathbf{P}_{\text{left}}^{\mathsf{T}}\mathbf{D}\mathbf{P}_{\text{left}}$. Thus, the permutation $\mathbf{P}_{\text{right}}$ shuffles the variables that the optimization algorithm sees and queries. The performance of a good optimization algorithms is expected to be invariant to this permutation, since the *order* in which the variables are presented should not be relevant. On the other hand, $\mathbf{P}_{\text{left}}$, by applying to $\mathbf{D}$, shuffles the variables in which the raw function is defined, that is the variables of the function before applying $\mathbf{R}$. As a result, and coupled with $\mathbf{B}$, this permutation determines which coefficients are used within the blocks defined by $\mathbf{B}$ and, consequently, the block condition numbers and the difficulty of the problem (see the Section 5.4).

5.2.2.1 Generating the Random Permutations

In what follows, we call a *swap* the exchange of positions of two variables. A swap involves two variables, a first swap variable, i, and a second swap variable, j, that will (the second swap variable j) generally depend, in some way, on the choice of the first swap variable. Thus, a permutation is the order obtained after applying a given number of swaps, that we will note n_s. Ideally the parameterization of the permutations should allow to have a certain level of control over the difficulty of the resulting problem. Since we define a permutation as a succession of swaps, the first parameter of the permutation would be the number of swaps n_s. Then, it will also has any parameter that intervenes in the swap strategy.

A first simple idea would be to apply a random uniformly chosen permutation of the variables, which is equivalent to swapping each of the $d-1$ first variables once with a random uniformly chosen second swap variable $j \sim \mathcal{U}(\{1,\dots,d\})$, where $\mathcal{U}(S)$ designates the uniform distribution in the set S. The number of random uniform swaps one applies to obtain the final permutation is a parameter that we can vary. However, the problem with this approach is that the distance each variable *travels* is subject to a high amount of variance for a low number of swaps (variance is maximized on a uniform distribution). In fact, in the context of applying $\mathbf{B}$ too, and depending on which variables are swapped, the resulting problem can differ substantially. Swapping variables which belong to the same block has a significantly smaller effect on a separable problem than swapping variables that do not (belong to the same blocks).

In order to reduce this variance, we considered other strategies where, instead of having $j \sim \mathcal{U}(\{1,\dots,d\})$, we make j follow a Zipf-like distribution. That is a discrete

power law where the probability of each element is inversely proportional to its rank. In our case, we define the rank of a value j by its distance to i and generalize the distribution with a parameter α giving the distribution of j:

$$\Pr\left(j = \tilde{j} | i = \tilde{i}\right) = \begin{cases} \frac{1}{|\tilde{i} - \tilde{j}|^{\alpha}} \times \frac{1}{\sum_{k=1, k|\neq \tilde{i}}^{d} |\tilde{i} - k|^{-\alpha}} & \text{if } \tilde{j} \in \{1, \ldots, d\} \text{ and } \tilde{j} \neq \tilde{i} \\ 0 & \text{otherwise} \end{cases} , \tag{5.21}$$

where the parameter α sets the penalization of the distance to the first swap variable. This approach was also tried in a variant where we first sample the blocks of the variables to be swapped similarly to (5.21) and then chose the variables within them uniformly at random.

The main problem with the Zipf-like distribution based approach was the relatively high number of swaps needed to obtain problems that are, separability-wise, significantly different from the raw problems transformed by **B**.

In order to introduce more change in the shape of the block-matrix while keeping control of this amount of change, we want each variable to be moved, in average, a fixed distance. That is, in the produced permutation (after the n_s swaps are done), we want to have variables in positions which are, in average, a fixed distance away from their starting positions. In order to reduce the number of swaps needed to obtain a significant change in the shape of the block-matrix, we go back and use uniform distributions of the second swap variables. However, and in order to retain control over these *travel* distances, we introduce a parameter to the uniform distribution, the swap range r_s that delimits the support of the uniform distribution. We call this *truncated uniform swaps*.

For a given number of variables (problem dimension) d and a swap range r_s and given i, the first swap variable, j follows the distribution:

$$j \sim \mathcal{U}\left(\{l_b(i), \ldots, u_b(i)\} \setminus \{i\}\right) \quad , \tag{5.22}$$

where $l_b(i) = \max(1, i - r_s)$ and $u_b(i) = \min(d, i + r_s)$. In all what follows, real valued swap ranges are rounded down to the closest integer (the fractional part is ignored).

Roughly speaking (if we consider an infinite number of variables), each variable should travel, in average, around $r_s/2$ from its initial position. More precisely, we need to take into account the borders imposed by the finite number of variables and the absence of a circular placement of the variables (the first and last variables are not considered neighbors and have a distance of $d - 1$ between them). Let us note the average distance traveled by a variable i $\text{avg}_{\text{dist}}(i)$. If we suppose $r_s \leq (d-1)/2$, and consider the case $i - 1 < d - i$, that is the first swap variable is closer to the first variable (or is this first swap-variable) than to the last variable (we can obtain the other case by symmetry) and

that $i < r_s + 1$ (the value is otherwise straightforward to compute):

$$\begin{aligned}
\text{avg}_{\text{dist}}(i) &= \frac{1}{r_s + i - 1} \sum_{k=1, k\neq i}^{i+r_s} |i - k| \\
&= \frac{1}{r_s + i - 1} \Big(\sum_{k=1}^{r_s} k + \sum_{k=1}^{i-1} k \Big) \\
&= \frac{1}{2\,(r_s + i - 1)} \Big(r_s\,(r_s + 1) + i\,(i - 1) \Big) \\
&= \frac{1}{2\,(r_s + i - 1)} \Big(i^2 - i + r_s^2 + r_s \Big) \ .
\end{aligned} \tag{5.23}$$

Then, we investigate the values of i that minimizes this average distance by deriving with regards to i:

$$\begin{aligned}
\frac{\partial\, \text{avg}_{\text{dist}}}{\partial i} &= \frac{1}{2(r_s + i - 1)^2} \Big((2i - 1)(r_s + i - 1) - (i^2 - i + r_s^2 + r_s) \Big) \\
&= \frac{1}{2(r_s + i - 1)^2} \Big(i^2 + 2(r_s - 1)i - (r_s^2 + 2r_s - 1) \Big) \ .
\end{aligned} \tag{5.24}$$

The discriminant of the numerator $\Delta = 8r_s^2 \implies \sqrt{\Delta} = 2\sqrt{2}r_s$. Thus the solutions: $i_1 = 1 - (\sqrt{2} + 1)r_s < 0$ that we discard and $i_2 = 1 + (\sqrt{2} - 1)r_s$ that we retain and which is the index that minimizes the average distance since the derivative is negative for values smaller than i_2 and positive for values larger than i_2.

We replace in the expression of $\text{avg}_{\text{dist}}(i)$ in (5.23) and find the minimal value:

$$\min(\text{avg}_{\text{dist}}(i)) = (\sqrt{2} - 1)r_s + \frac{1}{2} \ . \tag{5.25}$$

Given the behavior of the derivative, the maximal value is reached in the extremes of the interval ($i = 1$ or $i = d$) or when i is at least r_s away from the extremes and reads (replacing in (5.23):

$$\max(\text{avg}_{\text{dist}}(i)) = \frac{1}{2}r_s + \frac{1}{2} \ . \tag{5.26}$$

This leaves us with average distances in the interval $[(\sqrt{2} - 1)r_s + \frac{1}{2}, \frac{1}{2}r_s + \frac{1}{2}]$ which has a size smaller than 10% of r_s.

Algorithm 3 describes the process of generating a permutation vector $\mathbf{p}$ using a series of the truncated uniform swaps described in (5.22).

The parameters used in Algorithm 3 are:

- d, the number of variables, which does not need to be set since it is defined by the problem
- n_s, the number of swaps. We choose values proportional to d to make the last parameter the only free one,

Algorithm 3 Truncated Uniform Permutations [Ait Elhara et al., 2016]

Inputs: problem dimension d, number of swaps $n_s \leq d$, swap range r_s.
Output: a vector $\mathbf{p} \in \mathbb{N}^d$, defining a permutation.

1: $\mathbf{p} \leftarrow (1, \ldots, d)$
2: generate a uniformly random permutation π
3: **for** $1 \leq k \leq n_s$ **do**
4: $\quad i \leftarrow \pi(k)$, $x_{\pi(k)}$ as first swap variable
5: $\quad l_b \leftarrow \max(1, i - r_s)$
6: $\quad u_b \leftarrow \min(d, i + r_s)$
7: $\quad S \leftarrow \{l_b, \ldots, u_b\} \setminus \{i\}$
8: $\quad$ Sample j uniformly in S
9: $\quad$ swap p_i and p_j
10: **end for**
11: return $\mathbf{p}$

- r_s, the swap range and eventually the only free parameter. The swap range can be equivalently defined in the form $r_s = \lfloor r_r\, d \rfloor$, with $r_r \in [0, 1]$ ($r_s = 0$ is interpreted as no swaps). Given r_s, the average distance traveled by a variable in a single swap is delimited by (5.25) and (5.26).

In Algorithm 3, we use the random permutation π to take the indexes of the variables in a random order and avoid any bias with regards to which variables are selected as first swap variables when less than d swaps are carried. We start with $\mathbf{p}$ initially the identity permutation, then we apply the swaps defined in (5.22) taking $p_{\pi(1)}, p_{\pi(2)}, \ldots, p_{\pi(n_s)}$, successively, as first swap variable (that replace i in (5.22)). This way, as long as $n_s \geq d$, each variable is guaranteed to be swapped at least once. Values of n_s larger than d can be dealt with by cycling with a new random order of the variables π. The resulting vector $\mathbf{p}$ is returned as the desired permutation.

A different implementation was considered where Line 9 in Algorithm 3 swaps, instead, p_i^{-1} with p_j^{-1}, where $\mathbf{p}^{-1}$ is the, dynamically updated, inverse permutation of $\mathbf{p}$. However, this variant has shown, empirically, no significant differences to the version implemented in Algorithm 3, so we only consider the latter.

5.3 Transformation-Parameter Impact

Now that we have decided upon the transformation matrix that we want to apply to the raw functions, namely a permuted orthogonal block-diagonal matrix (5.18), and identified its free parameters, $(s_i)_{1 \leq i \leq n_b}$, n_s and r_s, we are interested in how these parameters, in addition to the imposed problem dimension d, affect the transformation matrix and more generally, the transformed problem.

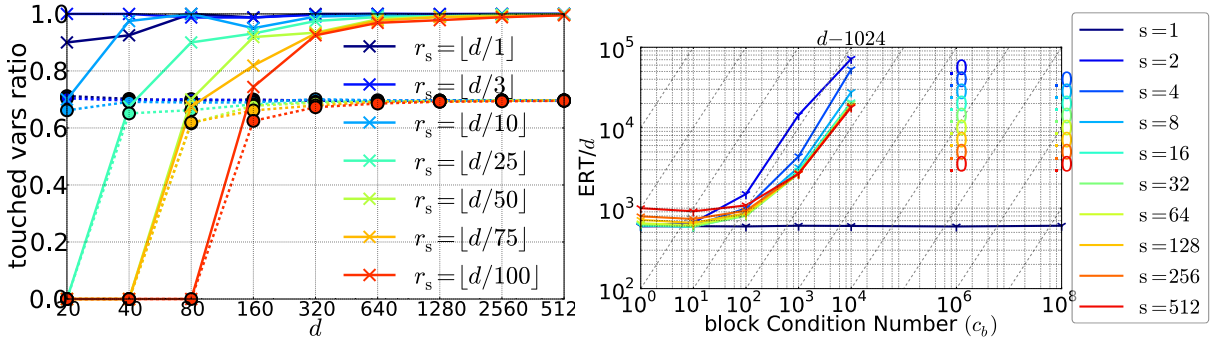

Figure 5.1: **Left:** Proportion of variables that are moved from their original position in $\mathbf{p}$ ($p_i \neq i$) when applying Algorithm 3 versus problem dimension d. Each graph corresponds to a different swap range r_s indicated in the legend. **Solid lines:** $n_s = d$; **Dashed lines:** $n_s = d/2$. The case $r_s = 0$ means no swap is performed ($\mathbf{p} = (1, \ldots, d)$). The averages of 51 repetitions per data point are shown. Standard deviations (not shown) are at most 10% of the mean. **Right:** ERT of sep-CMA-ES divided by dimension on the function defined in (5.36) versus block condition number c_b for different values of the block size s shown in the legend. The overall condition number of the problem is set to $c_o = 10^8$, the problem dimension to $d = 1024$ and the target value for computing the ERT is 10^{-8}. Three runs are done per data point, each with a budget of $10^5 \times d$ function evaluations. The numbers with the color codes represent the success rates of the corresponding configurations in s (same color) when these are lower than 1. The dashed gray lines show a linear scaling.

5.3.1 Impact of the Number of Swaps on the Proportion of Moved Variables

Looking into Algorithm 3, we see a possibility for the final permutation to contain variables that remain in their positions ($p_i = i$) or end up further than r_s away from their original position ($|p_i - i| > r_s$). This is due to the fact that variables can be involved in swaps more than once (or, more precisely, more than $\lceil n_s/d \rceil$ times) by being second swap variables. Ideally, we want to guarantee than most of the variables move from their original positions.

In the left plot of Figure 5.1, we look into the proportion of variables that are affected by the swap strategy and end up in positions different from their initial ones.

We see that as the problem dimension d increases, the ratio of moved variables ($p_i \neq i$) approaches what seem to be stationary values at 100% and 70% for respectively $n_s = d$ and $n_s = d/2$ swaps. These stationary values are independent of the relative swap range r_r (we remind that $r_s = \lfloor r_r d \rfloor$), except for relatively small dimensions. This is particularly the case when $n_s = d$. This probability is also affected by the actual value of the swap range r_s: the larger the swap range, the smaller the probability of a variable being swapped back to its original position. This is due to the fact that the probability of swapping a variable back to its original position increases as the swap range decreases

since the support of the truncated uniform distribution is smaller. Thus, for large enough dimensions (which are the values we are most interested in in our context), only few variables are left unmoved. We decide to fix the number of swaps to $n_s = d$, leaving only r_s as a free parameter when generating the permutation matrices $\mathbf{P}_\mathrm{left}$ and $\mathbf{P}_\mathrm{right}$.

5.3.2 Impact of the Parameters on the Structure of the Transformation Matrix

In this part, we are interested in the effect of $(s_i)_{1\leq i\leq n_b}$, r_s and d on the shape of the sparse orthogonal matrix defined in (5.18). To this end, we define a measure: *sum of distances to the diagonal of the non-zero elements* and investigate its evolution with the aforementioned parameters. This measure, (d_ToD) is defined as follows:

$$d_\mathrm{ToD}(\mathbf{R}) = \sum_{i=1}^{d}\sum_{j=1}^{d} \mathbb{1}_{r_{(i,j)}\neq 0}|i-j| \ , \tag{5.27}$$

where $r_{(i,j)}$ is element (i,j) of $\mathbf{R}$ and $\mathbb{1}_{r_{(i,j)}\neq 0}$ is the indicator function on the condition $r_{(i,j)} \neq 0$.

Our reasoning is that this measure allows us to estimate how *different* from a diagonal matrix the transformed matrix is, which gives us an idea of the non-separability of the resulting problem (since a diagonal transformation matrix introduces no non-separability).

In Figure 5.2, we show a normalized d_ToD of a simplified matrix $\mathbf{R} = \mathbf{P}_\mathrm{left}\mathbf{B}$ with blocks of equal sizes. This normalized distance is plotted against the values of d, for different values of s and r_s.

For a full square matrix of size d, we compute the corresponding d_ToD by replacing in (5.27) and find:

$$d_\mathrm{ToD}(\mathrm{full}) = \frac{1}{3}d(d-1)(d+1) \ . \tag{5.28}$$

From this equation, we deduce the value of d_ToD for a block-diagonal matrix $\mathbf{B}$ with blocks of equal sizes s:

$$d_\mathrm{ToD}(\mathbf{B}) = \left\lfloor \frac{d}{s} \right\rfloor \frac{s}{3}(s-1)(s+1) + \frac{1}{3}c(c-1)(c+1) \ , \tag{5.29}$$

where $c = d - s \times \lfloor d/s \rfloor$ accounts for a potential last block of size $c < s$.

For a better normalization of d_ToD, we consider the following model:

$$\hat{d}_\mathrm{ToD}(\alpha, d, s, r_\mathrm{r}) = \alpha_0(d+\alpha_1)^{\alpha_2} \times (s+\alpha_3)^{\alpha_4} \times (r_\mathrm{r}+\alpha_5)^{\alpha_6} \times (\frac{s}{r_\mathrm{r}\, d} + \alpha_7)^{\alpha_8} \ , \tag{5.30}$$

where $\alpha = (\alpha_0, \ldots, \alpha_8)$ are the free parameters of the fit to be optimized. A simpler model with $\alpha_8 = 0$ was considered but resulted in no good fit.

We, then, use CMA-ES to find an optimal fit on α of the empirical data by minimizing

$$\alpha_\mathrm{opt} = \arg\min_{\alpha} \sum_{i=1}^{\#data} \left(\log\left(\hat{d}_\mathrm{ToD}(\alpha, d^i, s^i, r_\mathrm{r}^i)\right) - \log\left(d_\mathrm{ToD}(\mathbf{R}^i)\right)\right)^2 \ , \tag{5.31}$$

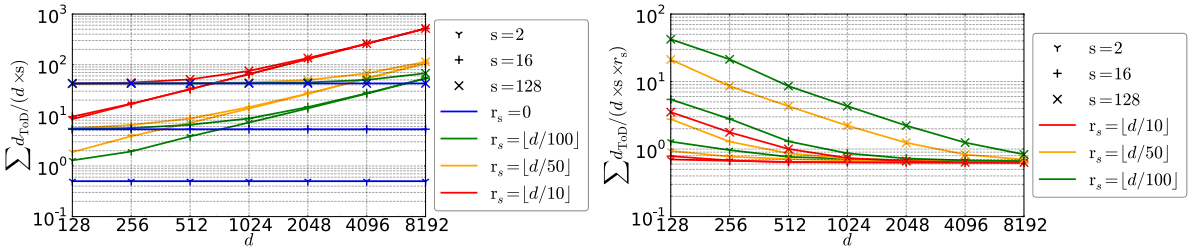

Figure 5.2: Normalized sum of distances $d_{\mathrm{ToD}}(\mathbf{B})$ (see (5.27)) of a full block-diagonal matrix after applying a truncated uniform permutation (Algorithm 3), $\mathbf{P}_{\mathrm{left}}\mathbf{B}$, with $n_s = d$ and equal block-sizes versus problem dimension d. Different block-sizes s and swap-ranges r_{s} are considered and shown in the legend. Each plotted symbol is the average of 19 repetitions. The same data is shown in the two plots with different normalizations. **Left:** normalization by $d \times s$; **Right:** normalization by $d \times s \times r_{\mathrm{s}}$.

where $\mathbf{R}^i$ is the i^{th} sampled matrix and $d^i, s^i, r_{\mathrm{r}}^i$ are its corresponding parameter values. We use the logarithm in order to avoid biasing in favor of larger values of d (which result in larger d_{ToD}). The first fit obtained by optimizing over all the variables $\alpha_0, \ldots, \alpha_8$ reads

$$0.15 \times (d - 2.25)^{2.03} \times (s + 0.07)^{0.95} \times (r_{\mathrm{r}} - 0.00)^{1.01} \times \left(\frac{s}{r_{\mathrm{r}}\, d} + 2.85\right)^{1.25} . \tag{5.32}$$

Then, we fix the values of some α's: $\alpha_1 = 0$ (since dimensions are expected to be significantly larger than 2), $\alpha_3 = 0$ and $\alpha_5 = 0$ and exclude them from the next optimization to end up with:

$$0.15 \times d^{2.04} \times s^{0.94} \times r_{\mathrm{r}}^{1.03} \times \left(\frac{s}{r_{\mathrm{r}} \times d} + 2.83\right)^{1.24} . \tag{5.33}$$

With the next round of rounding values and fixing them: $\alpha_2 = 2, \alpha_4 = 1, \alpha_6 = 1$ resulting in:

$$0.11 \times d^2 \times s \times r_{\mathrm{r}} \times \left(\frac{s}{r_{\mathrm{r}} d} + 3.91\right)^{1.27} . \tag{5.34}$$

And finally: $\alpha_0 = 0.1, \alpha_7 = 4, \alpha_8 = 1.3$:

$$d_{\mathrm{ToD}}(\mathbf{P}_{\mathrm{left}}\mathbf{B}) \approx 0.1 \times d^2 \times s \times r_{\mathrm{r}} \times \left(\frac{s}{r_{\mathrm{r}}\, d} + 4\right)^{1.3} . \tag{5.35}$$

We see a quadratic dependency on the problem dimension d and a linear one on the block-size s. In addition, the last term tends to be constant when $s = O(d)$ which means the scaling of d_{ToD} is $d \times s^2$. This is the same scaling as that of the non-permuted matrix ($\mathbf{P}_{\mathrm{left}} = \mathbf{I}_d$ and/or $r_{\mathrm{s}} = 0$) (5.29), and thus also becomes a scaling in d^3 for a single-block matrix ($s = d$) (5.28). The effect of the permutation can be estimated by the difference between the value in (5.35) and the value using the expression for the non permuted block-matrix (5.29).

5.3.3 Measure of Difficulty

In order to better quantify the effect of the transformation matrix defined in (5.18), we estimate the difficulty associated with a problem by the Expected Running Time (ERT) of sep-CMA-ES on this problem, that is the expected number of function evaluations needed (by the restarted algorithm) to reach a target value for the first time [Hansen et al., 2010a]. The reason for this choice is that because of its restricted, diagonal, covariance matrix model, sep-CMA-ES only manages to solve problems with either no dependencies between the parameters or limited ones. It performs well when solving separable and block-separable functions as long as the condition numbers of the blocks are relatively small. Also, it is invariant to the coordinate permutation $\mathbf{P}_{\text{right}}$ which allows us to restrict our study to $\mathbf{P}_{\text{left}}$ and $\mathbf{B}$. Thus, by comparing a performance to that on the original, non transformed and separable problem, we get an idea of how *non-separable* the problem becomes; helping us decide on the parameter choice. Ultimately, we aim for parameter values that make the problem unsolvable by sep-CMA-ES in reasonable time, since otherwise, the properties of the problems can be exploited.

5.4 Impact of the Block Condition Number

In this section, we are interested in the effect of the block-diagonal matrix $\mathbf{B}$ on the difficulty of a problem as defined in Section 5.3.3. This difficulty is tightly related to the condition numbers in the Hessian matrix of the transformed problem, within the blocks defined by $\mathbf{B}$. That is, the maximal ratio between the eigenvalues in the blocks that are delimited by $\mathbf{B}$.

To do so, we define a specific, ellipsoid-like, convex-quadratic function that suits exactly the block structure of our matrix $\mathbf{B}$:

$$f_{\mathbf{B},\mathbf{D}}(\mathbf{x}) = \mathbf{x}^T\mathbf{B}^T\mathbf{D}\mathbf{B}\mathbf{x} \ , \tag{5.36}$$

where $\mathbf{D} \in \mathbb{R}^{d\times d}$ is a diagonal matrix and $\mathbf{B}$ an orthogonal block-diagonal matrix with blocks of the same size s. In order to have control over both the overall condition number that we note c_o and the block condition number c_b of the Hessian matrix of (5.36), we define the diagonal entries of $\mathbf{D}$:

$$\mathbf{D}_{i,i} = c_b^{\left(\frac{i\%s}{s-1}\right)} \times c_{ib}^{\lfloor i/s \rfloor} \ , \tag{5.37}$$

where % designates the modulo operator, that is the rest of euclidean division, and c_{ib} a constant that ensures the overall condition number is kept under control by setting

$$c_o = c_b \times c_{ib}^{(n_b-1)} \ , \tag{5.38}$$

where n_b is the number of blocks in $\mathbf{B}$. In $\mathbf{D}$, the eigenvalues are distributed uniformly in the logarithmic scale in what corresponds to the blocks of $\mathbf{B}$. Since $\mathbf{B}$ is orthogonal, $\mathbf{D}$ preserves its eigenvalues and thus condition numbers (block and overall).

In the right plot of Figure 5.1, we show the ERT of sep-CMA-ES while varying the block size s and the block condition number c_b of the function defined in (5.36). The overall condition number c_o remains, however, constant to allow comparison between problems of seemingly similar difficulty (we expect the standard CMA-ES to perform similarly on these problems). The graph for $s = 1$ serves as a baseline comparison to the case where no transformation matrix is applied (entries of the then diagonal $\mathbf{B}$ are -1 and 1). Note that in such a case ($s = 1$), the value of the parameter c_b is irrelevant since each blocks contains a single element and thus has an internal condition number of 1 (the element in question is different from 0).

We see a linear scaling between the ERT and the block condition number (compare with the dashed gray line). In addition, when keeping the same overall condition number c_o, we see a reverse in the effect of the block-size on the difficulty of the problems depending on the block condition-number c_b. For smaller values ($c_b < 10^2$), smaller block-sizes result in easier problems (although the difference in ERT is relatively small) while larger values ($c_b \geq 10^2$) show a negative correlation of the ERT with the block-size. For $c_b > 10^4$, the algorithm does not manage to solve the problem regardless of the block size ($s = 1$ excluded). We also see little difference in the difficulty of the problem on different low values of c_b ($\leq 10^2$), except when $s \in \{2, 4\}$.

The block condition number has a direct effect on the difficulty of the problem as we define it. In fact, even without the permutations, a high enough block condition-number results in a difficult problem for sep-CMA-ES. This, despite the fact that sep-CMA-ES manages to solve the non-transformed version of the problem with the same overall condition number c_o (horizontal dark blue for $s = 1$) and the fact that the transformation matrix is orthogonal.

5.5 Parameter Choice for the Benchmarks

In this section, we set the remaining two parameters of our transformation $\mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}$, namely the block-size s and swap range r_{s} or its multiplier r_{r}. Out of the properties we needed our transformation to satisfy that were defined in Section 5.2, only the orthogonality property (Property 3) is independent of the parameter setting of $\mathbf{R} = \mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}$. Thus, both the cost of applying $\mathbf{R}$ (Property 1) and the level of non-separability introduced (Property 2) should be taken into consideration when deciding on the parameter setting to adapt. We have seen in Section 5.4 that only introducing non-separability might not be enough for the problem to be non-trivial and hard to solve for algorithms such as sep-CMA-ES.

5.5.1 Initial Guess

The cost of applying $\mathbf{R}$ depends solely on $\mathbf{B}$ and $(s_i)_{1 \leq i \leq n_b}$ since $\mathbf{P}_{\text{left}}$ and $\mathbf{P}_{\text{right}}$ have a predetermined cost of d (used as indexes) and thus would not increase any linear complexity. If we assume that all blocks have the same size s (when block-sizes are different, we can simply apply the same reasoning about complexity on the largest block), $\mathbf{B}$ would contain,

in theory, $s \times d$ non-zero elements than need to be stored. The application of $\mathbf{B}$ also scales in $s \times d$ as we will see in detail in Section 5.6.3. Thus, for cost of s (memory and number of operations) linear in d, we need s to be in $O(1)$ of d. This implies a value that does not scale with d, complexity-wise; the smaller the better. However, we still need the blocks to be large enough to impact the difficulty of the problem, which results in contrasting objectives. The largest considered orthogonal matrix in BBOB-2009 is of size 40, so a parameter choice $s = \min(d, 40)$ seems reasonable, limiting the scaling of the cost beyond dimension 40.

Then we set the swap range r_s. We are interested in the difficulty of the resulting problem in order to satisfy Property 2. Clearly, the swap range should not be too small in comparison to the block-size, since otherwise, the block structure of $\mathbf{B}$ would not change significantly enough when applying $\mathbf{P}_\mathrm{left}$ and $\mathbf{P}_\mathrm{right}$. Complexity-wise however, the permutations have a constant cost of d regardless of r_s; so r_s can be set dependent of d. Let us consider an ellipsoid function with a fixed overall condition number. Since we consider bounded block-sizes, when the problem dimension tends to infinity, the block condition-numbers (similarly to the ones defined in Section 5.4) tends to 1. Having a constant swap range will maintain this small block condition-number, since the eigenvalues associated with the swapped variables would tend, as $d \to \infty$, to be similar and the distance between swapped variables is determined by r_s which defines the largest distance in a single swap. On a function like the ellipsoid where the eigenvalues are ordered and uniformly distributed in the logarithmic scale, in order to obtain a constant expected ratio between the eigenvalues of the swapped variables that does not tend to 1 as $d \to \infty$, we need $r_\mathrm{r} = r_\mathrm{s}/d$ to be constant. Thus the expected relative distance between the swapped variables remains invariant with the dimension. A relative swap range r_r of a third of the dimension seems reasonable. Smaller values would result in the problem keeping most of its block-structure; an exploitable feature. Larger values have the reverse effect, risking to lose too much of the original block-structure and fall in a case similar to that of the non-truncated uniform swaps (all other variables are equally likely potential second swap variables) distribution where we lack control over the resulting matrix and difficulty of the resulting problem.

5.5.2 Empirical Validation on sep-CMA-ES

In order to confirm the parameter choice suggested above, we investigate the evolution of the difficulty of the problem, as we define it in Section 5.3.3, with different values of these parameters. In fact, we expect the transformed problems to remain non-solvable by separable algorithms such as sep-CMA-ES despite the restricted model of the transformation matrix (no longer a full rotation matrix as is the case in BBOB-2009).

First, we look into the performance of sep-CMA-ES on a transformed Ellipsoid function since we based our reasoning about the choice of r_s on it. In this case, we consider the following problem

$$g(\mathbf{x}) = f_\mathrm{raw}^\mathrm{Ellipsoid}(\mathbf{z}), \tag{5.39}$$

where $\mathbf{z} = \mathbf{P}_\mathrm{left}\mathbf{B}\mathbf{P}_\mathrm{right}\mathbf{x}$, $\gamma(d) = \min(1, 40/d)$ and $f_\mathrm{raw}^\mathrm{Ellipsoid}$ as defined in Table 5.1. The

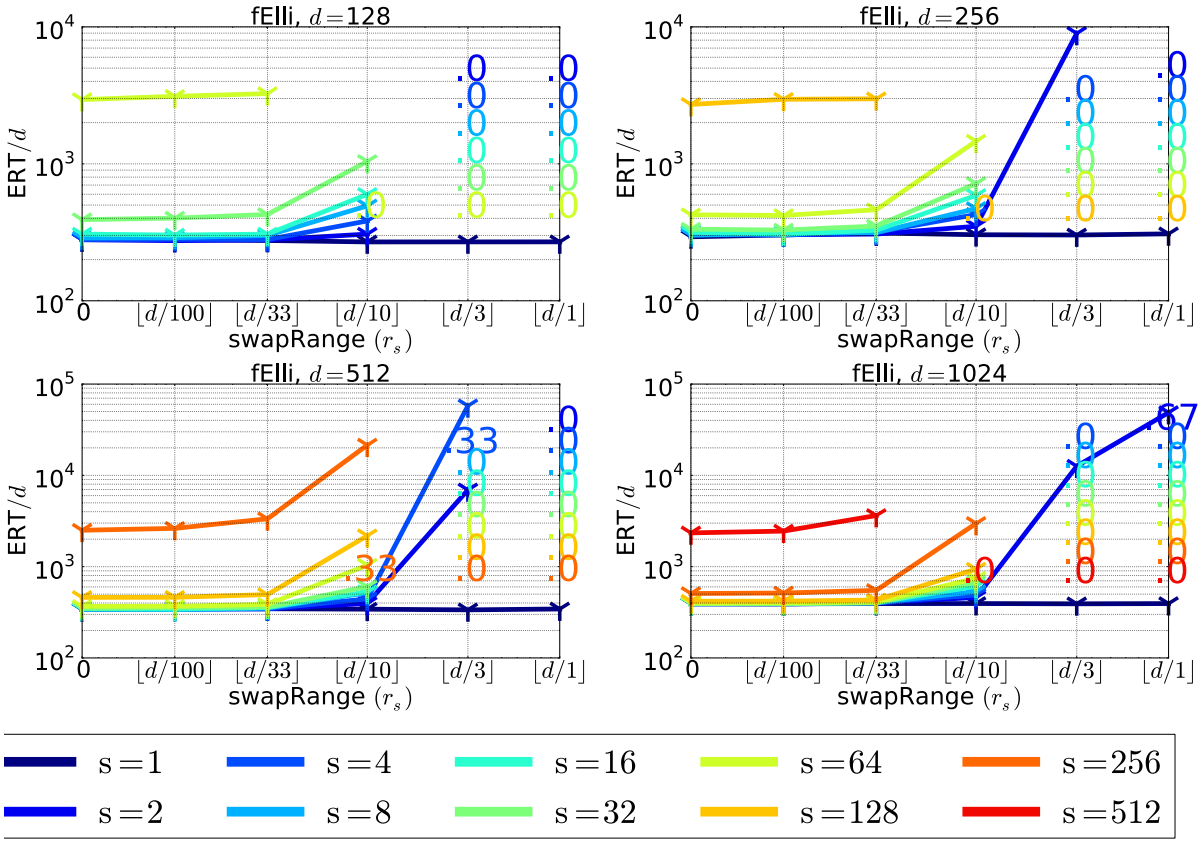

Figure 5.3: ERT of sep-CMA-ES divided by problem dimension d on the transformed ellipsoid function (5.39) versus the swap range r_s for different values of the problem dimension d and the block-size s. A budget of $10^5 \times d$ is used with 3 runs on each configuration. and a target value set to 10^{-8}. The setting $r_\mathrm{s} = 0$ means no swap is applied ($\mathbf{P}_\mathrm{left} = \mathbf{P}_\mathrm{right} = \mathbf{I}_d$). The numbers that are shown on the plots, coded by color for the corresponding block-size, are for success rates when smaller than 1.

normalizing term $\gamma(d)$ is used to obtain, given a target precision, a similar difficulty level across the dimensions. It also allows, in principle, to not increase the distance in the objective space between the initial solution and the target solution by increasing the dimension. We use the minimum in order to be backward compatible with BBOB-2009 where the largest dimension is 40.

In Figure 5.3, the ERT of sep-CMA-ES on (5.39) is plotted against different values of r_s, s and d.

We see that both s and r_s increase the ERT as their values increase. sep-CMA-ES manages to solve all non-permuted problems ($r_\mathrm{s} = 0$), even on the larger block-sizes. The overall condition number is 10^8 which means the block condition-numbers are relatively low for all block-sizes, $c_b \approx 10^{6/n_b}$; and do not exceed 10^3. For small swap-ranges, the

ERTs do not seem to change, however, we notice a change in phase in the ERT between $r_s = \lfloor d/33 \rfloor$ and $r_s = \lfloor d/10 \rfloor$ where the ERTs start increasing. For $r_s = \lfloor d/3 \rfloor$, sep-CMA-ES solves only problems with up to $s = 4$ across the tested dimensions.

The permutation $\mathbf{P}_{\text{left}}$ changes the ratios between the eigenvalues associated to each set of dependent variables, where the dependencies are defined by the block-diagonal matrix $\mathbf{B}$. That is variables are dependent if they belong to the same block,/their indexes coincide with indexes in $\mathbf{B}$ that are contained in the same block. This change in ratios affects the *condition number* within each block of dependent variables, an effect similar to that witnessed when *manually* varying the block condition numbers of non-permuted block-diagonal matrices in Section 5.4. As seen in Section 5.4, this increases the difficulty of the problem for sep-CMA-ES.

In Figure 5.4, we do the same experiments as in Figure 5.3 on transformed versions of the Cigar, Tablet (also known as the Discus function), Sum of Different Powers and Rosenbrock functions (see the raw functions in Table 5.1 with $\mathbf{z} = \mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}\mathbf{x}$).

On the Sum of Different Powers function, the results are quite similar to what was observed on the transformed Ellipsoid function while the generalized Cigar and Tablet functions seem to be difficult enough with the block-matrix transformation alone. The only successes we see on these last two functions (for $s > 1$) are for $s = 2$ with the smaller swap ranges.

On the other hand, the Rosenbrock function is not convex quadratic and is multi-modal. It is also non-separable in its raw form, partially separable with a tri-band structure where each variable interact directly only with the two variables that are adjacent to it. We see failed runs of sep-CMA-ES even on the raw function ($s = 1, r_s = 0$). However, we still observe an effect of s and r_s on the ERT to a lesser extent because of the noise provided by the failed runs. We also notice the changes in phase between the same two values of the swap range $r_s = \lfloor d/33 \rfloor$ and $r_s = \lfloor d/10 \rfloor$, but only for relatively higher block-sizes. For $r_s = \lfloor d/3 \rfloor$, successes are observed only for block-sizes as high as $s = 8$.

On the generalized Tablet and Cigar functions, and for $d = 128$, 4 eigenvalues are different from the rest (larger and smaller respectively). In the raw function, these eigenvalues are adjacent in the first 4 positions (see Table 5.1). Thus, the largest block condition number is equal to the overall condition number, except for the case $s \in \{1, 2, 4\}$ and $n_s = 0$ where it is equal to 1. This is why the only all-successful runs are observed on these configurations. Further confirmation of this observation on Cigar and Tablet functions with smaller overall condition numbers can be found in Figure 5.6 and Figure 5.5 respectively. This is also a good illustration of the effect and importance of $\mathbf{P}_{\text{left}}$ on the structure and difficulty of the problems since we know that sep-CMA-ES is invariant to the effect of $\mathbf{P}_{\text{right}}$.

All unsuccessful runs in Figures 5.3,5.4, 5.6 and 5.5 terminate with the stop flag *tolupsigma*. This indicates (taken from the documentation of the CMA-ES algorithm (https://pypi.python.org/pypi/cma): "creeping behavior with usually minor improvements". We interpret this as the model of sep-CMA-ES being unable to fit these problems. Thus, we consider that sep-CMA-ES fails to solve the problem in this scenario, which is the

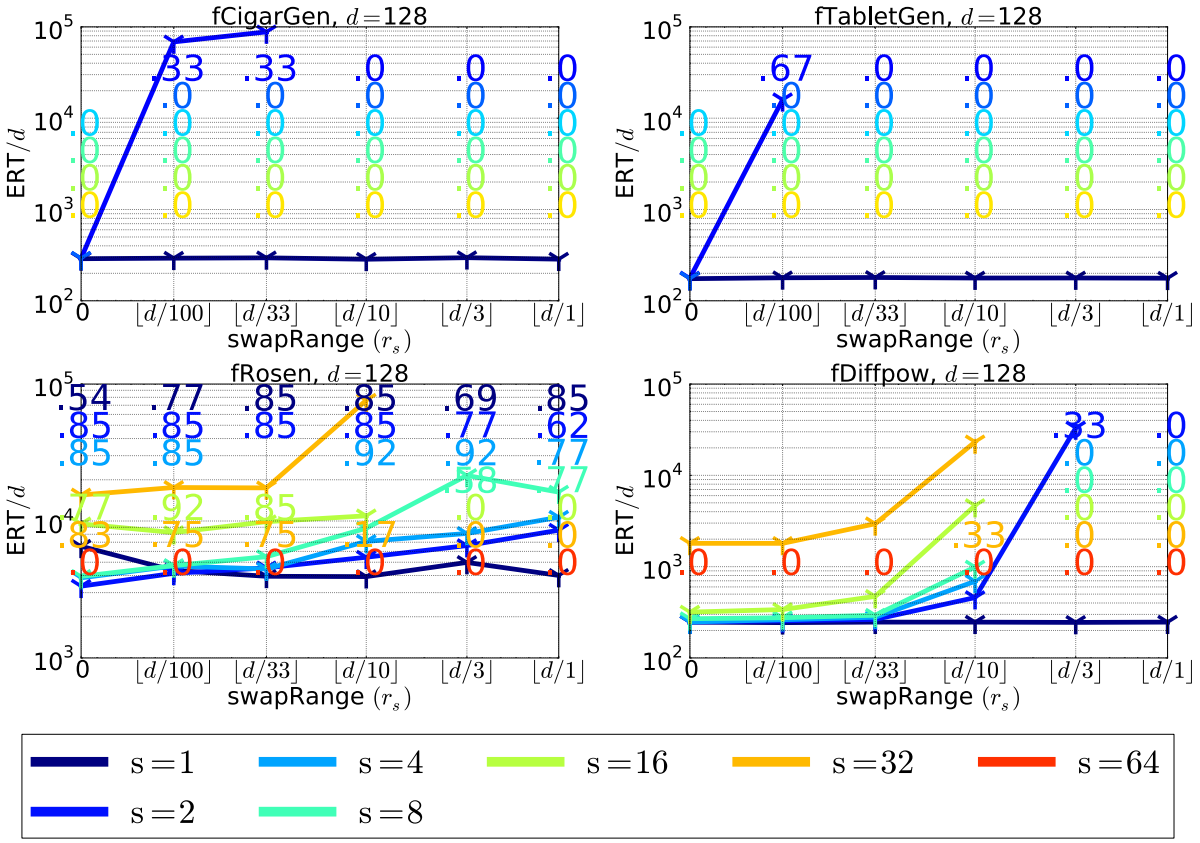

Figure 5.4: ERT of sep-CMA-ES divided by d on transformed functions in dimension 128. From **left** to **right**, **top** to **bottom**: Cigar, Tablet (or Discus), Rosenbrock and Sum of Different Powers functions. The problems are defined as $f(\mathbf{x}) = f_{\text{raw}}(\mathbf{z})$, with $\mathbf{z} = \mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}\mathbf{x}$ and taking the corresponding f_{raw} from Table 5.1. The experimental setup is the same as in Figure 5.3 except on the transformed Rosenbrock functions where the ERT is computed from 19 runs (because of higher variance).

desired result in our case.

Following these experiments, we see that a parameter setting $r_{\text{s}} = \lfloor d/3 \rfloor$ and $s = 16$ is sufficient to make the problems hard enough that sep-CMA-ES can not exploit the block-diagonal structure. This parameter setting is also in accordance with the considerations taken in the beginning of this section so we can safely choose

$$r_{\text{s}} = \lfloor d/3 \rfloor, s = \min(d, 40) \ . \tag{5.40}$$

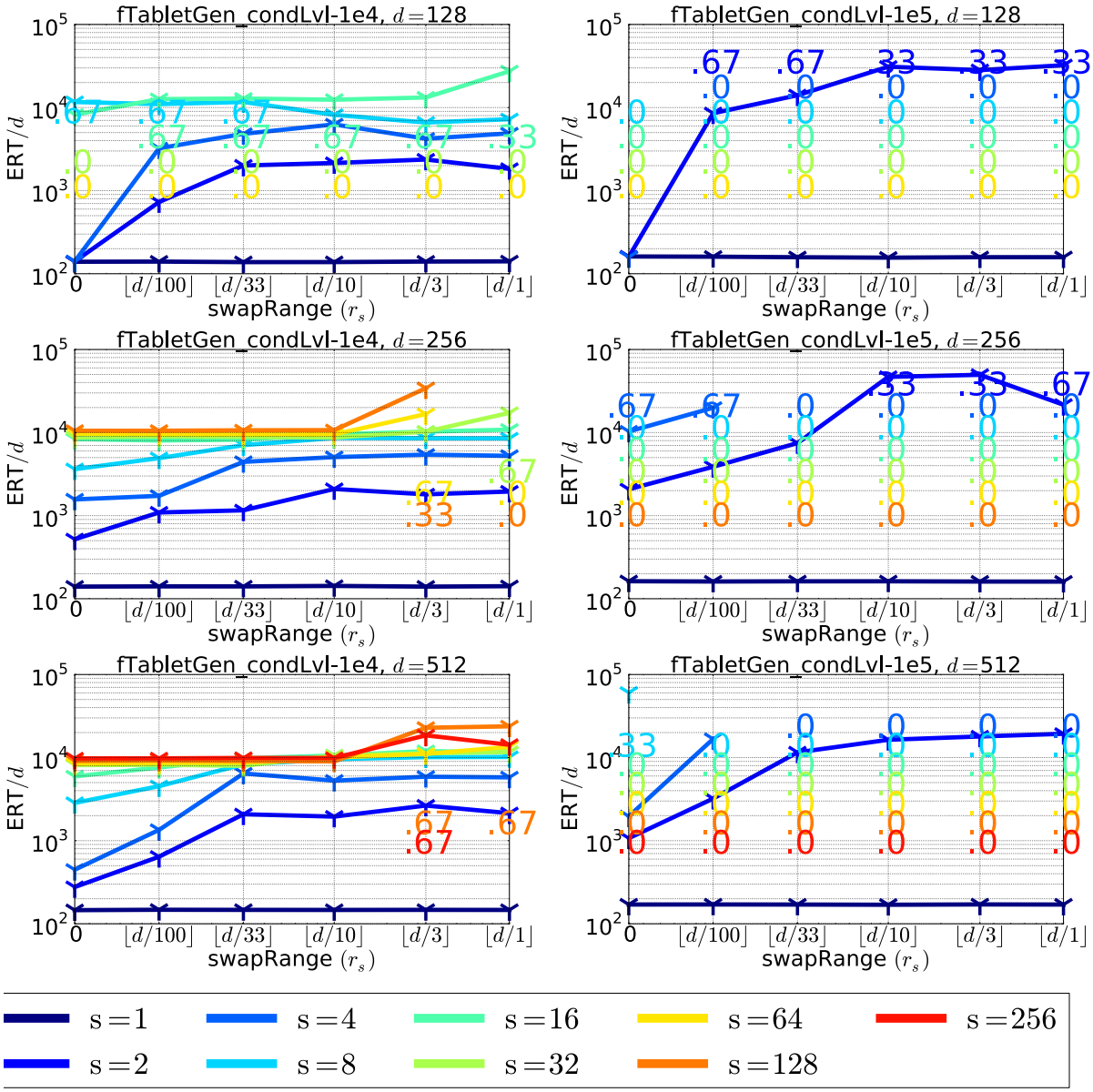

Figure 5.5: Similar to Figure 5.2 but only considering the Tablet (Discus) functions with different overall condition numbers and different dimensions.

5.6 The Large Scale Benchmark

5.6.1 Changes to the Raw Functions

Before applying the new transformation matrix (5.18) in place of full orthogonal matrices, we make two modifications to the raw functions in BBOB-2009. Table 5.1 describes all

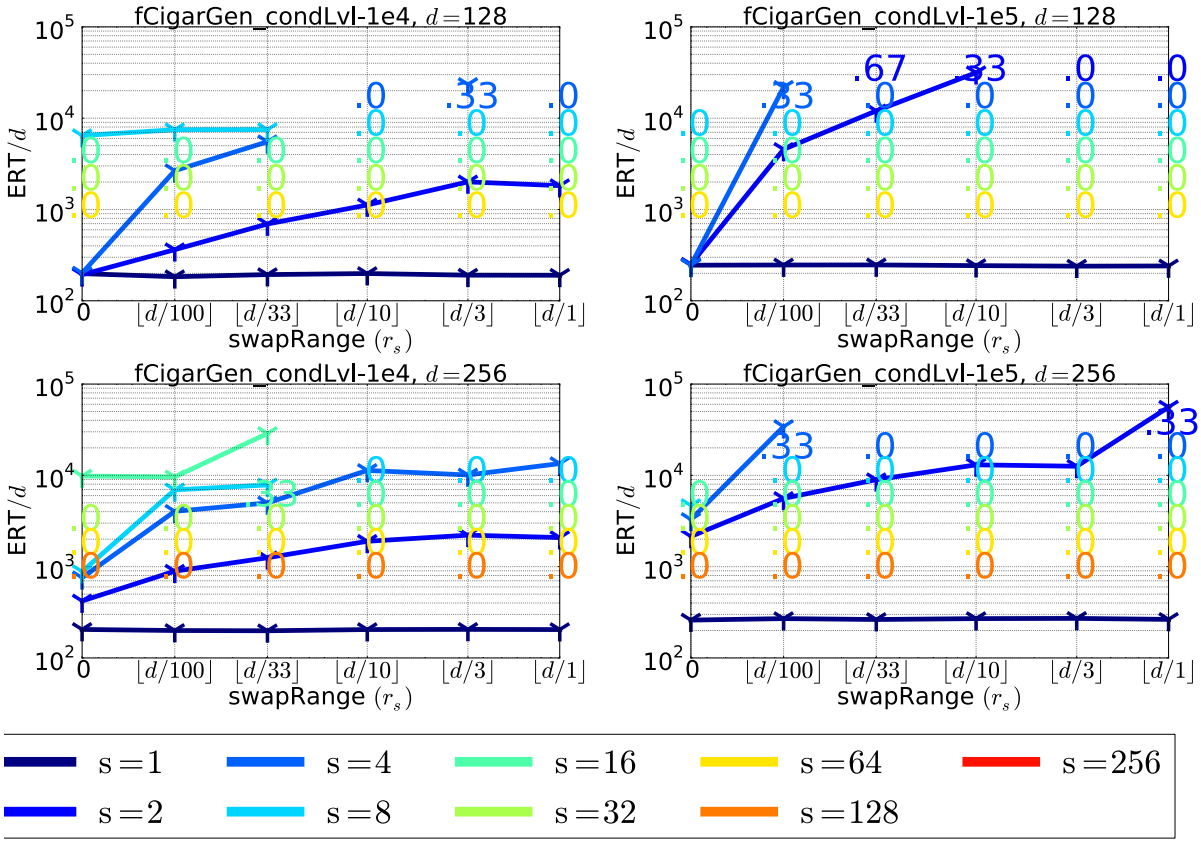

Figure 5.6: Same as Figure 5.5 on the transformed Cigar function.

the raw functions used in the large-scale testbed. This testbed is derived from the BBOB-2009 functions [Hansen et al., 2009] with two major changes. Note that these changes are set such that the problem definitions remain backward compatible with the problems in BBOB-2009. What we mean by backward compatible here is that the problems expressed in this chapter are the same as the ones defined in BBOB-2009 when $d \leq 40$.

The first change, and as previously described, is that we normalize the raw functions such that the difficultly associated to a same target values is similar across the dimensions. The backward compatibility is insured by making sure that $\gamma(d) = 1$ when $d \leq 40$. In our case, we choose:

$$\gamma(d) = \min(1, 40/d) \ . \tag{5.41}$$

This normalization was only applied to functions whose original definitions do not include a similar normalization by the dimension. Thus, it (the normalization) is omitted from the raw definitions of $f_{\text{raw}}^{\text{Weierstrass}}$, $f_{\text{raw}}^{\text{SchaffersF7}}$, $f_{\text{raw}}^{\text{Griewank-Rosenbrock}}$, $f_{\text{raw}}^{\text{Schwefel}}$ and $f_{\text{raw}}^{\text{Gallagher}}$ in Table 5.2.

The second change is that the Cigar, Discus (Tablet) and the Sharp Ridge functions (see new raw definition in Table 5.1) are generalized to have a (to some extent given the

rounding) constant ratio between short axes and long axes. The backward compatibility with BBOB-2009 is preserved by setting this ratio to 1/40 (non-integer values are rounded to the closest larger integer).

5.6.2 The Test-Suite Problem Definitions

For each function, we consider the same transformations that are used in [Hansen et al., 2009] except the full orthogonal matrices that we replace by $\mathbf{R} = \mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}$. Section 5.1.1 describes these transformations that are, except for the orthogonal transformations, the same than those found in [Hansen et al., 2009]. The full list of the 24 large scale problems of the benchmark is shown in Tables 5.3 and 5.4 where we can see the list of transformations applied for each problem. The definitions of the raw functions used in these problems are presented in Tables 5.1 and 5.2.

We keep the functions in their respective categories (as defined in [Hansen et al., 2009]) since the introduced transformation satisfies Property 2 and Property 3.

5.6.3 Implementation Details and Cost of Applying the Transformation

We now present some of the implementation details (mostly technical) that allow to store and compute the transformation matrix $\mathbf{R} = \mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}$ in an efficient way.

In order to apply the transformation on a solution $\mathbf{x}$ and obtain its transformed counterpart $\mathbf{z}$, we naturally need to compute for each $i \in \{1, \ldots, d\}$:

$$z_i = [\mathbf{R}\mathbf{x}]_i = [\mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}\mathbf{x}]_i \ , \tag{5.42}$$

where $[\mathbf{x}]_i$ designate the i^{th} coordinate of $\mathbf{x}$, $\mathbf{P}_{\text{left}}$ and $\mathbf{P}_{\text{right}}$ are permutation matrices and $\mathbf{B}$ is a block-diagonal matrix (orthogonality is not important here) with block-sizes $(s_i)_{1 \leq i \leq n_b}$.

We store $\mathbf{B}$ (5.16) in a list of vectors $\bar{\mathbf{B}} = (\bar{\mathbf{b}}_1, \bar{\mathbf{b}}_2, \ldots, \bar{\mathbf{b}}_d)$ that are the rows of the matrices $\mathbf{B}_1, \ldots, \mathbf{B}_{n_b}$ that compose $\mathbf{B}$ (5.16). We denote $\bar{b}_{i,j}$ the coordinate j of $\bar{\mathbf{b}}_i$ and $\mathbf{b}_k$ the k^{th} row of $\mathbf{B}$.

Since $\mathbf{P}_{\text{left}}$ and $\mathbf{P}_{\text{right}}$ are permutation matrices, they can, completely, be described using permutation vectors such as the ones generated by Algorithm 3. Thus, we store $\mathbf{P}_{\text{left}}$ and $\mathbf{P}_{\text{right}}$ as two index vectors $\mathbf{p}^{\text{left}}$ and $\mathbf{p}^{\text{right}}$ respectively. They are index vector in the sens that they are used to index the coordinates to be permuted: $z_i^{\text{new}} = z_{p_i}$.

We also keep two lists of d elements, $\mathbf{s}^{\text{row}}$ and $\mathbf{j}^{\text{first}}$. These lists are defined for each $k \in \{1, 2, \ldots, d\}$ as follows [Ait Elhara et al., 2016]:

- s_k^{row} is the size of the vector $\bar{\mathbf{b}}_k$, which allows to only access as many entries per vector as there are, in theory, non-zero ones (no extra accessing or checking for entries that are, given the block-diagonal nature of $\mathbf{B}$, set to 0)
- j_k^{first} is the number of leading zeros, excluding the eventual zeros in $\bar{\mathbf{b}}_k$, of $\mathbf{b}_k$. It allows to know the original position of each element $\bar{b}_{k,j}$ in $\mathbf{b}_k$ and is needed in order to multiply $\bar{b}_{k,j}$ by the corresponding coordinate of $\mathbf{x}$ as we will see in (5.45).

$f_{\text{raw}}^{\text{Sphere}}(\mathbf{z}) =$	$\gamma(d) \times \sum_{i=1}^{d} z_i^2$
$f_{\text{raw}}^{\text{Ellipsoid}}(\mathbf{z}) =$	$\gamma(d) \times \sum_{i=1}^{d} 10^{6\frac{i-1}{d-1}} z_i^2$
$f_{\text{raw}}^{\text{Rastrigin}}(\mathbf{z}) =$	$\gamma(d) \times \left(10d - 10\sum_{i=1}^{d} \cos(2\pi z_i) + \|\mathbf{z}\|^2\right)$
$f_{\text{raw}}^{\text{LinearSlope}}(\mathbf{z}) =$	$\gamma(d) \times \left(\sum_{i=1}^{d} \left(5\lvert s_i^{f5}\rvert - s_i^{f5} z_i\right)\right)$, with $s_i^{f5} = \text{sign}(x_i^{\text{opt}}) 10^{\frac{i-1}{d-1}}$
$f_{\text{raw}}^{\text{AttractiveSector}}(\mathbf{z}) =$	$\gamma(d) \times T^{\text{osz}}\left(\sum_{i=1}^{d} \left(s_i^{f6} z_i\right)^2\right)^{0.9}$, with $s_i^{f6} = \begin{cases} 10^2 & \text{if } z_i \times x_i^{\text{opt}} > 0 \\ 1 & \text{otherwise} \end{cases}$
$f_{\text{raw}}^{\text{StepEllispoid}}(\mathbf{z}) =$	$\gamma(d) \times 0.1 \max\left(\lvert\hat{z}_1\rvert/10^4, \sum_{i=1}^{d} 10^{2\frac{i-1}{d-1}} z_i^2\right)$ $\hat{\mathbf{z}}$ defined in Table 5.3
$f_{\text{raw}}^{\text{Rosenbrock}}(\mathbf{z}) =$	$\gamma(d) \times \sum_{i=1}^{d} \left(100\left(z_i^2 - z_{i+1}\right)^2 + \left(1 - z_i\right)^2\right)$
$f_{\text{raw}}^{\text{Discus}}(\mathbf{z}) =$	$\gamma(d) \times \left(10^6 \sum_{i=1}^{\lceil d/40\rceil} z_i^2 + \sum_{i=\lceil d/40\rceil+1}^{d} z_i^2\right)$
$f_{\text{raw}}^{\text{BentCigar}}(\mathbf{z}) =$	$\gamma(d) \times \left(\sum_{i=1}^{\lceil d/40\rceil} z_i^2 + 10^6 \sum_{i=\lceil d/40\rceil+1}^{d} z_i^2\right)$
$f_{\text{raw}}^{\text{SharpRidge}}(\mathbf{z}) =$	$\gamma(d) \times \left(\sum_{i=1}^{\lceil d/40\rceil} z_i^2 + 100\sqrt{\sum_{i=\lceil d/40\rceil+1}^{d} z_i^2}\right)$
$f_{\text{raw}}^{\text{DifferentPowers}}(\mathbf{z}) =$	$\gamma(d) \times \sum_{i=1}^{d} \lvert z_i\rvert^{\left(2+4\times\frac{i-1}{d-1}\right)}$

Table 5.1: Definitions of the raw functions used in the large-scale test-bed. These raw functions are obtained from BBOB-2009 [Hansen et al., 2009] by assuming all transformations are identity transformations (orthogonal matrices are $\mathbf{I}_d$, $f_{\text{opt}} = 0$,...). The problem dimension is d and we suppose all problems have, as input, a solution $\mathbf{x}_{\text{opt}}$ that will become the optimal solution of the generated problem (needed in order to properly define s_i^{f5} and s_i^{f6}). In addition, a normalizing-by-dimension factor , $\gamma(d)$, is applied and the Cigar, Discuss, and Sharp Ridge functions are generalized. All the functions of this table are multiplied by $\gamma(d) = \min(1, 40/d)$ such that a constant target value (e.g., 10^{-8}) represents the *same level of difficulty* across all dimensions $d \geq 40$. That is no additional difficulty because of the larger number of variables and the fact that the functions defined here are sums of d values. We use the variable $\mathbf{z}$ instead of $\mathbf{x}$ since these functions are meant to be applied to the transformed variables.

Thus, in total we need to store $\bar{\mathbf{B}}, \mathbf{p}^{\text{right}}, \mathbf{p}^{\text{left}}, \mathbf{j}^{\text{first}}$ and $\mathbf{s}^{\text{row}}$ which result in a memory cost in the order of

$$4d + \sum_{i=1}^{n_b} s_i^2 = 4d + n_b \text{avg}(s_i^2) \ , \tag{5.43}$$

where $\text{avg}(s_i^2)$ the average value of s_i^2.

If we consider equal block-sizes $s_i = s$, we obtain:

$$n_b \text{avg}(s_i^2) \approx n_b \text{avg}(s_i)^2 = d \times \text{avg}(s_i) \approx s \times d \tag{5.44}$$

$$f_{\text{raw}}^{\text{Weierstrass}}(\mathbf{z}) = 10\left(\frac{1}{d}\sum_{i=1}^{d}\sum_{k=0}^{11}\frac{1}{2^k}\cos(2\pi 3^k(z_i+1/2)) - f_0\right)^3 ,$$
with $f_0 = \sum_{k=0}^{11}\frac{1}{2^k}\cos(\pi 3^k)$

$$f_{\text{raw}}^{\text{SchaffersF7}}(\mathbf{z}) = \left(\frac{1}{d-1}\sum_{i=1}^{d-1}\left(\sqrt{s_i^{f17}} + \sqrt{s_i^{f17}}\sin^2\left(50(s_i^{f17})^{1/5}\right)\right)\right)^2 ,$$
with $s_i^{f17} = \sqrt{z_i^2 + z_{i+1}^2}$

$$f_{\text{raw}}^{\text{GR}}(\mathbf{z}) = \frac{10}{d-1}\sum_{i=1}^{d-1}\left(\frac{s_i^{f19}}{4000} - \cos\left(s_i^{f19}\right)\right) + 10 ,$$
with $s_i^{f19} = 100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2$

$$f_{\text{raw}}^{\text{Schwefel}}(\mathbf{z}) = -\frac{1}{d}\sum_{i=1}^{d} z_i \sin\left(\sqrt{|z_i|}\right) + s_{f20} ,$$
with $s_{f20} = 4.189828872724339$

$$f_{\text{raw}}^{\text{Gallagher}}(\mathbf{z}) = \left(10 - \max_{i=1}^{n_{\text{peaks}}} w_i \exp\left(-\frac{1}{2d}(\mathbf{z}-\mathbf{y}_i)^{\mathsf{T}}(\mathbf{z}-\mathbf{y}_i)\right)\right)^2 ,$$
with $n_{\text{peaks}} > 2$,
$$w_i = \begin{cases} 1.1 + 8 \times \dfrac{i-2}{n_{\text{peaks}}-2} & \text{if } 2 \le i \le k \\ 10 & \text{otherwise} \end{cases} ,$$
$\mathbf{y}_i$ are the local optima (to be set) and $\mathbf{x}_{\text{opt}} = \mathbf{y}_1$

$$f_{\text{raw}}^{\text{Katsuura}}(\mathbf{z}) = \gamma(d) \times \frac{10}{d^2}\left(\prod_{i=1}^{d}\left(1 + i\sum_{j=1}^{32}\frac{|2^j z_i - [2^j z_i]|}{2^j}\right)^{10/d^{1.2}} - 1\right)$$

$$f_{\text{raw}}^{\text{LR}}(\mathbf{z}) = \gamma(d) \times \left(\min\left(\sum_{i=1}^{d}(\hat{x}_i - \mu_0)^2, s\sum_{i=1}^{d}(\hat{x}_i - \mu_1)^2\right) + 10\left(d - \sum_{i=1}^{d}\cos(2\pi z_i)\right)\right)$$
with $\hat{\mathbf{x}} = 2\,\text{sign}(\mathbf{x}_{\text{opt}}) \otimes \mathbf{x}$, $\mathbf{x}_{\text{opt}} = \mu_0 \mathbf{1}_d(^+_-)$,
$\otimes$ the element-wise multiplication,
$\mathbf{1}_d(^+_-) \in \{-1, 1\}^d$, entries chosen uniformly at random,
and $s = 1 - \frac{1}{2\sqrt{d+20} - 8.2}, \mu_0 = 2.5, \mu_1 = -\sqrt{\frac{\mu_0^2 - 1}{s}}$

Table 5.2: Follow up to Table 5.1. The abbreviations GR for composite Griewank-Rosenbrock and LR for Lunacek bi-Rastrigin are used. The functions that are inherently normalized by d are not additionally multiplied by $\gamma(d)$.

$f_1(\mathbf{x}) = f^{\text{Sphere}}_{\text{raw}}(\mathbf{z}) + f_{\text{opt}}$	$\mathbf{z} = \mathbf{x} - \mathbf{x}_{\text{opt}}$
$f_2(\mathbf{x}) = f^{\text{Ellipsoid}}_{\text{raw}}(\mathbf{z}) + f_{\text{opt}}$	$\mathbf{z} = T^{\text{osz}}\left(\mathbf{x} - \mathbf{x}_{\text{opt}}\right)$
$f_3(\mathbf{x}) = f^{\text{Rastrigin}}_{\text{raw}}(\mathbf{z}) + f_{\text{opt}}$	$\mathbf{z} = \Lambda^{10} T^{\text{asy}}_{0.2}\left(T^{\text{osz}}\left(\mathbf{x} - \mathbf{x}_{\text{opt}}\right)\right)$
$f_4(\mathbf{x}) = f^{\text{Rastrigin}}_{\text{raw}}(\mathbf{z})$ $+100 f_{\text{pen}}(\mathbf{x}) + f_{\text{opt}}$	$\mathbf{z} = s_i^{f4} T^{\text{osz}}\left(\mathbf{x} - \mathbf{x}_{\text{opt}}\right)$, $s_i^{f4} = \begin{cases} 10 \times 10^{\frac{1}{2}\frac{i-1}{d-1}} & \text{if } z_i > 0 \text{ and } i \text{ odd} \\ 10^{\frac{1}{2}\frac{i-1}{d-1}} & \text{otherwise} \end{cases}$
$f_5(\mathbf{x}) = f^{\text{LinearSlope}}_{\text{raw}}(\mathbf{z}) + f_{\text{opt}}$	$z_i = \begin{cases} x_i & \text{if } x_i^{\text{opt}} x_i < 25 \\ x_i^{\text{opt}} & \text{otherwise} \end{cases}$, $\mathbf{x}_{\text{opt}} = \mathbf{z}_{\text{opt}} = 5 \times \mathbf{1}_d(^+_-)$
$f_6(\mathbf{x}) = f^{\text{AttractiveSector}}_{\text{raw}}(\mathbf{z})$ $+ f_{\text{pen}}(\mathbf{x}) + f_{\text{opt}}$	$\mathbf{z} = \mathbf{Q}\Lambda^{10}\mathbf{R}(\mathbf{x} - \mathbf{x}_{\text{opt}})$
$f_7(\mathbf{x}) = f^{\text{StepEllispoid}}_{\text{raw}}(\mathbf{z}) + f_{\text{opt}}$	$\hat{\mathbf{z}} = \mathbf{\Lambda}^{10}\mathbf{R}(\mathbf{x} - \mathbf{x}_{\text{opt}})$, $\tilde{z}_i = \begin{cases} \lfloor 0.5 + \hat{z}_i \rfloor & \text{if } \hat{z}_i > 0.5 \\ \lfloor 0.5 + 10\hat{z}_i \rfloor / 10 & \text{otherwise} \end{cases}$, $\mathbf{z} = \mathbf{Q}\tilde{\mathbf{z}}$
$f_8(\mathbf{x}) = f^{\text{Rosenbrock}}_{\text{raw}}(\mathbf{z}) + f_{\text{opt}}$	$\mathbf{z} = \max\left(1, \frac{\sqrt{d}}{8}\right)(\mathbf{x} - \mathbf{x}_{\text{opt}}) + \mathbf{1}_d$, $\mathbf{z}_{\text{opt}} = \mathbf{1}_d$
$f_9(\mathbf{x}) = f^{\text{Rosenbrock}}_{\text{raw}}(\mathbf{z}) + f_{\text{opt}}$	$\mathbf{z} = \max\left(1, \frac{\sqrt{d}}{8}\right)\mathbf{R}\mathbf{x} + \frac{1}{2}\mathbf{1}_d$, $\mathbf{z}_{\text{opt}} = \mathbf{1}_d$
$f_{10}(\mathbf{x}) = f^{\text{Ellispsoid}}_{\text{raw}}(\mathbf{z}) + f_{\text{opt}}$	$\mathbf{z} = T^{\text{osz}}(\mathbf{R}(\mathbf{x} - \mathbf{x}_{\text{opt}}))$
$f_{11}(\mathbf{x}) = f^{\text{Discus}}_{\text{raw}}(\mathbf{z}) + f_{\text{opt}}$	$\mathbf{z} = T^{\text{osz}}(\mathbf{R}(\mathbf{x} - \mathbf{x}_{\text{opt}}))$
$f_{12}(\mathbf{x}) = f^{\text{BentCigar}}_{\text{raw}}(\mathbf{z}) + f_{\text{opt}}$	$\mathbf{z} = \mathbf{R}T^{\text{asy}}_{0.5}(\mathbf{R}((\mathbf{x} - \mathbf{x}_{\text{opt}}))$
$f_{13}(\mathbf{x}) = f^{\text{SharpRidge}}_{\text{raw}}(\mathbf{z}) + f_{\text{opt}}$	$\mathbf{z} = \mathbf{Q}\mathbf{\Lambda}^{10}\mathbf{R}(\mathbf{x} - \mathbf{x}_{\text{opt}})$
$f_{14}(\mathbf{x}) = f^{\text{DifferentPowers}}_{\text{raw}}(\mathbf{z}) + f_{\text{opt}}$	$\mathbf{z} = \mathbf{R}(\mathbf{x} - \mathbf{x}_{\text{opt}})$

Table 5.3: Transformed functions used in the large-scale test-bed as an extension to [Hansen et al., 2009]. The functions T^{osz} and T^{asy} are defined in Section 5.1.1 (respectively (5.6) and (5.8)) and $\mathbf{R}$ is an orthogonal (rotation) matrix. The raw definitions of the functions can be found in Table 5.1.

and the number of entries scales linearly in d only if $\text{avg}(s_i) = O(1)$ (as previously stated in Section 5.5.1).

Thanks to the associativity of the matrix product, we can make the vector $\mathbf{p}^{\text{left}}$ permute the rows of $\mathbf{B}$ (and also $\bar{\mathbf{B}}$) but the vector $\mathbf{p}^{\text{right}}$ permute the coordinates of $\mathbf{x}$, instead of the columns of $\mathbf{B}$, thus z_i would satisfy:

$$z_i = \sum_{j=\text{start}(p_i^{\text{left}})}^{\text{end}(p_i^{\text{left}})} \bar{b}_{p_i^{\text{left}}, j-\text{start}(p_i^{\text{left}})} x_{p_j^{\text{right}}} \quad , \tag{5.45}$$

$f_{15}(\mathbf{x}) =$	$f_{\text{raw}}^{\text{Rastrigin}}(\mathbf{z}) + f_{\text{opt}}$	$\mathbf{z} = \mathbf{R}\boldsymbol{\Lambda}^{10}\mathbf{Q}T_{0.2}^{\text{asy}}\left(T^{\text{osz}}\left(\mathbf{R}\left(\mathbf{x} - \mathbf{x}_{\text{opt}}\right)\right)\right)$
$f_{16}(\mathbf{x}) =$	$f_{\text{raw}}^{\text{Weierstrass}}(\mathbf{z}) + \frac{10}{d} f_{\text{pen}}(\mathbf{x}) + f_{\text{opt}}$	$\mathbf{z} = \mathbf{R}\boldsymbol{\Lambda}^{1/100}\mathbf{Q}T^{\text{osz}}(\mathbf{R}(\mathbf{x} - \mathbf{x}_{\text{opt}}))$
$f_{17}(\mathbf{x}) =$	$f_{\text{raw}}^{\text{SchaffersF7}}(\mathbf{z}) + 10 f_{\text{pen}}(\mathbf{x}) + f_{\text{opt}}$	$\mathbf{z} = \boldsymbol{\Lambda}^{10}\mathbf{Q}T_{0.5}^{\text{asy}}(\mathbf{R}(\mathbf{x} - \mathbf{x}_{\text{opt}}))$
$f_{18}(\mathbf{x}) =$	$f_{\text{raw}}^{\text{SchaffersF7}}(\mathbf{z}) + 10 f_{\text{pen}}(\mathbf{x}) + f_{\text{opt}}$	$\mathbf{z} = \boldsymbol{\Lambda}^{1000}\mathbf{Q}T_{0.5}^{\text{asy}}(\mathbf{R}(\mathbf{x} - \mathbf{x}_{\text{opt}}))$
$f_{19}(\mathbf{x}) =$	$f_{\text{raw}}^{\text{GriewankRosenbrock}}(\mathbf{z}) + f_{\text{opt}}$	$\mathbf{z} = \max\left(1, \frac{\sqrt{d}}{8}\right)\mathbf{R}\mathbf{x} + \frac{1}{2}\mathbf{1}_d$, $\mathbf{z}_{\text{opt}} = \mathbf{1}_d$
$f_{20}(\mathbf{x}) =$	$f_{\text{raw}}^{\text{Schwefel}}(\mathbf{z}) + 100 f_{\text{pen}}(\mathbf{z}/100) + f_{\text{opt}}$	$\hat{\mathbf{x}} = 2 \times \mathbf{1}_d(^{-}_{+}) \otimes \mathbf{x}$, $\hat{z}_1 = \hat{x}_1, \hat{z}_{i+1} = \hat{x}_{i+1} + 0.25(\hat{x}_i - x_i^{\text{opt}})$ $\mathbf{z} = 100(\boldsymbol{\Lambda}^{10}(\hat{\mathbf{z}} - \mathbf{x}_{\text{opt}}) + \mathbf{x}_{\text{opt}})$, $\mathbf{x}_{\text{opt}} = 4.2096874633/2\mathbf{1}_d(^{+}_{-})$
$f_{21}(\mathbf{x}) =$	$f_{\text{raw}}^{\text{Gallagher}}(\mathbf{z}) + f_{\text{pen}}(\mathbf{x}) + f_{\text{opt}}$	$n_{\text{peaks}} = 101$, $\mathbf{z} = \mathbf{C}_i^{1/2}\mathbf{R}(\mathbf{x} - \mathbf{y}_i) + \mathbf{y}_i$, $\mathbf{C}_i = \hat{\boldsymbol{\Lambda}}^{\alpha_i}/\alpha_i^{1/4}$, $[\hat{\boldsymbol{\Lambda}}^{\alpha_i}]_j = [\boldsymbol{\Lambda}^{\alpha_i}]_{\pi_1(j)}, 1 \le j \le d$, $\alpha_1 = 1000^2$, for $i \ge 2$, $\alpha_i = 1000^{2\frac{\pi_2(i)-2}{n_{\text{peaks}}-2}}$
$f_{22}(\mathbf{x}) =$	$f_{\text{raw}}^{\text{Gallagher}}(\mathbf{z}) + f_{\text{pen}}(\mathbf{x}) + f_{\text{opt}}$	same as f_{21} with $n_{\text{peaks}} = 21$,
$f_{23}(\mathbf{x}) =$	$f_{\text{raw}}^{\text{Katsuura}}(\mathbf{z}) + f_{\text{pen}}(\mathbf{x}) + f_{\text{opt}}$	$\mathbf{z} = \mathbf{Q}\boldsymbol{\Lambda}^{100}\mathbf{R}(\mathbf{x} - \mathbf{x}_{\text{opt}})$
$f_{24}(\mathbf{x}) =$	$f_{\text{raw}}^{\text{Lunacekbi-Rastrigin}}(\mathbf{z}) + 10^4 f_{\text{pen}}(\mathbf{x}) + f_{\text{opt}}$	$\mathbf{z} = \mathbf{Q}\boldsymbol{\Lambda}^{100}\mathbf{R}(\hat{\mathbf{x}} - \mu_0\mathbf{1}_d)$

Table 5.4: Follow up to Table 5.3. On the Gallagher functions, $\pi_1 \in \mathbb{N}^d$ and $\pi_2 \in \mathbb{N}^{n_{\text{peaks}}-2}$ are two random uniform permutations of the vectors $(1, \ldots, d)^\mathsf{T}$ and $(1, \ldots, n_{\text{peaks}} - 2)$ respectively. The i^{th} element of a permutation is designated by $\pi(i)$.

where $\text{start}(i) = j_i^{\text{first}}$ and $\text{stop}(i) = \text{start}(i) + s_i^{\text{row}}$ allow to loop through the elements of the row in question. The computational cost of the transformation is then at worse in $O(d \times \max_i(s_i))$.

5.6.4 CPU Timing

We now look into the actual cost of computing the introduced transformation in practice in terms of CPU time. Since we are in a large scale setting, we are particularly interested in its scaling with the problem dimension d. For this part, we use the C version of the COCO code (https://github.com/numbbo/coco).

Figure 5.7 shows the scaling of the CPU time per function evaluation spent in the transformation defined in (5.42) on all coordinates and implemented as explained in

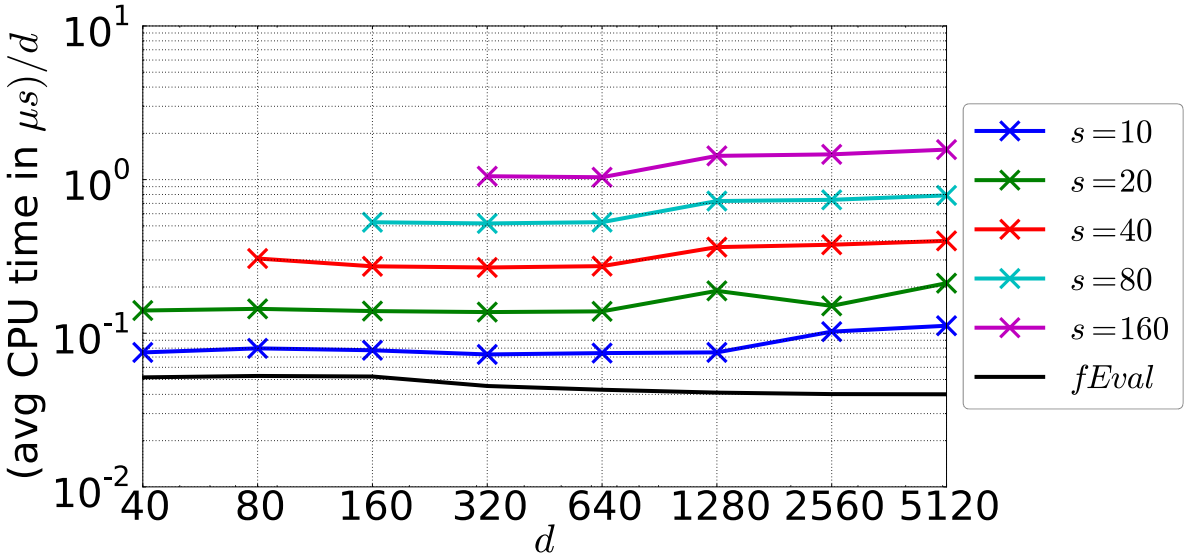

Figure 5.7: Average (over 10^4 samples) time needed to apply the transformation matrix (5.18) to a solution in micro-seconds, μs, divided by dimension for different values of the dimension d and of the block-size s. The solid black line shows the average cost of computing the raw ellipsoid function (see Table 5.1) in each dimension.

Section 5.6.3. We want to estimate the additional cost coming from the application of the permuted orthogonal block-diagonal transformation for different values of the parameters defining this transformation. The cost of applying the permutations depends only on d, so the parameters we consider are d and s (we assume blocks of equal sizes). The initialization cost is not taken into account here (which also depends on n_s) since the initialization is done only once per run.

We compute the averages of the time spent within the transformation $\mathbf{R} = \mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}$, that is the time between the start and the end of the code block responsible for the transformation. The transformations are sequentially applied in COCO (no overlapping) and the code is run sequentially (no parallelization) so the result is, in theory, independent of which other transformations are in play. We also show, for comparison, the average CPU time spent computing the raw Ellipsoid function ($f_{\text{raw}}^{\text{Elli}}$) on the same machine (solid black graph). These experiments were run on a MacBook Pro with a 2.3GHz quad-core Intel Core i7 processor and 8 GB of RAM.

As expected from (5.45), the needed CPU time scales linearly with d when the block size is kept constant and linearly with the block-size s for each given dimension. The linear scaling in s can be deducted from the similar distances, in a log-scale, between the graphs for different block-sizes since we considered block-sizes that increase by a constant factor 2. Given that we choose s to be constant for $d \geq 40$ (see Section 5.5), the overall CPU usage of the transformation is linear in d, which satisfies the cost related Property 1.

A linear scaling can still be unusable in practice because of a large constant multiplier. In Figure 5.7, the transformation with the largest considered block-size $s = 160$ takes around 40 times longer than computing the raw function value. We see a factor of around

10 for the chosen maximal block-size in (5.40), $s = 40$ which we deem reasonable.

5.7 Benchmarking Large-Scale Algorithms

In this section, we benchmark three stochastic large-scale continuous optimization algorithms sep-CMA-ES, LM-CMA-ES and VD-CMA-ES (see Section 2.2.1.3) on the newly introduced large scale coco test-bed (see Section 5.6).

Regarding the choice of problem dimensions, we choose to start by the largest mandatory dimension in BBOB-2009, which is $d = 20$ and then increase the dimension by doubling it. This allows backward compatibility with BBOB-2009 and makes the large-scale suite complement the standard dimension one. In Fact, the chosen parameter setting of s and $\gamma(d)$ in the large scale test-bed makes it so that for $d \leq 40$, the problems are the same than the ones in BBOB-2009; more so since the same methods are used to generate the orthogonal matrices in BBOB-2009 and the blocks of the block diagonal matrix in large scale (in dimensions 20 and 40, a single block is used and $\gamma(d) = 1$).

Figure 5.8 shows the scaling plots of the three algorithms in problem dimension. These plots show the Expected Running Time (ERT) to reach a target precision of 10^{-8} divided by dimension.

We note that functions f_1 to f_5 are separable functions that do not use the transformation matrix $\mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}$. However, on dimensions larger than 40 the scaling factor $\gamma(d)$ is applied and the Discus and Cigar functions no-longer have a single short, respectively long, axis.

On uni-modal separable functions (f_1, f_2 and f_5), we see the regular behavior of the three algorithms as they scale linearly in the problem dimension (compare with the horizontal dashed gray lines) except for a slight less-than-linear scaling of LM-CMA-ES on the sphere function and a more pronounced one for VD-CMA-ES on f_5. The remaining two functions are multi-modal with a number of local optima exponential in the problem dimension which makes the problem hard for the considered algorithms even without the rotations.

All but two (the original Rosenbrock function f_8 and the Schwefel function f_{20}) of the remaining problems (non-separable $f_6, \ldots, f_{24}$) use between one and two instances of the transformation matrix introduced in this chapter. We see that the algorithms manage to solve only a few of these problems. Mainly sep-CMA-ES and VD-CMA-ES (up to $d = 160$) on f_6, all three algorithms on f_8, VD-CMA-ES and LM-CMA-ES on f_9 and LM-CMA-ES on f_{11} and f_{12}. In addition, sep-CMA-ES manages to produce some successful runs on f_{17} (not all runs thought since the ERT is larger than the budget).

In Figure 5.9 and Figure 5.10, we look into another set of interesting plots generated using the COCO post-processing code. These figures, generally referred to as cumulative distribution figures, show the evolution of the proportion of problems solved by the algorithm on the y-axis versus the number of function evaluations divided by dimension shown in the x-axis [Hansen et al., 2010b]. In this context, a problem is defined by a target precision on a function (more precisely, a given instance of a function). In COCO, a total of 50 target precisions (difference to the optimal f-value) are chosen per instance of

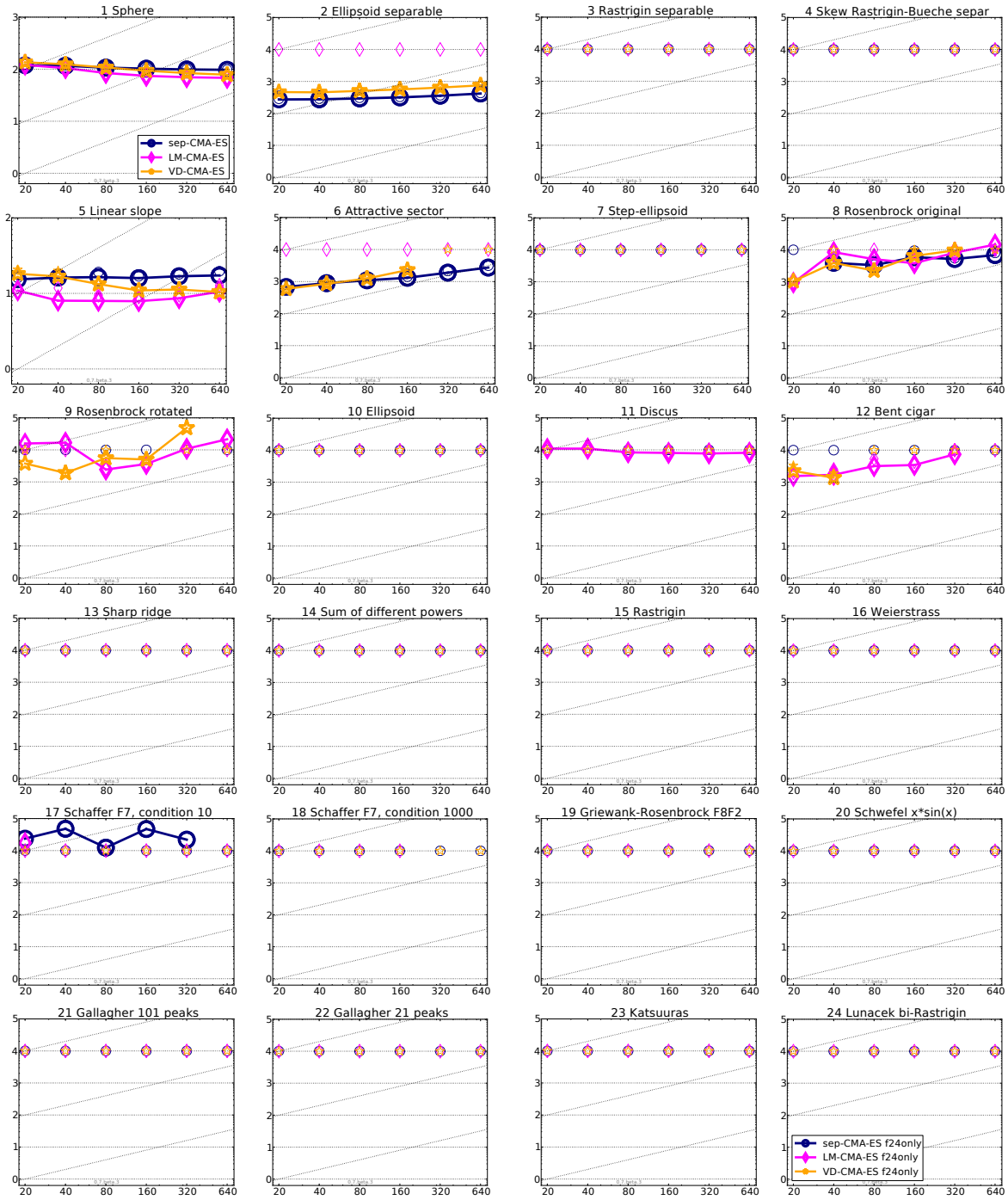

Figure 5.8: Expected running time divided by dimension versus dimension of sep-CMA-ES, LM-CMA-ES and VD-CMA-ES on the 24 functions of the COCO large-scale test-bed introduced in this chapter. The first 5 instances of each problem are used with a budget of $10^4 \times d$ (shown by the smaller markers). The rest is the same as Figure 3.11 with a different set of dimension values and without the artificial algorithm (no data yet).

a function; these targets are uniformly spaced on the log-scale between 10^{-8} and 10^{2}. The plots are shown for each function separately and aggregated over the different instances of a same function. One can also generate plots aggregated over different functions of a same group (separable, low condition number...) or simply over all the functions to have a global overview of the performance. However, we avoid aggregating results on different dimensions and rely on plots such as the ones in Figure 5.8 to study the scaling of an algorithm with the problem dimension (the same figure can be plotted for different target values which is actually done when not comparing algorithms).

First, in Figure 5.9, we compare the results on rotated and non-rotated version of the Ellipsoid and Rosenbrock functions. We see that LM-CMA-ES is the least affected by the orthogonal transformation, managing to have similar performance on both functions. sep-CMA-ES, however, suffers greatly from the rotation (actually, the matrices are orthogonal not necessarily proper rotation matrices so rotation matrices here include both proper rotation with a determinant of 1 and improper rotations with a determinant of -1) of the search-space managing to efficiently solve the non-rotated versions and barely solving a small proportion of problems on f_9 and none on f_{10}. VD-CMA-ES seems to take a longer time to adapt its covariance matrix on f_{10} and suffers greatly from the dimension but manages to still solve f_9 except for $d = 640$.

On other rotated functions shown in Figure 5.10, other than what was already said on Figure 5.8, that is LM-CMA-ES handles well the transformation matrix, VD-CMA-ES to a lower extent and suffering significantly from increasing d and sep-CMA-ES being unable to solve the rotated problems; we mainly notice that f_6 is surprisingly hard for LM-CMA-ES when VD-CMA-ES and especially sep-CMA-ES perform well on it.

As to be expected, sep-CMA-ES only manages to solve non-rotated problems except on the Attractive Sector where it solves the problem with a quasi linear scaling (see Figure 5.8). In addition, the rotated problems seem to be so hard, separability wise, that sep-CMA-ES reaches only a few of the targets. The good performance on f_6 is mainly due to the relatively low overall condition number of the function, and thus even lower block-condition numbers (see right plot of Figure 5.1).

Note that, because of the parameter setting chosen, the permuted block-separability only appears for $d > 40$ so the results on dimensions 20 and 40 are the same as the ones we would see in BBOB-2009. LM-CMA-ES retains its quite good performance on some of the functions by managing to learn enough short axes to reach the target values. However, it still suffers on functions such as the Attractive Sector and Sharp Ridge despite its good performance on the Bent Cigar function which requires a similar approach.

VD-CMA-ES seems to overall suffer more from the increasing dimensionality. This is probably due to the constant number of vectors it adapts in its covariance matrix contrarily to LM-CMA-ES where this number of vectors is dependent on the problem dimension. We expect the recently introduced generalized version of VD-CMA-ES, VkD-CMA-ES [Akimoto and Hansen, 2016b], to perform better on this test-suite, especially with the even more recent proposed k (number of vectors) online adaptation scheme [Akimoto and Hansen, 2016a].

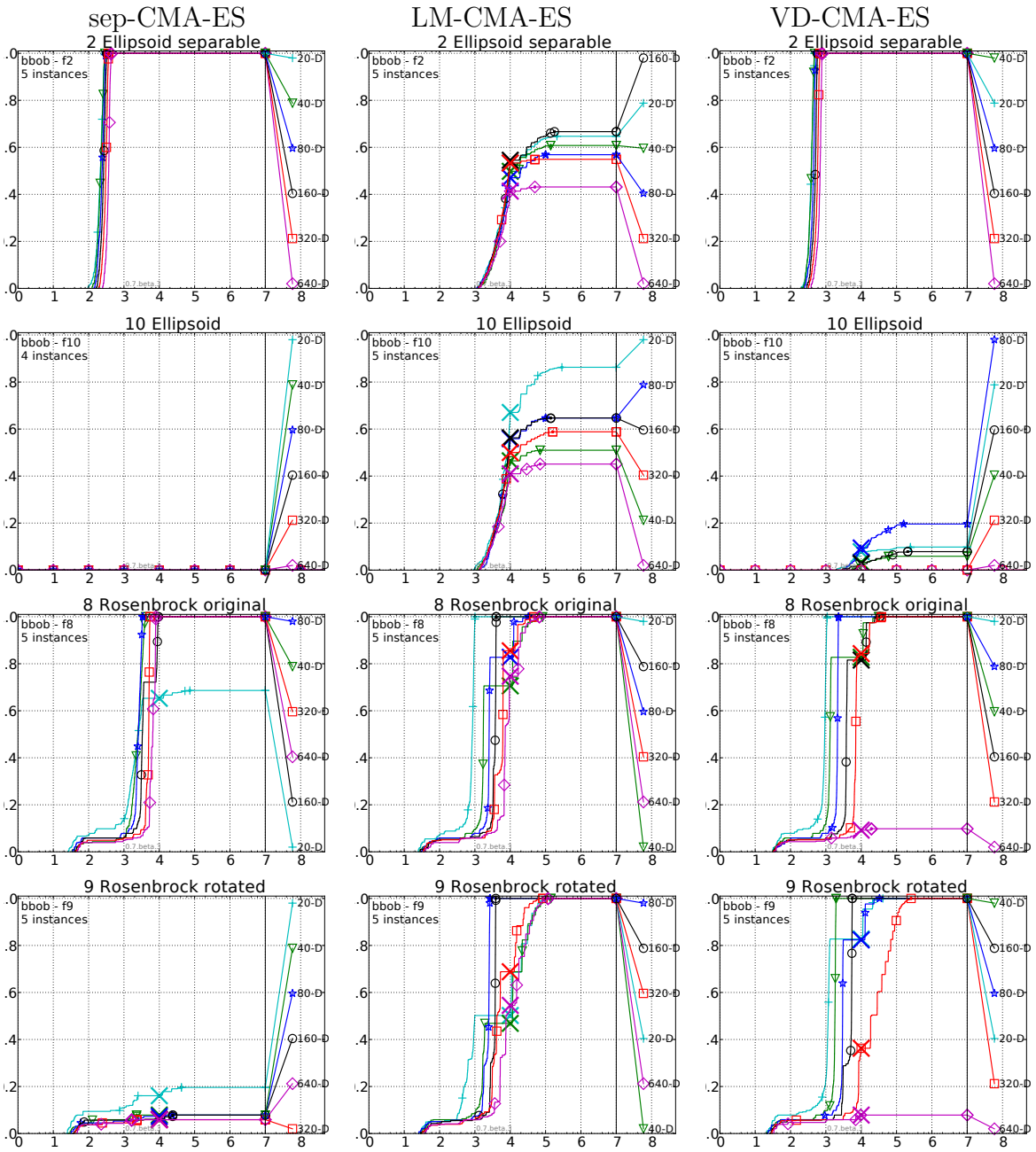

Figure 5.9: Evolution of the proportion of solved problems with number of function evaluations divided by the problem dimension.

The rest of the plots on the remaining functions can be found on the Appendix of this Thesis.

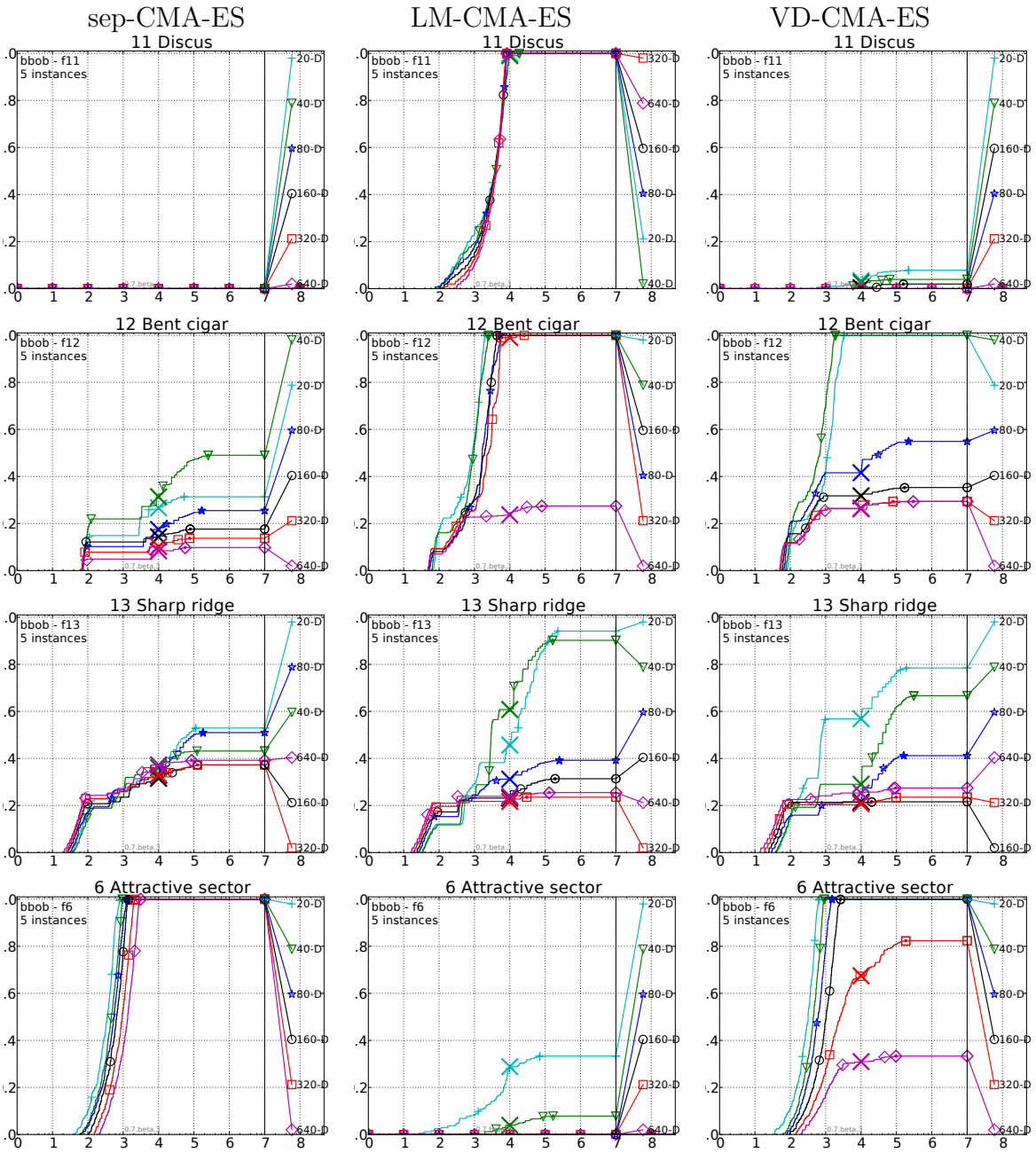

Figure 5.10: Follow up to Figure 5.9 on other functions.

5.8 Conclusion

This chapter allowed the extension of the COCO framework, and more particularly the BBOB-2009 testbed, to a larger scale in a practical and cost-wise reasonable way. The proposed benchmark acts as a natural expansion to the already present low to medium dimension test-beds and allows, given the parameter setting, backward compatibility.

Being computationally efficient is not enough for a test-suite to be relevant, the proposed problems need to be representative enough of the problems one is expected to

encounter in real life, and more specifically, representative of the difficulties and challenges these problems offer. We ensure this by using orthogonal transformation that, while being sparse, manage to preserve a number of properties a full rotation matrix offers. In addition to conserving the eigenvalues when present, the proposed transformation allows to introduce non-separability in a similar way to full rotations. Also, the parameterization of the proposed transformation is such that the level of non-separability introduced can be controlled by two parameters: the block-size and the swap-range.

We deem the CPU effort of applying this transformation reasonable given the block-size choice. The space complexity of the transformation is similar, being only linear in the problem dimension but depending also on the block-sizes. In practice, these costs are reasonable in the sense that they do not introduce a bottleneck where an algorithm spends most of its time evaluating the function, especially when said algorithm is large-scale based and thus has a reduced complexity in the problem dimension to accommodate large-scale settings. This would be the case have we used, instead, the original full orthogonal matrices to introduce non-separability and coordinate system independence.

We also generalized the Discus, Cigar and Sharp Ridge functions in a way that makes these functions more naturally interpretable and more importantly, more relevant in a large-scale setting. Having a constant, and in this case small, number of axes with a different length means that as the problem dimension increases, these function tend to resemble a sphere function as the relative contribution of the differently sized axes diminishes. This effect is witnessed more on the Cigar function since the differently sized axes are long ones. The new formulation allows to have a constant *ratio* of these differently sized axes. It can be seen as a generalization of the Two-Axes function in which half the axes are short and half the axes long (with the same lengths of the short axes and the same lengths of the long axes).

We normalize the raw functions (that are not already normalized) by the problem dimension in order to have a similar level of difficulty across the dimensions; another property which becomes more relevant in a large-scale setting.

The orthogonal transformation in our case uses a core block-diagonal matrix that is permuted on both sizes in order to hide its block-diagonal structure. We have seen the effectiveness of these permutations by testing on a number of functions sep-CMA-ES which solves, when the block-condition numbers are low enough, non-permuted versions but not permuted ones.

One property that the proposed transformation does not conserve completely, when compared to the original full matrix, is coordinate-system independence. This is both because the core block-diagonal matrix is not coordinate system independent, and because the introduced permutations are not uniformly chosen from the set of all possible permutations. A choice we make in order to allow control over the difficulty of the resulting problem and the underlying variance in this difficulty and that (this choice) does not allow complete unbiasedness with regards to the starting positions of the variables, given a reasonable parameter choice (having $s = d$ puts us back in the full orthogonal matrix case, regardless of the permutations).

Overall, the proposed approach to large-scale benchmarking by extending on already

established benchmarks produced the desired results. The resulting test-suite has the advantage of being quite similar to the original one, retaining most of its properties which makes interpreting the results easier. This means that it also reproduces these properties that we expect to see in real-world problems, while introducing the large-scale component. It also represents a good class of problems separability-wise, since most of the time, we do not expect all the variables of a large-scale problem to be directly dependent, but rather form groups that are not necessarily aligned with the order of the variables neither as presented to the algorithm nor inside the problem. The benchmarking of three large-scale evolution strategies did not result in any anomalies. None of the algorithms managed to perform well on all problems and many properties we observed in smaller dimension problems persist. There is, also, no problem with regards to the interaction with other transformations or the raw function definitions since the transformations are applied in an independent fashion.

It would definitely be interesting to extend the large-scale benchmarks even more since larger dimensions allows for properties to appear that are generally not seen or, at best, ignored or neglected (because of their small effect) in small to medium dimensions. One such property is the effective and epsilon-effective dimensionalities of a problem (see Chapter 4). After the design of a large-scale test suite which is closely similar to its lower-dimension counterpart, the next step would be to add, amongst others, test suites or at least classes of problems whose effective and/or ε-effective dimensionalities are low.

Chapter 6

Final Conclusion

This thesis brings three main contributions to continuous black-box optimization in general and to the large-scale scenario in particular.

First, a new success-based step-size adaptation mechanism for evolution strategies, the Median Success Rule, is proposed in Chapter 3. This method showed comparative results to the widely used Cumulative Step-size Adaptation, notably on the COCO noiseless testbed. In addition, studying the impact of choosing a non-optimal comparison index (the main parameter of the strategy) showed that the method is robust, and still converges for a wide range of values on a number of typical functions.

Being a success-based method, MSR, and in comparison to CSA, relies on less restrictive properties of the distribution of solutions. It does not require the solutions to be normally distributed neither does it need access to the steps in the isotropic space. It does, however, rely on a correlation between the defined success probability and the *desired* change in the step-size (ideally, toward an optimal step-size).

One other important feature of MSR is its low computational cost. It scales linearly with the population size, a parameter that is set by the user unlike the dimension which is inherent to the problem. It is true that the population size is generally set depending on the problem dimension; but generally with values not exceeding a linear scaling in it. In addition, the user can always set it according to the complexity constraints at hand. This makes MSR ideal for large-scale algorithms where the scaling of the computational cost in the dimension of the problem is critical. Not only that, but MSR is also expected to remain effective when the population size and the number of parents in recombination-based strategies is high.

In Chapter 4, we formalize a method of constructing low effective dimension problems from commonly used continuous benchmarking functions. We also extend the notion of low effective dimension to become low epsilon-effective dimension which allows for all the variables to contribute to a relatively low proportion of the fitness defined by a parameter ε.

A sub-space CMA-ES variant, SS-CMA-ES, is proposed to tackle specifically these newly introduced problems. It relies on optimizing in an embedded sub-space of dimension lower than that of the problem. When comparing the performance of SS-CMA-ES to default CMA-ES and other, large-scale, algorithms, a number of observations were made.

First, an important result was that using CSA as a step-size adaptation method instead of MSR or the Two Point Adaptation, which are success-based rules, resulted in worse performance of the algorithm and slower convergence rates to the optimum. This is due to CSA comparing the steps realized by the algorithms with the random steps. However, these random steps are assumed to happen in the full dimension of the problem when the problem has a lower effective dimension. This prevents the algorithm from taking advantage on the lower effective dimension of the problem, and thus, in comparison to other methods that do, makes it slower. On the other hand, SS-CMA-ES, using MSR and TPA, and Limited Memory CMA-ES (LM-CMA-ES) performed remarkably well on several low effective and low epsilon-effective dimension test problems. While this was expected of SS-CMA-ES on low effective dimension problems since the algorithm was designed for this class of problems specifically, it also managed to *solve* low epsilon-effective dimension problems even though its search is carried out in a search space of lower dimension while the problem has full effective dimension. In fact, with target values that require the fitness not to deteriorate (in comparison to that of the initial solution) on the non-effective part, algorithms are encouraged to optimize the effective space first, then fine tune in the later stages of optimization the deterioration of the non-effective fitness that might have happened when optimizing the effective part.

Chapter 5 focused on extending the COCO noiseless test-suite, to large scales. All the problems of the suite are defined for any dimension. However, in a large-scale setting, quadratic complexities that come from the use of full orthogonal matrices in order to introduce non-separability need to be reduced. This is done, while keeping the main effects of the rotation matrices, by replacing full orthogonal matrices by permuted orthogonal block-diagonal matrices.

We tested three large-scale optimization algorithms on this newly introduced test-bed. These algorithms that rely on a restricted model of the covariance matrixare: sep-CMA-ES, LM-CMA-ES and VD-CMA-ES. In general, we observed that the new transformation introduces sufficiently challenging-enough problems whose structure can not be exploited by these algorithms. The permutations, more specifically the permutation that is applied to the rows of the block diagonal matrix, allow to *hide* the block structure of the underlying matrix, and thus make it harder to exploit in comparison to other approaches in the literature that only rely on permuting the columns of the matrix. The effect of this permutation is seen on the performance of sep-CMA-ES which manages to solve functions with non-permuted block-diagonal transformation matrices up to certain block condition-numbers but fails when the permutation of the rows is introduced. The other permutation (on the columns) that simply changes the order of the variables as they are presented to the algorithm has no effect on the performance of sep-CMA-ES and most evolution strategies.

The performance assessment of the above mentioned algorithms shows a *smooth* transition between dimensions 20 and 40 that reproduce the problems of the original small to medium COCO test-bed and the higher dimensions that include the actual permuted block-diagonal transformation. This means that the proposed method is a natural extension of the test-bed that does not break its properties.

In addition to the contributions described above, this thesis paves the way to a number of future works in its area. The Median Success Rule can further be improved with better tuned damping parameters and learning rates. In the context of low effective dimensions, a completely black-box adaptive SS-CMA-ES can be a competitive method for efficiently tackling low effective and low epsilon-effective dimension problems. It can even be considered for more general classes of problems when the intended point is not to *find* the optimal solution but just to produce a relatively high quality solution within a limited budget and tight complexity constraints. A basic approach of starting with small optimization sub-space dimensions and then increase after each failure was already suggested in this thesis. However, one can also consider more sophisticated approaches that allow the estimation of the effective dimension of a problem online similarly to what is done in [Akimoto and Hansen, 2016a] for the number of vectors. The large-scale testbed can definitely be enriched with a number of extensions. Allowing differently sized and/or overlapping blocks in the block-diagonal transformation-matrix proposed in this thesis is a first extension that should not be complicated to implement and test.

Further down the line, it would be interesting to have different test-suites for large-scale optimization that include other classes of problems such as the low effective dimension and low epsilon-effective dimension problems introduced in this thesis. Even though it seems natural to make the large-scale test suite of Chapter 5 a low (ε-)effective dimension one given the results of Chapter 4, we opted, as the first large-scale extension of the bbob-2009 testbed, for a more *natural* generalization that does not deviate much from the original test suite. This, however, does not diminish the importance of including such a class of problems in the future variants and extensions of the large-scale test suite. This is becoming especially relevant in the recent years since large-scale optimization is increasingly used in machine-learning applications that deal with big-data problems. These problems have, generally, sparse, low effective dimension and block-structured dependency functions to optimize (generally using gradient-based/inspired approaches).

Bibliography

[Ait Elhara et al., 2013] Ait Elhara, O., Auger, A., and Hansen, N. (2013). A Median Success Rule for Non-Elitist Evolution Strategies: Study of Feasibility. In Christian, B. e. a., editor, *Genetic and Evolutionary Computation Conference*, pages 415–422, Amsterdam, Netherlands. ACM, ACM Press. Author version with appendix.

[Ait Elhara et al., 2016] Ait Elhara, O., Auger, A., and Hansen, N. (2016). Permuted Orthogonal Block-Diagonal Transformation Matrices for Large Scale Optimization Benchmarking. In *GECCO 2016*, Denver, United States.

[Akat and Gazi, 2008] Akat, S. B. and Gazi, V. (2008). Particle swarm optimization with dynamic neighborhood topology: three neighborhood strategies and preliminary results. In *2008 IEEE Swarm Intelligence Symposium*.

[Akimoto et al., 2014] Akimoto, Y., Auger, A., Hansen, N., et al. (2014). Comparison-based natural gradient optimization in high dimension. In *Genetic and Evolutionary Computation Conference GECCO'14*.

[Akimoto and Hansen, 2016a] Akimoto, Y. and Hansen, N. (2016a). Online model selection for restricted covariance matrix adaptation. In *Parallel Problem Solving from Nature 2016*.

[Akimoto and Hansen, 2016b] Akimoto, Y. and Hansen, N. (2016b). Projection-based restricted covariance matrix adaptation for high dimension. In *Genetic and Evolutionary Computation Conference 2016*.

[Arnold, 2005] Arnold, D. V. (2005). Optimal weighted recombination. In *Foundations of Genetic Algorithms*, pages 215–237. Springer Verlag.

[Arnold, 2006] Arnold, D. V. (2006). Weighted multirecombination evolution strategies. *Theoretical Computer Science*, 361(1):18–37.

[Arnold and MacLeod, 2008] Arnold, D. V. and MacLeod, A. (2008). Step length adaptation on ridge functions. *Evol. Comput.*, 16(2):151–184.

[Asotsky et al., 2006] Asotsky, D., Myshetskaya, E., et al. (2006). The average dimension of a multidimensional function for quasi-monte carlo estimates of an integral. *Computational Mathematics and Mathematical Physics*, 46(12):2061–2067.

[Atamna, 2015] Atamna, A. (2015). Benchmarking ipop-cma-es-tpa and ipop-cma-es-msr on the bbob noiseless testbed. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1135–1142. ACM.

[Atkinson et al., 1997] Atkinson, C., Moore, A., and Stefan, S. (1997). Locally weighted learning. *Artif Intell Rev*, 11(1-5):11–73.

[Auger et al., 2011] Auger, A., Brockhoff, D., and Hansen, N. (2011). Mirrored sampling in evolution strategies with weighted recombination. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 861–868. ACM.

[Auger et al., 2013] Auger, A., Brockhoff, D., and Hansen, N. (2013). Benchmarking the local metamodel cma-es on the noiseless bbob'2013 test bed. In *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, pages 1225–1232. ACM.

[Auger and Hansen, 2005a] Auger, A. and Hansen, N. (2005a). Performance evaluation of an advanced local search evolutionary algorithm. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1777–1784. IEEE.

[Auger and Hansen, 2005b] Auger, A. and Hansen, N. (2005b). A restart CMA evolution strategy with increasing population size. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1769–1776. IEEE.

[Auger and Hansen, 2011] Auger, A. and Hansen, N. (2011). Theory of evolution strategies: A new perspective. In Auger, A. and Doerr, B., editors, *Theory of Randomized Search Heuristics: Foundations and Recent Developments*, chapter 10, pages 289–325. World Scientific Publishing Company.

[Auger and Hansen, 2013a] Auger, A. and Hansen, N. (2013a). Linear convergence of comparison-based step-size adaptive randomized search via stability of markov chains. *arXiv preprint arXiv:1310.7697*.

[Auger and Hansen, 2013b] Auger, A. and Hansen, N. (2013b). Linear convergence on positively homogeneous functions of a comparison based step-size adaptive randomized search: the (1+1)-ES with generalized one-fifth success rule. *arXiv preprint arXiv:1310.8397*.

[Auger et al., 2004] Auger, A., Schoenauer, M., and Vanhaecke, N. (2004). LS-CMA-ES: A second-order algorithm for covariance matrix adaptation. In *International Conference on Parallel Problem Solving from Nature*, pages 182–191. Springer.

[Baluja and Caruana, 1995] Baluja, S. and Caruana, R. (1995). Removing the genetics from the standard genetic algorithm. In *Machine Learning: Proceedings of the Twelfth International Conference*, pages 38–46.

[Banzhaf and Langdon, 2002] Banzhaf, W. and Langdon, W. B. (2002). Some considerations on the reason for bloat. *Genetic Programming and Evolvable Machines*, 3(1):81–91.

[Bergstra and Bengio, 2012] Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305.

[Berthier and Teytaud, 2015] Berthier, V. and Teytaud, O. (2015). On the codimension of the set of optima: large scale optimisation with few relevant variables. In *International Conference on Artificial Evolution (Evolution Artificielle)*, pages 234–247. Springer.

[Beyer and Schwefel, 2002] Beyer, H.-G. and Schwefel, H.-P. (2002). Evolution strategies–a comprehensive introduction. *Natural computing*, 1(1):3–52.

[Birattari et al., 2010] Birattari, M., Yuan, Z., Balaprakash, P., and Stützle, T. (2010). F-race and iterated f-race: An overview. In *Experimental methods for the analysis of optimization algorithms*, pages 311–336. Springer.

[Bleuler et al., 2001] Bleuler, S., Brack, M., Thiele, L., and Zitzler, E. (2001). Multiobjective genetic programming: Reducing bloat using spea2. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 1, pages 536–543. IEEE.

[Bosman et al., 2013] Bosman, P. A., Grahl, J., and Thierens, D. (2013). Benchmarking parameter-free amalgam on functions with and without noise. *Evolutionary computation*, 21(3):445–469.

[Bottou et al., 2016] Bottou, L., Curtis, F. E., and Nocedal, J. (2016). Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838*.

[Bouzarkouna et al., 2010] Bouzarkouna, Z., Auger, A., and Ding, D. Y. (2010). Investigating the local-meta-model cma-es for large population sizes. In *European Conference on the Applications of Evolutionary Computation*, pages 402–411. Springer.

[Bouzarkouna et al., 2012] Bouzarkouna, Z., Ding, D. Y., and Auger, A. (2012). Well placement optimization with the covariance matrix adaptation evolution strategy and meta-models. *Computational Geosciences*, 16(1):75–92.

[Brest et al., 2008] Brest, J., Zamuda, A., Boskovic, B., Maucec, M., and Zumer, V. (2008). High-dimensional real-parameter optimization using self-adaptive differential evolution algorithm with population size reduction. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 2032–2039. IEEE.

[Brockhoff et al., 2010] Brockhoff, D., Auger, A., Hansen, N., Arnold, D. V., and Hohm, T. (2010). Mirrored sampling and sequential selection for evolution strategies. In *Parallel Problem Solving from Nature, PPSN XI*, pages 11–21. Springer.

[Brockhoff et al., 2016] Brockhoff, D., Tušar, T., Tušar, D., Wagner, T., Hansen, N., and Auger, A. (2016). Biobjective performance assessment with the coco platform. *arXiv preprint arXiv:1605.01746*.

[Broyden, 1970] Broyden, C. G. (1970). The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90.

[Caflisch et al., 1997] Caflisch, R. E., Morokoff, W. J., and Owen, A. B. (1997). *Valuation of mortgage backed securities using Brownian bridges to reduce effective dimension*. Department of Mathematics, University of California, Los Angeles.

[Chandrasekaran et al., 2012] Chandrasekaran, V., Recht, B., Parrilo, P. A., and Willsky, A. S. (2012). The convex geometry of linear inverse problems. *Foundations of Computational mathematics*, 12(6):805–849.

[Chen et al., 2011] Chen, W., Weise, T., Yang, Z., and Tang, K. (2011). Large-scale global optimization using cooperative coevolution with variable interaction learning. *Parallel Problem Solving from Nature–PPSN XI*, pages 300–309.

[Claudiu Ciresan et al., 2010] Claudiu Ciresan, D., Meier, U., Gambardella, L. M., and Schmidhuber, J. (2010). Deep big simple neural nets excel on handwritten digit recognition. *arXiv preprint arXiv:1003.0358*.

[Coates et al., 2011] Coates, A., Ng, A. Y., and Lee, H. (2011). An analysis of single-layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics*, pages 215–223.

[De Jong et al., 2001] De Jong, E., Watson, R., and Pollack, J. (2001). Reducing bloat and promoting diversity using multi-objective methods.

[Dean et al., 2012] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. (2012). Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231.

[Dong et al., 2013] Dong, W., Chen, T., Tiňo, P., and Yao, X. (2013). Scaling up estimation of distribution algorithms for continuous optimization. *IEEE Transactions on Evolutionary Computation*, 17(6):797–822.

[Edelman, 1989] Edelman, A. (1989). *Eigenvalues and Condition Numbers of Random Matrices*. PhD thesis, Massachusetts Institute of Technology.

[Ekárt and Németh, 2002] Ekárt, A. and Németh, S. Z. (2002). Maintaining the diversity of genetic programs. In *European Conference on Genetic Programming*, pages 162–171. Springer.

[Fasano and Lucidi, 2009] Fasano, G. and Lucidi, S. (2009). A nonmonotone truncated newton–krylov method exploiting negative curvature directions, for large scale unconstrained optimization. *Optimization Letters*, 3(4):521–535.

[Fletcher, 1970] Fletcher, R. (1970). A new approach to variable metric algorithms. *The computer journal*, 13(3):317–322.

[Foli et al., 2006] Foli, K., Okabe, T., Olhofer, M., Jin, Y., and Sendhoff, B. (2006). Optimization of micro heat exchanger: Cfd, analytical approach and multi-objective evolutionary algorithms. *International Journal of Heat and Mass Transfer*, 49(5):1090–1099.

[Girosi, 1998] Girosi, F. (1998). An equivalence between sparse approximation and support vector machines. *Neural computation*, 10(6):1455–1480.

[Glasmachers et al., 2010] Glasmachers, T., Schaul, T., Yi, S., Wierstra, D., and Schmidhuber, J. (2010). Exponential natural evolution strategies. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 393–400. ACM.

[Goldfarb, 1970] Goldfarb, D. (1970). A family of variable-metric methods derived by variational means. *Mathematics of computation*, 24(109):23–26.

[Han and Fan, 2010] Han, M. and Fan, J. (2010). Particle swarm optimization using dynamic neighborhood topology for large scale optimization. In *Intelligent Control and Automation (WCICA), 2010 8th World Congress on*, pages 3138–3142. IEEE.

[Hansen, 2008] Hansen, N. (2008). Cma-es with two-point step-size adaptation. *arXiv preprint arXiv:0805.0231*.

[Hansen, 2009] Hansen, N. (2009). Benchmarking a bi-population cma-es on the bbob-2009 function testbed. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2389–2396. ACM.

[Hansen et al., 2014] Hansen, N., Atamna, A., and Auger, A. (2014). How to assess step-size adaptation mechanisms in randomised search. In *Parallel Problem Solving from Nature–PPSN XIII*, pages 60–69. Springer.

[Hansen et al., 2010a] Hansen, N., Auger, A., Finck, S., and Ros, R. (2010a). Real-Parameter Black-Box Optimization Benchmarking 2010: Experimental Setup. Research Report RR-7215, INRIA.

[Hansen et al., 2010b] Hansen, N., Auger, A., Ros, R., Finck, S., and Pošík, P. (2010b). Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, pages 1689–1696. ACM.

[Hansen et al., 2009] Hansen, N., Finck, S., Ros, R., and Auger, A. (2009). Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Research Report RR-6829, INRIA.

[Hansen and Kern, 2004] Hansen, N. and Kern, S. (2004). Evaluating the cma evolution strategy on multimodal test functions. In *International Conference on Parallel Problem Solving from Nature*, pages 282–291. Springer.

[Hansen et al., 2003] Hansen, N., Müller, S. D., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18.

[Hansen and Ostermeier, 1996] Hansen, N. and Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 312–317. IEEE.

[Hansen and Ostermeier, 2001] Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195.

[Hansen et al., 1995] Hansen, N., Ostermeier, A., and Gawelczyk, A. (1995). On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation. In *ICGA*, pages 57–64.

[Hansen and Ros, 2010] Hansen, N. and Ros, R. (2010). Benchmarking a weighted negative covariance matrix update on the bbob-2010 noiseless testbed. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, pages 1673–1680. ACM.

[Hansen et al., 2011] Hansen, N., Ros, R., Mauny, N., Schoenauer, M., and Auger, A. (2011). Impacts of invariance in search: When cma-es and pso face ill-conditioned and non-separable problems. *Applied Soft Computing*, 11(8):5755–5769.

[Howell, 2008] Howell, R. R. (2008). On asymptotic notation with multiple variables. Technical report, Citeseer.

[Hsieh et al., 2008a] Hsieh, C.-J., Chang, K.-W., Lin, C.-J., Keerthi, S. S., and Sundararajan, S. (2008a). A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th international conference on Machine learning*, pages 408–415. ACM.

[Hsieh et al., 2008b] Hsieh, S.-T., Sun, T.-Y., Liu, C.-C., and Tsai, S.-J. (2008b). Solving large scale global optimization using improved particle swarm optimizer. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 1777–1784. IEEE.

[Hu and Eberhart, 2002] Hu, X. and Eberhart, R. (2002). Multiobjective optimization using dynamic neighborhood particle swarm optimization. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 2, pages 1677–1681. IEEE.

[Hutter, 2009] Hutter, F. (2009). *Automated configuration of algorithms for solving hard computational problems*. PhD thesis, University of British Columbia.

[Igel et al., 2007] Igel, C., Hansen, N., and Roth, S. (2007). Covariance matrix adaptation for multi-objective optimization. *Evolutionary computation*, 15(1):1–28.

[James and Russell, 1995] James, K. and Russell, E. (1995). Particle swarm optimization. In *Proceedings of 1995 IEEE International Conference on Neural Networks*, pages 1942–1948.

[Jastrebski and Arnold, 2006] Jastrebski, G. A. and Arnold, D. V. (2006). Improving evolution strategies through active covariance matrix adaptation. In *2006 IEEE International Conference on Evolutionary Computation*, pages 2814–2821. IEEE.

[Joachims, 2006] Joachims, T. (2006). Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226. ACM.

[Kabán et al., 2015] Kabán, A., Bootkrajang, J., and Durrant, R. J. (2015). Toward large-scale continuous eda: A random matrix theory perspective. *Evolutionary computation*.

[Kearns et al., 2002] Kearns, M., Mansour, Y., and Ng, A. Y. (2002). A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 49(2-3):193–208.

[Kern et al., 2006] Kern, S., Hansen, N., and Koumoutsakos, P. (2006). Local metamodels for optimization using evolution strategies. In *Parallel Problem Solving from Nature-PPSN IX*, pages 939–948. Springer.

[Knight and Lunacek, 2007] Knight, J. N. and Lunacek, M. (2007). Reducing the space-time complexity of the cma-es. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 658–665. ACM.

[Korenaga et al., 2007] Korenaga, T., Hatanaka, T., and Uosaki, K. (2007). Performance improvement of particle swarm optimization for high-dimensional function optimization. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 3288–3293. IEEE.

[Krause and Glasmachers, 2015] Krause, O. and Glasmachers, T. (2015). A cma-es with multiplicative covariance matrix updates. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 281–288. ACM.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Langdon and Poli, 1998] Langdon, W. B. and Poli, R. (1998). Fitness causes bloat: Mutation. In *European Conference on Genetic Programming*, pages 37–48. Springer.

[Larranaga and Lozano, 2002] Larranaga, P. and Lozano, J. A. (2002). *Estimation of distribution algorithms: A new tool for evolutionary computation*, volume 2. Springer Science & Business Media.

[Li et al., 2013] Li, X., Tang, K., Omidvar, M. N., Yang, Z., Qin, K., and China, H. (2013). Benchmark functions for the cec 2013 special session and competition on large-scale global optimization. *gene*, 7(33):8.

[Li and Yao, 2009] Li, X. and Yao, X. (2009). Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 1546–1553. IEEE.

[Li and Yao, 2012] Li, X. and Yao, X. (2012). Cooperatively coevolving particle swarms for large scale optimization. *IEEE Transactions on Evolutionary Computation*, 16(2):210–224.

[Liao et al., 2005] Liao, L.-Z., Qi, L., and Tam, H. W. (2005). A gradient-based continuous method for large-scale optimization problems. *Journal of Global Optimization*, 31(2):271–286.

[Lin and Moré, 1999] Lin, C.-J. and Moré, J. J. (1999). Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization*, 9(4):1100–1127.

[Lin et al., 2008] Lin, C.-J., Weng, R. C., and Keerthi, S. S. (2008). Trust region newton method for logistic regression. *Journal of Machine Learning Research*, 9(Apr):627–650.

[Liu and Nocedal, 1989] Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528.

[Liu and Li, 2010] Liu, Y. and Li, S. (2010). Differential evolution with neighborhood search. In *Computational Intelligence and Natural Computing Proceedings (CINC), 2010 Second International Conference on*, volume 1, pages 76–79. IEEE.

[Liu et al., 2001] Liu, Y., Yao, X., Zhao, Q., and Higuchi, T. (2001). Scaling up fast evolutionary programming with cooperative coevolution. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 2, pages 1101–1108. IEEE.

[Liu et al., 2009] Liu, Y., Zhao, Q., Shao, Z., Shang, Z., and Sui, C. (2009). Particle swarm optimizer based on dynamic neighborhood topology. In *International Conference on Intelligent Computing*, pages 794–803. Springer.

[Loshchilov, 2013] Loshchilov, I. (2013). CMA-ES with restarts for solving cec 2013 benchmark problems. In *2013 IEEE Congress on Evolutionary Computation*, pages 369–376. Ieee.

[Loshchilov, 2014] Loshchilov, I. (2014). A computationally efficient limited memory CMA-ES for large scale optimization. *arXiv preprint arXiv:1404.5520*.

[Loshchilov, 2015] Loshchilov, I. (2015). LM-CMA: An alternative to L-BFGS for large-scale black box optimization. *Evolutionary computation*.

[Loshchilov et al., 2010] Loshchilov, I., Schoenauer, M., and Sebag, M. (2010). Comparison-based optimizers need comparison-based surrogates. In *International Conference on Parallel Problem Solving from Nature*, pages 364–373. Springer.

[Loshchilov et al., 2012a] Loshchilov, I., Schoenauer, M., and Sebag, M. (2012a). Black-box optimization benchmarking of ipop-saacm-es and bipop-saacm-es on the bbob-2012 noiseless testbed. In *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, pages 175–182. ACM.

[Loshchilov et al., 2012b] Loshchilov, I., Schoenauer, M., and Sebag, M. (2012b). Self-adaptive surrogate-assisted covariance matrix adaptation evolution strategy. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 321–328. ACM.

[Loshchilov et al., 2013a] Loshchilov, I., Schoenauer, M., and Sebag, M. (2013a). Bi-population CMA-ES agorithms with surrogate models and line searches. In *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, pages 1177–1184. ACM.

[Loshchilov et al., 2013b] Loshchilov, I., Schoenauer, M., and Sebag, M. (2013b). Intensive surrogate model exploitation in self-adaptive surrogate-assisted cma-es (saacm-es). In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 439–446. ACM.

[Loshchilov et al., 2014] Loshchilov, I., Schoenauer, M., Sebag, M., and Hansen, N. (2014). Maximum likelihood-based online adaptation of hyper-parameters in CMA-ES. In *International Conference on Parallel Problem Solving from Nature*, pages 70–79. Springer.

[Lu and Xiao, 2013] Lu, Z. and Xiao, L. (2013). Randomized block coordinate non-monotone gradient method for a class of nonlinear programming. *arXiv preprint arXiv:1306.5918*.

[Luke and Panait, 2006] Luke, S. and Panait, L. (2006). A comparison of bloat control methods for genetic programming. *Evolutionary Computation*, 14(3):309–344.

[Mahdavi et al., 2015] Mahdavi, S., Shiri, M. E., and Rahnamayan, S. (2015). Metaheuristics in large-scale global continues optimization: a survey. *Information Sciences*, 295:407–428.

[Mei et al., 2016] Mei, Y., Omidvar, M. N., Li, X., and Yao, X. (2016). A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization. *ACM Transactions on Mathematical Software (TOMS)*, 42(2):13.

[Metropolis et al., 1953] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092.

[Molina et al., 2010] Molina, D., Lozano, M., and Herrera, F. (2010). MA-SW-Chains: Memetic algorithm based on local search chains for large scale continuous global optimization. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE.

[Montes de Oca and Stützle, 2011] Montes de Oca, Marco Aand Aydın, D. and Stützle, T. (2011). An incremental particle swarm for large-scale continuous optimization problems: an example of tuning-in-the-loop (re) design of optimization algorithms. *Soft Computing*, 15(11):2233–2255.

[Montes de Oca et al., 2008] Montes de Oca, M., Van den Enden, K., and Stützle, T. (2008). Incremental particle swarm-guided local search for continuous optimization. *Hybrid Metaheuristics*, pages 72–86.

[Montes de Oca and Stützle, 2008] Montes de Oca, M. A. and Stützle, T. (2008). Towards incremental social learning in optimization and multiagent systems. In *Proceedings of the 10th annual conference companion on Genetic and evolutionary computation*, pages 1939–1944. ACM.

[Moskowitz and Caflisch, 1996] Moskowitz, B. and Caflisch, R. E. (1996). Smoothness and dimension reduction in quasi-monte carlo methods. *Mathematical and Computer Modelling*, 23(8):37–54.

[Mühlenbein et al., 1996] Mühlenbein, H., Bendisch, J., and Voigt, H.-M. (1996). From recombination of genes to the estimation of distributions ii. continuous parameters. In *International Conference on Parallel Problem Solving from Nature*, pages 188–197. Springer.

[Mühlenbein and Paass, 1996] Mühlenbein, H. and Paass, G. (1996). From recombination of genes to the estimation of distributions i. binary parameters. In *International Conference on Parallel Problem Solving from Nature*, pages 178–187. Springer.

[Nesterov, 2012] Nesterov, Y. (2012). Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362.

[Nesterov, 2014] Nesterov, Y. (2014). Subgradient methods for huge-scale optimization problems. *Mathematical Programming*, 146(1-2):275–297.

[Ngiam et al., 2011] Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., Le, Q. V., and Ng, A. Y. (2011). On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 265–272.

[Nocedal, 1980] Nocedal, J. (1980). Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782.

[Omidvar et al., 2010a] Omidvar, M., Li, X., Yang, Z., and Yao, X. (2010a). Cooperative co-evolution for large scale optimization through more frequent random grouping. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE.

[Omidvar et al., 2010b] Omidvar, M., Li, X., and Yao, X. (2010b). Cooperative co-evolution with delta grouping for large scale non-separable function optimization. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, pages 1762–1769.

[Omidvar et al., 2014] Omidvar, M. N., Li, X., Mei, Y., and Yao, X. (2014). Cooperative co-evolution with differential grouping for large scale optimization. *IEEE Transactions on Evolutionary Computation*, 18(3):378–393.

[Omidvar et al., 2015] Omidvar, M. N., Li, X., and Tang, K. (2015). Designing benchmark problems for large-scale continuous optimization. *Information Sciences*, 316:419–436.

[Ostermeier et al., 1994a] Ostermeier, A., Gawelczyk, A., and Hansen, N. (1994a). A derandomized approach to self-adaptation of evolution strategies. *Evolutionary Computation*, 2(4):369–380.

[Ostermeier et al., 1994b] Ostermeier, A., Gawelczyk, A., and Hansen, N. (1994b). Step-size adaptation based on non-local use of selection information. In Davidor, Y. et al., editors, *Parallel Problem Solving from Nature (PPSN III)*, volume 866 of *Lecture Notes in Computer Science*, pages 189–198. Springer Verlag.

[Owen, 2002] Owen, A. B. (2002). Necessity of low effective dimension.

[Parsopoulos, 2009] Parsopoulos, K. (2009). Cooperative micro-differential evolution for high-dimensional problems. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 531–538. ACM.

[Patrascu and Necoara, 2015] Patrascu, A. and Necoara, I. (2015). Efficient random coordinate descent algorithms for large-scale structured nonconvex optimization. *Journal of Global Optimization*, 61(1):19–46.

[Poland and Zell, 2001] Poland, J. and Zell, A. (2001). Main vector adaptation: A CMA variant with linear time and space complexity. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1050–1055. Citeseer.

[Potter and De Jong, 1994] Potter, M. and De Jong, K. (1994). A cooperative coevolutionary approach to function optimization. *Parallel Problem Solving from Nature–PPSN III*, pages 249–257.

[Powell, 2006] Powell, M. J. (2006). The newuoa software for unconstrained optimization without derivatives. In *Large-scale nonlinear optimization*, pages 255–297. Springer.

[Qin et al., 2009] Qin, A. K., Huang, V. L., and Suganthan, P. N. (2009). Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE transactions on Evolutionary Computation*, 13(2):398–417.

[Ratitch and Precup, 2004] Ratitch, B. and Precup, D. (2004). Sparse distributed memories for on-line value-based reinforcement learning. In *European Conference on Machine Learning*, pages 347–358. Springer.

[Rechenberg, 1973] Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologishen Evolution*. Frommann-Holzboog.

[Rechenberg, 1994] Rechenberg, I. (1994). *Evolutionsstrategie '94*. Frommann-Holzboog Verlag.

[Richard, 1957] Richard, B. (1957). Dynamic programming. *Princeton University Press*, 89:92.

[Richtárik and Takáč, 2012] Richtárik, P. and Takáč, M. (2012). Efficient serial and parallel coordinate descent methods for huge-scale truss topology design. In *Operations Research Proceedings 2011*, pages 27–32. Springer.

[Richtárik and Takác, 2013] Richtárik, P. and Takác, M. (2013). Distributed coordinate descent method for learning with big data. *arXiv preprint arXiv:1310.2059*.

[Richtárik and Takáč, 2014] Richtárik, P. and Takáč, M. (2014). Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1-2):1–38.

[Richtárik and Takáč, 2016] Richtárik, P. and Takáč, M. (2016). Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, 156(1-2):433–484.

[Rios and Sahinidis, 2013] Rios, L. M. and Sahinidis, N. V. (2013). Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293.

[Ros and Hansen, 2008] Ros, R. and Hansen, N. (2008). A simple modification in CMA-ES achieving linear time and space complexity. In *Parallel Problem Solving from Nature–PPSN X*, pages 296–305. Springer.

[Rudolph, 1997] Rudolph, G. (1997). *Convergence properties of evolutionary algorithms*. Kovac.

[Salomon, 1996] Salomon, R. (1996). Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. a survey of some theoretical and practical aspects of genetic algorithms. *BioSystems*, 39(3):263–278.

[Schlierkamp-Voosen and Mühlenbein, 1993] Schlierkamp-Voosen, D. and Mühlenbein, H. (1993). Predictive models for the breeder genetic algorithm. *Evolutionary Computation*, 1(1):25–49.

[Schwefel, 1965] Schwefel, H.-P. (1965). Kybernetische evolution als strategie der experimentellen forschung in der strömungstechnik. *Master's thesis, Technical University of Berlin*.

[Schwefel, 1977] Schwefel, H.-P. (1977). *Numerische optimierung von computer-modellen mittels der evolutionsstrategie*, volume 1. Birkhäuser, Basel Switzerland.

[Schwefel, 1981] Schwefel, H.-P. (1981). *Numerical optimization of computer models*. John Wiley & Sons, Inc.

[Schwefel, 1995] Schwefel, H.-P. (1995). *Evolution and Optimum Seeking*. Wiley.

[Shalev-Shwartz et al., 2011] Shalev-Shwartz, S., Singer, Y., Srebro, N., and Cotter, A. (2011). Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30.

[Shanno and Kettler, 1970] Shanno, D. F. and Kettler, P. C. (1970). Optimal conditioning of quasi-newton methods. *Mathematics of Computation*, 24(111):657–664.

[Shi and Eberhart, 1998] Shi, Y. and Eberhart, R. (1998). A modified particle swarm optimizer. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 69–73. IEEE.

[Shi et al., 2005] Shi, Y., Teng, H., and Li, Z. (2005). Cooperative co-evolutionary differential evolution for function optimization. *Advances in natural computation*, pages 428–428.

[Silva and Costa, 2004] Silva, S. and Costa, E. (2004). Dynamic limits for bloat control. In *Genetic and Evolutionary Computation Conference*, pages 666–677. Springer.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

[Solis and Wets, 1981] Solis, F. J. and Wets, R. J.-B. (1981). Minimization by random search techniques. *Mathematics of operations research*, 6(1):19–30.

[Sonoda et al., 2003] Sonoda, T., Yamaguchi, Y., Arima, T., Olhofer, M., Sendhoff, B., and Schreiber, H.-A. (2003). Advanced high turning compressor airfoils for low reynolds number condition: Part 1—design and optimization. In *ASME Turbo Expo 2003, collocated with the 2003 International Joint Power Generation Conference*, pages 437–450. American Society of Mechanical Engineers.

[Soule, 2002] Soule, T. (2002). Exons and code growth in genetic programming. In *European Conference on Genetic Programming*, pages 142–151. Springer.

[St-Pierre et al., 2011] St-Pierre, D. L., Louveaux, Q., and Teytaud, O. (2011). Online sparse bandit for card games. In *Advances in Computer Games*, pages 295–305. Springer.

[Steihaug, 1983] Steihaug, T. (1983). The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637.

[Storn and Price, 1995] Storn, R. and Price, K. (1995). *Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces*, volume 3. ICSI Berkeley.

[Storn and Price, 1997] Storn, R. and Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359.

[Sun et al., 2015] Sun, Y., Kirley, M., and Halgamuge, S. K. (2015). Extended differential grouping for large scale global optimization with direct and indirect variable interactions. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 313–320. ACM.

[Surkov, 2004] Surkov, V. (2004). *Valuation of Mortgage–Backed Securities in a Distributed Environment*. PhD thesis, University of Toronto.

[Sutton, 1996] Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, pages 1038–1044.

[Tang et al., 2009] Tang, K., Xiaodong, L., Suganthan, P. N., Yang, Z., and Wiese, T. (2009). Benchmark functions for the cec'2010 special session and competition on large-scale global optimization. Technical report, Nature Inspired Computation and Applications Laboratory (NICAL), School of Computer Science and Technology, University of Science and Technology of China, Electric Building No. 2, Room 504, West Campus, Huangshan Road, Hefei 230027, Anhui, China.

[Tang et al., 2007] Tang, K., Yáo, X., Suganthan, P. N., MacNish, C., Chen, Y.-P., Chen, C.-M., and Yang, Z. (2007). Benchmark functions for the cec'2008 special session and competition on large scale global optimization. *Nature Inspired Computation and Applications Laboratory, USTC, China*, pages 153–177.

[Tezuka, 2005] Tezuka, S. (2005). On the necessity of low-effective dimension. *Journal of Complexity*, 21(5):710–721.

[Tusar et al., 2016] Tusar, T., Brockhoff, D., Hansen, N., and Auger, A. (2016). Coco: The bi-objective black box optimization benchmarking (bbob-biobj) test suite. *arXiv preprint arXiv:1604.00359*.

[Van den Bergh and Engelbrecht, 2004] Van den Bergh, F. and Engelbrecht, A. (2004). A cooperative approach to particle swarm optimization. *Evolutionary Computation, IEEE Transactions on*, 8(3):225–239.

[Vicini and Quagliarella, 1999] Vicini, A. and Quagliarella, D. (1999). Airfoil and wing design through hybrid optimization strategies. *AIAA journal*, 37(5):634–641.

[Wang and Fang, 2003] Wang, X. and Fang, K.-T. (2003). The effective dimension and quasi-monte carlo integration. *Journal of Complexity*, 19(2):101–124.

[Wang and Sloan, 2005] Wang, X. and Sloan, I. H. (2005). Why are high-dimensional finance problems often of low effective dimension? *SIAM Journal on Scientific Computing*, 27(1):159–183.

[Wang and Li, 2009] Wang, Y. and Li, B. (2009). A self-adaptive mixed distribution based uni-variate estimation of distribution algorithm for large scale global optimization. In *Nature-Inspired Algorithms for Optimisation*, pages 171–198. Springer.

[Wang et al., 2010] Wang, Y., Li, B., and Weise, T. (2010). Estimation of distribution and differential evolution cooperation for large scale economic load dispatch optimization of power systems. *Information Sciences*, 180(12):2405–2420.

[Wang et al., 1998] Wang, Z., Droegemeier, K., and White, L. (1998). The adjoint newton algorithm for large-scale unconstrained optimization in meteorology applications. *Computational Optimization and Applications*, 10(3):283–320.

[Wang et al., 2013] Wang, Z., Zoghi, M., Hutter, F., Matheson, D., and de Freitas, N. (2013). Bayesian optimization in a billion dimensions via random embeddings. *arXiv preprint arXiv:1301.1942*.

[Weicker and Weicker, 1999] Weicker, K. and Weicker, N. (1999). On the improvement of coevolutionary optimizers by learning variable interdependencies. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3. IEEE.

[Whitley et al., 1989] Whitley, L. D. et al. (1989). The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *ICGA*, volume 89, pages 116–123.

[Wierstra et al., 2008] Wierstra, D., Schaul, T., Peters, J., and Schmidhuber, J. (2008). Natural evolution strategies. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 3381–3387. IEEE.

[Yang et al., 2007] Yang, Z., Tang, K., and Yao, X. (2007). Differential evolution for high-dimensional function optimization. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 3523–3530. Ieee.

[Yang et al., 2008a] Yang, Z., Tang, K., and Yao, X. (2008a). Large scale evolutionary optimization using cooperative coevolution. *Information Sciences*, 178(15):2985–2999.

[Yang et al., 2008b] Yang, Z., Tang, K., and Yao, X. (2008b). Multilevel cooperative coevolution for large scale optimization. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 1663–1670. IEEE.

[Yang et al., 2008c] Yang, Z., Tang, K., and Yao, X. (2008c). Self-adaptive differential evolution with neighborhood search. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 1110–1116. IEEE.

[Yao et al., 1999] Yao, X., Liu, Y., and Lin, G. (1999). Evolutionary programming made faster. *Evolutionary Computation, IEEE Transactions on*, 3(2):82–102.

[Yuan et al., 2012] Yuan, G.-X., Ho, C.-H., and Lin, C.-J. (2012). Recent advances of large-scale linear classification. *Proceedings of the IEEE*, 100(9):2584–2603.

[Yuan, 2010] Yuan, Y. (2010). Gradient methods for large scale convex quadratic functions. In *Optimization and regularization for computational inverse problems and applications*, pages 141–155. Springer.

[Zhang et al., 1997] Zhang, B.-T., Ohm, P., and Mühlenbein, H. (1997). Evolutionary induction of sparse neural trees. *Evolutionary Computation*, 5(2):213–236.

[Zhang and Sanderson, 2009] Zhang, J. and Sanderson, A. (2009). Jade: adaptive differential evolution with optional external archive. *Evolutionary Computation, IEEE Transactions on*, 13(5):945–958.

[Zhang, 2004] Zhang, T. (2004). Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, page 116. ACM.

[Zhao et al., 2008] Zhao, S., Liang, J., Suganthan, P., and Tasgetiren, M. (2008). Dynamic multi-swarm particle swarm optimizer with local search for large scale global

optimization. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 3845–3852. IEEE.

[Zhong et al., 2004] Zhong, W., Liu, J., Xue, M., and Jiao, L. (2004). A multiagent genetic algorithm for global numerical optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(2):1128–1141.

Appendix A

Appendix

A.1 Benchmarking Large-Scale Algorithms

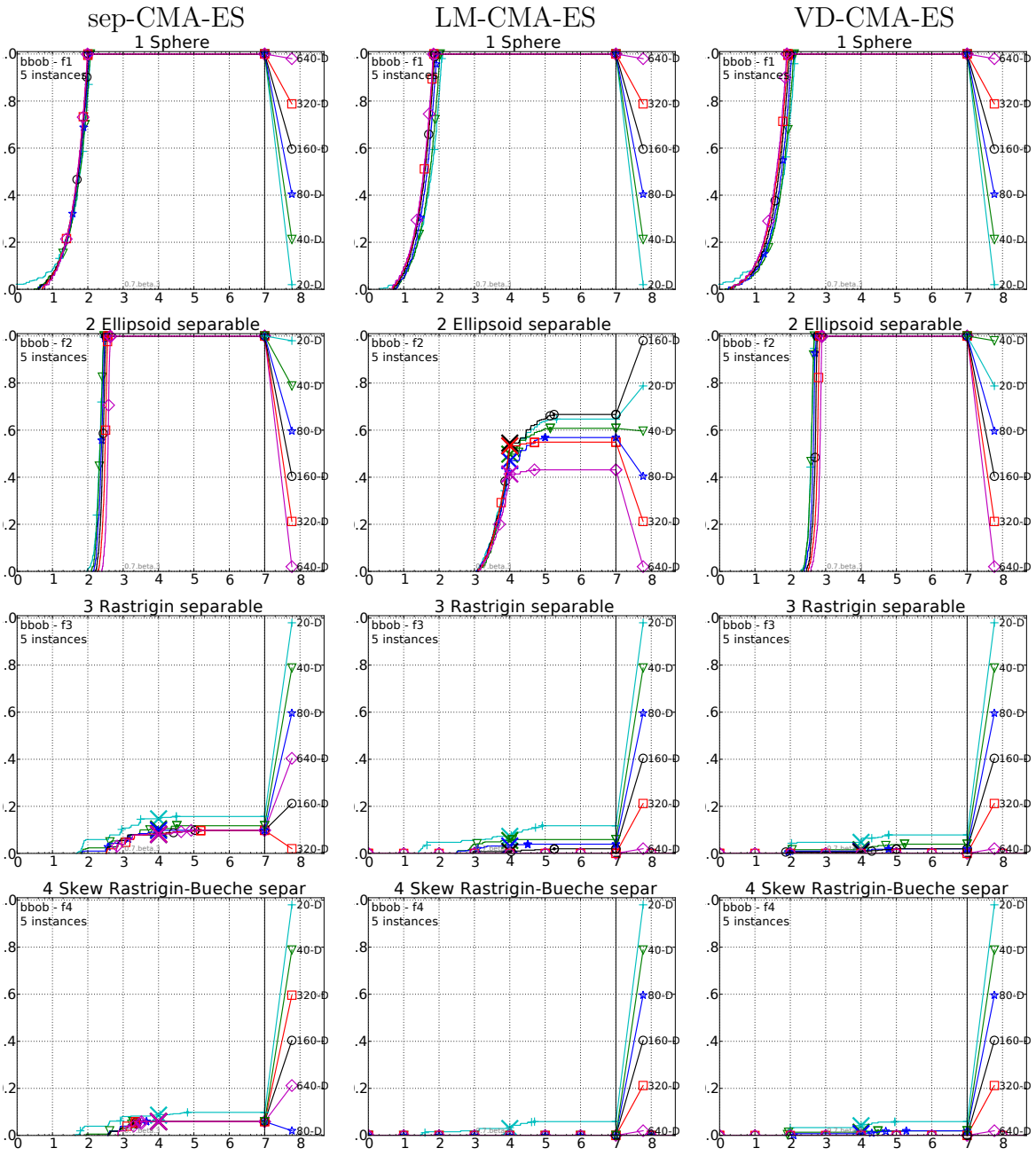
sep-CMA-ES
LM-CMA-ES
VD-CMA-ES
1 Sphere
2 Ellipsoid separable
3 Rastrigin separable
4 Skew Rastrigin-Bueche separ
bbob - f1
bbob - f2
bbob - f3
bbob - f4
5 instances
20-D
40-D
80-D
160-D
320-D
640-D

Figure A.1

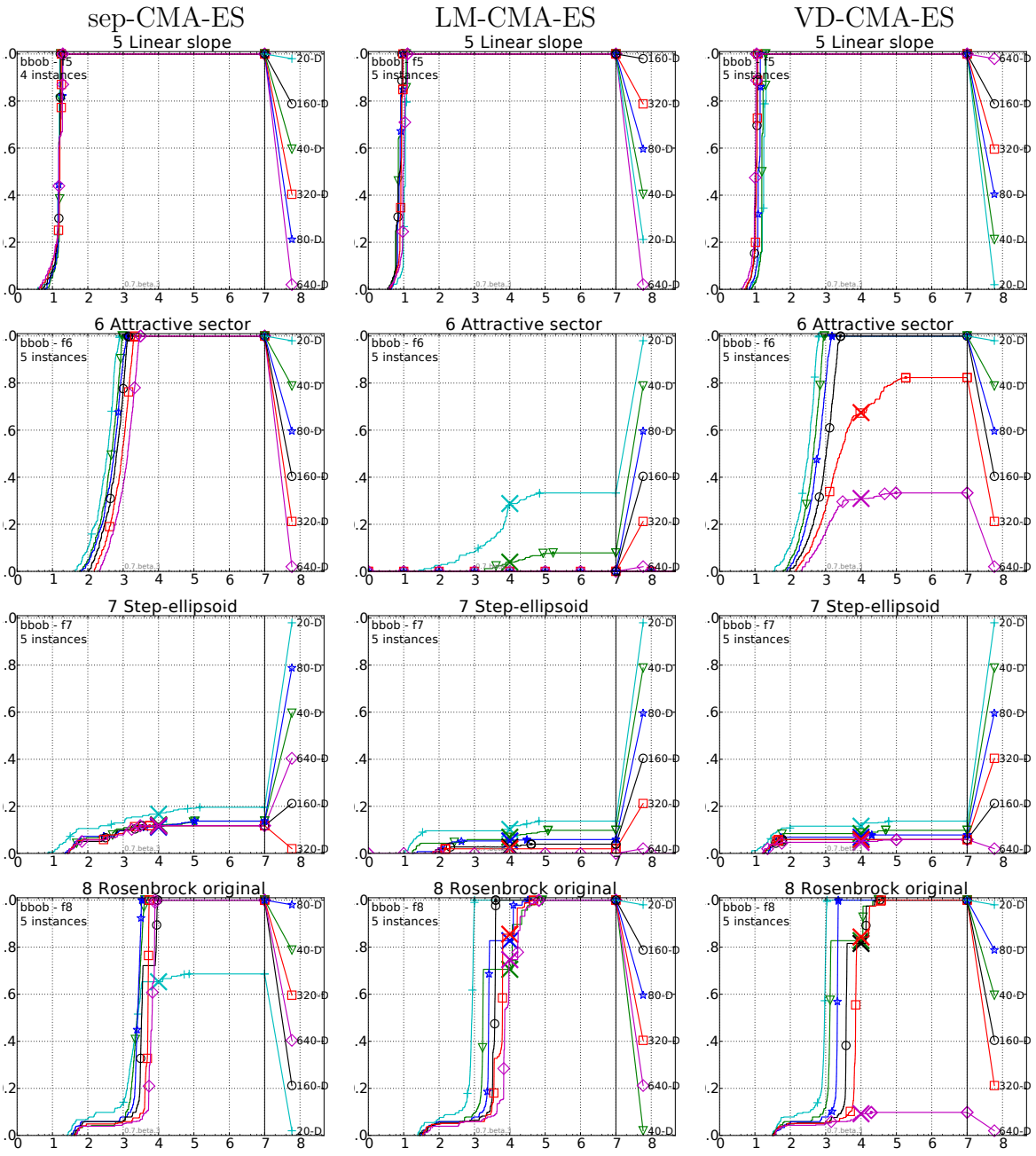
sep-CMA-ES
LM-CMA-ES
VD-CMA-ES
5 Linear slope
bbob - f5
4 instances
5 instances
6 Attractive sector
bbob - f6
5 instances
7 Step-ellipsoid
bbob - f7
5 instances
8 Rosenbrock original
bbob - f8
5 instances
20-D
40-D
80-D
160-D
320-D
640-D

Figure A.2

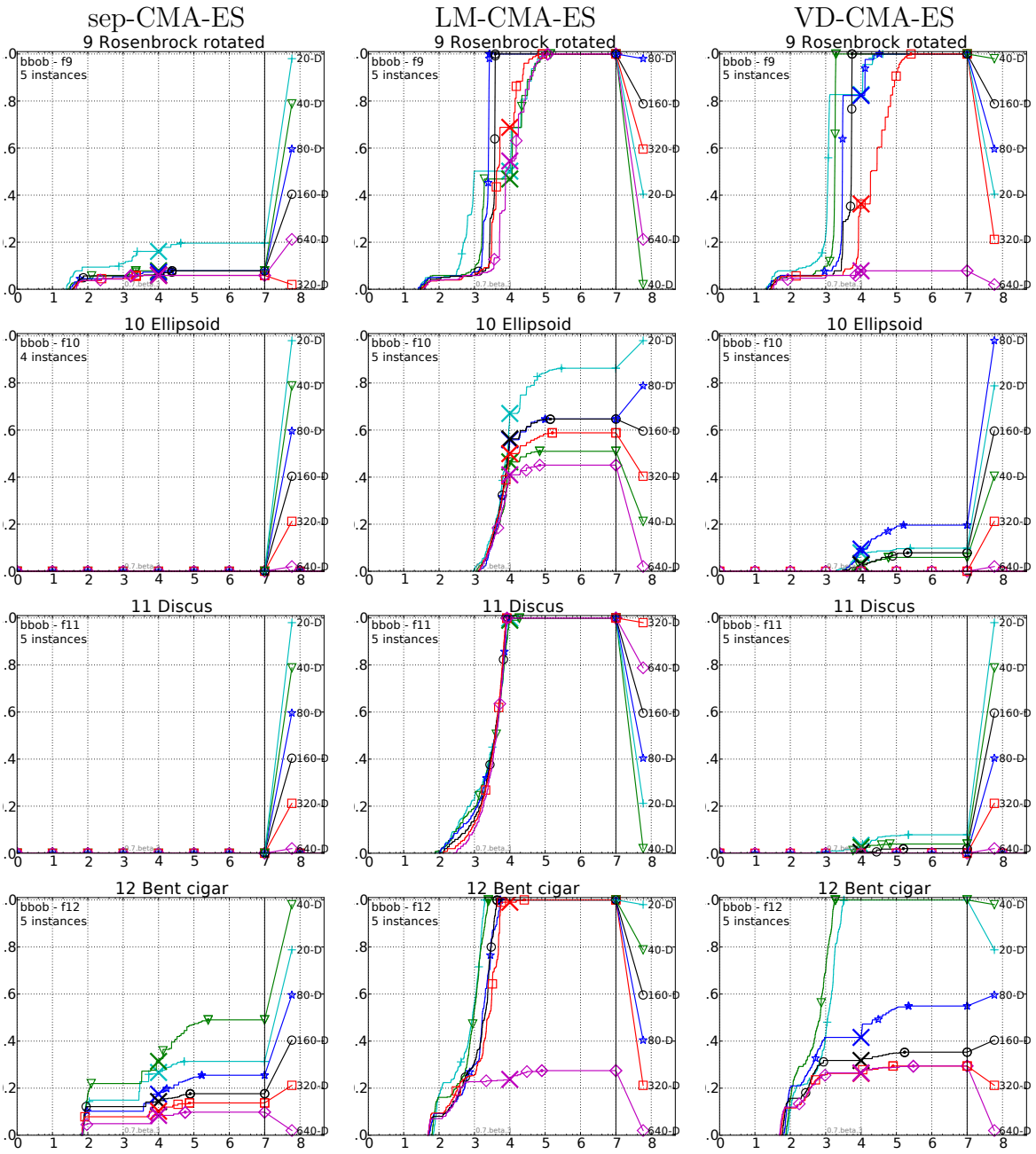
sep-CMA-ES
LM-CMA-ES
VD-CMA-ES
9 Rosenbrock rotated
10 Ellipsoid
11 Discus
12 Bent cigar
bbob - f9
5 instances
bbob - f10
4 instances
5 instances
bbob - f11
5 instances
bbob - f12
5 instances
20-D
40-D
80-D
160-D
320-D
640-D

Figure A.3

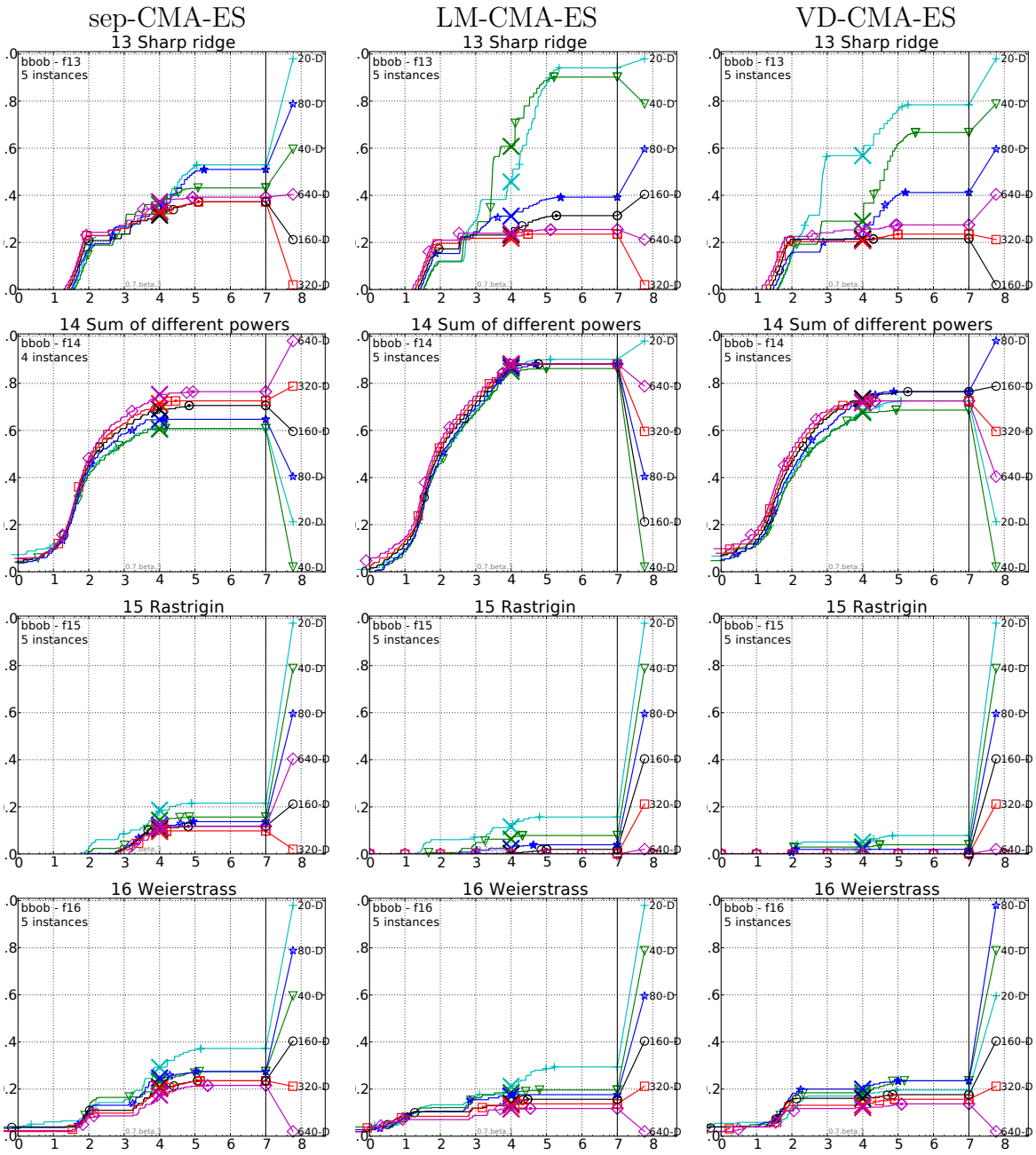
sep-CMA-ES
LM-CMA-ES
VD-CMA-ES
13 Sharp ridge
bbob - f13
5 instances
14 Sum of different powers
bbob - f14
4 instances
5 instances
15 Rastrigin
bbob - f15
16 Weierstrass
bbob - f16
20-D
40-D
80-D
160-D
320-D
640-D

Figure A.4

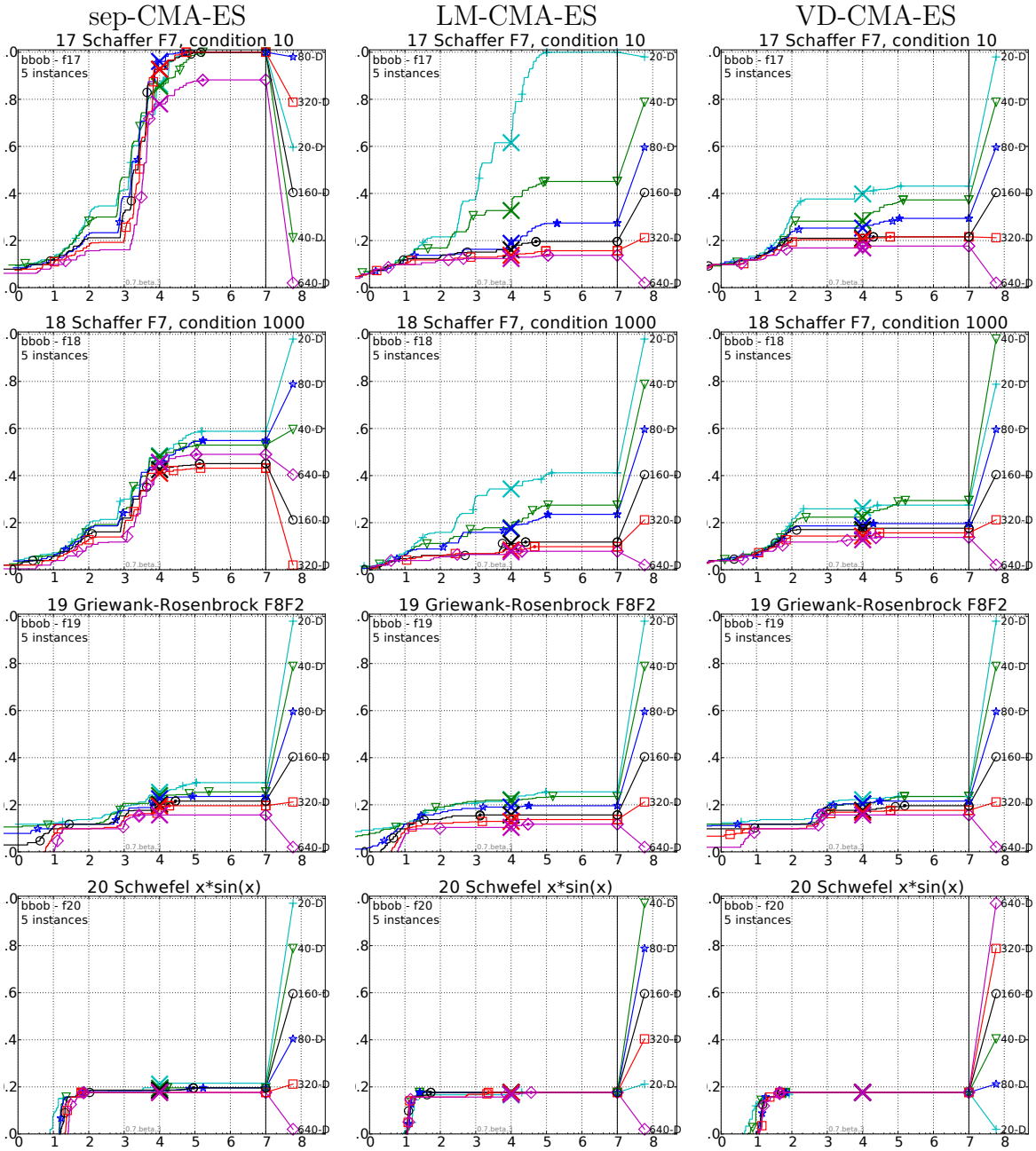
sep-CMA-ES
LM-CMA-ES
VD-CMA-ES
17 Schaffer F7, condition 10
18 Schaffer F7, condition 1000
19 Griewank-Rosenbrock F8F2
20 Schwefel x*sin(x)
bbob - f17
bbob - f18
bbob - f19
bbob - f20
5 instances
20-D
40-D
80-D
160-D
320-D
640-D

Figure A.5

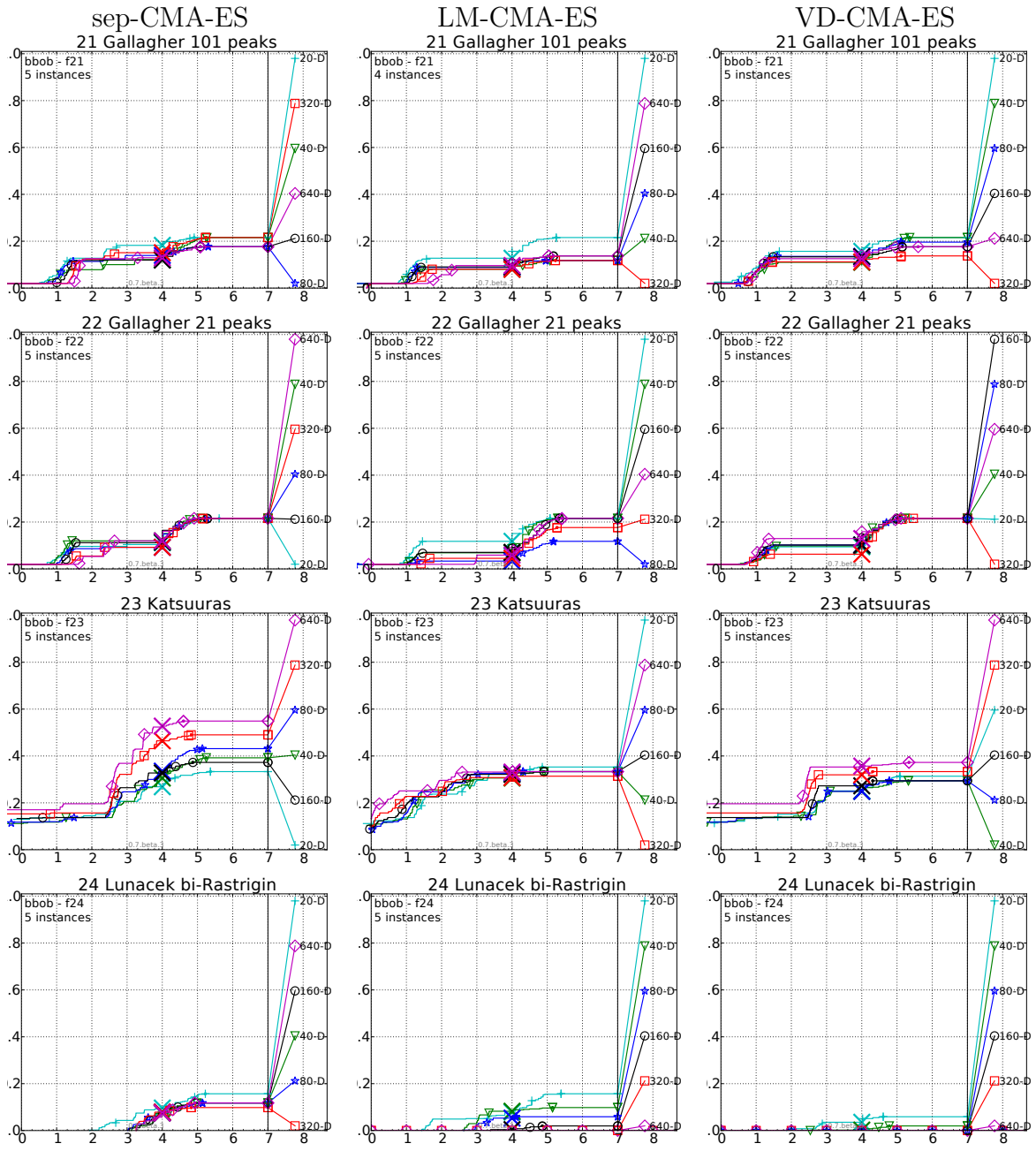
sep-CMA-ES
LM-CMA-ES
VD-CMA-ES
21 Gallagher 101 peaks
bbob - f21
5 instances
4 instances
22 Gallagher 21 peaks
bbob - f22
23 Katsuuras
bbob - f23
24 Lunacek bi-Rastrigin
bbob - f24
20-D
40-D
80-D
160-D
320-D
640-D

Figure A.6

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Optimisation stochastique de problèmes en boîtes noires et benchmarking en grandes dimensions.

Mots clés : optimisation continue, benchmarking, stratégies d'évolution, grandes dimensions

Résumé : Etant donné le coût élevé qui accompagne, en général, la résolution de problèmes en grandes dimensions, notamment quand il s'agit de problèmes réels ; le recours à des fonctions dites benchmarks et une approche communément utilisée pour l'évaluation d'algorithmes avec un coût minime.
Il est alors question de savoir identifier les formes par lesquelles ces problèmes se présentent pour pouvoir les reproduire dans ces benchmarks.
Une question dont la réponse est difficile vu la variété de ces problèmes, leur complexité, et la difficulté de tous les décrire pertinemment.
L'idée est alors d'examiner les difficultés qui accompagnent généralement ces problèmes, ceci afin de les reproduire dans les fonctions benchmarks et évaluer la capacité des algorithmes à les résoudre.
Dans le cas des problèmes de grandes dimensions, il serait pratique de pouvoir simplement étendre les benchmarks déjà utilisés pour les dimensions moins importantes.
Cependant, il est important de prendre en compte les contraintes additionnelles qui accompagnent les problèmes de grandes dimensions, notamment ceux liés à la complexité d'évaluer ces fonctions benchmark.
Idéalement, les fonctions benchmark en grandes dimensions garderaient la majorité des propriétés de leurs contreparties en dimensions réduites tout en ayant un coût raisonnable.
Les problèmes benchmark sont souvent classifiés en catégories suivant les difficultés qu'ils présentent.
Même dans un scénario en boîte-noire où ce genre d'information n'est pas partagé avec l'algorithme, il reste important et pertinent d'avoir cette classification.
Ceci permet d'identifier les lacunes d'un algorithme vis-à-vis d'une difficulté en particulier, et donc de plus facilement pouvoir l'améliorer.
Une autre question importante à se poser en modélisant des problèmes de grandes dimensions est la pertinence des variables.
En effet, quand la dimension est relativement petite, il n'est pas rare de voir toutes les variables contribuer à définir la qualité d'une solution.
Cependant, quand la dimension grandit, il arrive souvent que des variables deviennent redondantes voire inutiles ; notamment vu la difficulté de trouver une représentation minimaliste du problème.
Ce dernier point encourage la conception et d'algorithmes et de fonctions benchmark traitant cette classe de problèmes.
Dans cette thèse, on réponde, principalement, à trois questions rencontrées dans l'optimisation stochastique continue en grandes dimensions :
1. Comment concevoir une méthode d'adaptation du pas d'une stratégie d'évolution qui, à la fois, est efficace et a un coût en calculs raisonnable
2. Comment construire et généraliser des fonctions à faible dimension effective ?
3. Comment étendre un ensemble de fonctions benchmarks pour des cas de grandes dimensions en préservant leurs propriétés sans avoir des caractéristiques qui soient exploitables

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Title: Stochastic Black-Box Optimization and Benchmarking in Large Dimensions

Keywords: Continuous Optimization, Benchmarking, Evolution Strategies, Large Scale

Abstract: Because of the generally high computational costs that come with large-scale problems, more so on real world problems, the use of benchmarks is a common practice in algorithm design, algorithm tuning or algorithm choice/evaluation. The question is then the forms in which these real-world problems come. Answering this question is generally hard due to the variety of these problems and the tediousness of describing each of them. Instead, one can investigate the commonly encountered difficulties when solving continuous optimization problems. Once the difficulties identified, one can construct relevant benchmark functions that reproduce these difficulties and allow assessing the ability of algorithms to solve them.

In the case of large-scale benchmarking, it would be natural and convenient to build on the work that was already done on smaller dimensions, and be able to extend it to larger ones. When doing so, we must take into account the added constraints that come with a large-scale scenario. We need to be able to reproduce, as much as possible, the effects and properties of any part of the benchmark that needs to be replaced or adapted for large-scales. This is done in order for the new benchmarks to remain relevant.

It is common to classify the problems, and thus the benchmarks, according to the difficulties they present and properties they possess. It is true that in a black-box scenario, such information (difficulties, properties...) is supposed unknown to the algorithm. However, in a benchmarking setting, this classification becomes important and allows to better identify and understand the shortcomings of a method, and thus make it easier to improve it or alternatively to switch to a more efficient one (one needs to make sure the algorithms are exploiting this knowledge when solving the problems). Thus the importance of identifying the difficulties and properties of the problems of a benchmarking suite and, in our case, preserving them.

One other question that rises particularly when dealing with large-scale problems is the relevance of the decision variables. In a small dimension problem, it is common to have all variable contribute a fair amount to the fitness value of the solution or, at least, to be in a scenario where all variables need to be optimized in order to reach high quality solutions. This is however not always the case in large-scales; with the increasing number of variables, some of them become redundant or groups of variables can be replaced with smaller groups since it is then increasingly difficult to find a minimalistic representation of a problem. This minimalistic representation is sometimes not even desired, for example when it makes the resulting problem more complex and the trade-off with the increase in number of variables is not favorable, or larger numbers of variables and different representations of the same features within a same problem allow a better exploration.

This encourages the design of both algorithms and benchmarks for this class of problems, especially if such algorithms can take advantage of the low effective dimensionality of the problems, or, in a complete black-box scenario, cost little to test for it (low effective dimension) and optimize assuming a small effective dimension.

In this thesis, we address three questions that generally arise in stochastic continuous black-box optimization and benchmarking in high dimensions:

1. How to design cheap and yet efficient step-size adaptation mechanism for evolution strategies?
2. How to construct and generalize low effective dimension problems?
3. How to extend a low/medium dimension benchmark to large dimensions while remaining computationally reasonable, non-trivial and preserving the properties of the original problem?

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France