HAL
open science

Studies on stochastic optimisation and applications to the real world

Vincent Berthier

To cite this version:

Vincent Berthier. Studies on stochastic optimisation and applications to the real world. Numerical Analysis [cs.NA]. Université Paris 11, 2017. English. NNT : . tel-01668371

HAL Id: tel-01668371

https://inria.hal.science/tel-01668371v1

Submitted on 20 Dec 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

NNT : 2017SACLS336

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 communcation
Spécialité de doctorat: Informatique

par

M. VINCENT BERTHIER

Studies on stochastic optimisation and applications to the real world

Thèse présentée et soutenue à Orsay, le 29 septembre 2017.
Composition du Jury :

Mme.	ANNE VILNAT	Professeur Université Paris-Sud	(Présidente)
M.	MARCUS GALLAGHER	Docteur University of Queensland	(Rapporteur)
M.	CHRISTIAN GAGNÉ	Docteur Université de Laval	(Rapporteur)
M	GÜNTER RUDOLPH	Professeur Technische Universität Dortmund	(Examinateur)
M.	FRÉDÉRIC SAUBION	Professeur Université d'Angers	(Examinateur)
M.	OLIVIER TEYTAUD	Chargé de Recherche Université Paris-Saclay	(Directeur de thèse)
M.	MARC SCHOENAEUR	Directeur de Recherche Université Paris-Saclay	(Co-directeur de thèse)

Contents

1 Résumé **11**
1.1 De l'impact des variables inutiles sur l'optimisation 11
1.2 Étude de performances sur des problèmes mal conditionnés en grande dimension 12
1.3 Impact de l'utilisation des mutations Quasi-Aléatoire sur les performances de CMA-ES 13
1.4 Introduction de la "Sieves Method" dans le déroulement de l'optimisation 15
1.5 Application sur un problème de gestion de stocks 16
1.6 Combinaison de politiques de contrôles 17
1.7 Application de la "Sieves Method" sur un problème réel 18
1.8 Reproduire un résultat de l'évolution biologique avec des optimiseurs stochastiques 19
1.9 Conclusion 20

2 Introduction **22**
2.1 Generalities 22
2.1.1 Families of optimizers 22
2.1.2 Randomization 26
2.1.3 Invariances 29
2.1.4 Benchmarks 29
2.2 Outline & Contributions 30
2.2.1 Methods 30
2.2.2 Applications 30

Part I Methods **32**

3 On the codimension of the set of optima: large scale optimization with few relevant variables **33**
3.1 Introduction . . . 33
3.2 Theoretical analysis: impact of the codimension on the required number of function evaluations . . . 36
3.2.1 Lower bound . . . 36
3.2.2 Upper bound . . . 37
3.3 Algorithms & their invariances . . . 39
3.3.1 Old and new invariances . . . 40
3.3.2 Algorithms used in our experiments, and their invariances . . . 40
3.4 Experiments . . . 41
3.5 Conclusion . . . 50

4 Large scale ill conditioned functions: when criteria change the whole picture **52**
4.1 Introduction: optimization for artificial intelligence applications . . . 52
4.2 Experiments . . . 54
4.2.1 Test functions . . . 54
4.2.2 Rotations . . . 54
4.2.3 Invariances . . . 57
4.2.4 Experimental results . . . 58
4.3 Conclusion . . . 69

5 Experiments on the CEC 2015 expensive optimization testbed **71**
5.1 State of the Art . . . 71
5.2 Restart / Portfolio . . . 72
5.3 Experiments . . . 72
5.3.1 Preliminary experiments: comparing various algorithms . . . 75
5.3.2 Tuning, and adding quasi-random numbers . . . 75
5.4 CEC 2015 criteria . . . 80
5.5 Other results . . . 80
5.6 Conclusion . . . 82

6 Sieves method in fuzzy control: logarithmically increase the number of rules **86**
6.1 Introduction . . . 86
6.1.1 Sieves method (SM) . . . 87

6.1.2 Sieves method for evolutionary algorithms 87
6.2 Artificial experiments 88
6.3 Applications to fuzzy control 94
6.3.1 Direct Policy Search and noisy optimization 94
6.3.2 Fuzzy control 94
6.3.3 Objective function 95
6.3.4 Optimization algorithm 96
6.3.5 Preliminary experiments 96
6.3.6 Sieves parametrization 96
6.3.7 Experimental results: Sieves method for fuzzy control 97
6.4 Conclusions and further work 102

Part II Applications 104

7 Comparing optimizers on a unit commitment problem 105
7.1 Introduction 105
7.2 Testbed 105
7.3 Summary of results 107
7.3.1 Overview and results per problem size 107
7.3.2 Per family of controllers and per problem size 109
7.4 Conclusions and further work 109

8 Combining policies: the best of human expertise and neurocontrol 112
8.1 Introduction 112
8.2 Background and notations 113
8.2.1 Methodologies based on value functions 114
8.2.2 Direct Policy Search 114
8.3 Meta-policy search 115
8.3.1 Combining policies 115
8.3.2 Orthogonal policies 116
8.4 Experimental results: combining handcrafted functions and neural networks 116
8.4.1 Test problems: two types of unit commitment 117
8.4.2 Noise free setting 118
8.4.3 Noisy setting 118
8.4.4 Experimental results 118

8.4.5 Experimental results: others . . . 123
8.5 Conclusions and further works . . . 123

9 Progressive Differential Evolution on Clustering Real World Problems 125
9.1 Introduction . . . 125
9.2 Continuous Real-World Representative benchmark . . . 126
9.3 Implementation validation . . . 127
9.4 Differential Evolution . . . 128
9.5 Progressive Differential Evolution . . . 129
9.6 Results . . . 131
9.6.1 DE vs CMA-ES . . . 131
9.6.2 DE vs PDE . . . 132
9.6.3 The cost of PDE . . . 132
9.7 Conclusion . . . 133
9.8 Further work . . . 134

10 Understanding bio-physical structures with Genetic Algorithms 138
10.1 Introduction . . . 138
10.2 Structural colours . . . 139
10.3 Algorithms . . . 140
10.4 Results . . . 141
10.4.1 Bragg Mirrors . . . 141
10.4.2 Chirped dielectric mirrors . . . 144
10.4.3 Reproducing the structures found on the *Morpho* family . . . 148
10.4.4 QR on CMA-ES . . . 154
10.5 Conclusion & further work . . . 155

Part III Summary, discussions and perspectives 156

11 Discussions 157
11.1 Are those results unfair? . . . 157
11.1.1 Time budgets . . . 157
11.1.2 Default parameters . . . 158
11.1.3 CMA's diagonal variant . . . 158
11.2 Methods: Summary . . . 159

11.2.1 New invariances: beyond the dichotomy "full separability" and "invariance per rotation" 159
11.2.2 Portfolio methods in continuous optimization 159
11.2.3 Quasi Random: a generic improvement of randomized search heuristics 160
11.2.4 Sieves methods in optimization: a generic improvement of high-dimensional noisy optimization 160
11.3 Applications: modularity, real world, and beyond rotationally invariant artificial testbeds 160

12 Perspectives **163**
12.0.1 Alleviate CMA-ES dimensionality problem 163
12.0.2 Progressive Widening 164

Bibliography **165**

List of Figures

2.1 Comparison between random and quasi-random 28

3.1 Expected fitness value w.r.t computation time (functions 1 to 6) with 100 useless variables 42
3.2 Expected fitness value w.r.t computation time (functions 7 to 12) with 100 useless variables 43
3.3 Expected fitness value w.r.t computation time (functions 13 to 18) with 100 useless variables 44
3.4 Expected fitness value w.r.t computation time (functions 19 to 24) with 100 useless variables 45
3.5 Expected fitness value w.r.t computation time (functions 1 to 6) with 10000 useless variables 46
3.6 Expected fitness value w.r.t computation time (functions 7 to 12) with 10000 useless variables 47
3.7 Expected fitness value w.r.t computation time (functions 13 to 18) with 10000 useless variables 48
3.8 Expected fitness value w.r.t computation time (functions 19 to 24) with 10000 useless variables 49

4.1 Ellipsoid function (not rotated) in dimension 100 and slightly rotated ellipsoid function in dimension 100. 53
4.2 Slightly rotated ellipsoid function (a) 55
4.3 Slightly rotated ellipsoid function (a) 56
4.4 Fully rotated ellipsoid function 60
4.5 Schwefel function 61
4.6 Schwefel function 62
4.7 Cigar function, dimension 1000 (a) 63
4.8 Cigar function, dimension 1000 (b) 64
4.9 Cigar function, dimension 10000 (a) 65
4.10 Cigar function, dimension 10000 (b) 66

4.11 Sparsely rotated ellipsoid function (a) 67
4.11 Sparsely rotated ellipsoid function (b) 68

5.1 Optimizers results on the expensive benchmark in dimension 10 73
5.2 Optimizers results on the expensive benchmark in dimension 30 74
5.3 Portfolio results on the expensive benchmark in dimension 10 76
5.4 Portfolio results on the expensive benchmark in dimension 10 77
5.5 Two CMA variants results on the expensive benchmark in dimension 10 . 78
5.6 Two CMA variants results on the expensive benchmark in dimension 30 . 79

6.1 Impact of the k parameter on the Sieves method. 91
6.2 Logarithmic Sieves Method applied to small test-case (first membership function) 98
6.3 Results with Eq. 6.1, large test case, logarithmic Sieves. "NO" indicates the results with no Sieves Method, while the numbers on the other curves indicates the rates used. We compare our Sieves method to the standard optimization with 32 optimized rules from the start. In this case, we can see a very noticeable increase of performances when using the Sieves Method compared to the vanilla process. 99

8.1 Hydro-electric valley in the noise free case 119
8.2 Hydro-electric network in the noise free case 120
8.3 Hydro-electric valley in the noisy setting 121
8.4 Hydro-electric valley in the noisy setting 122

9.1 Clusters position on failure cases, Ruspini dataset with $k = 3$ 130
9.2 Fitness statistics evolution on the Ruspini dataset with $k = 6$ 134
9.3 Performance as a ratio to the optimum ($\frac{\hat{f}}{f^*}$) 137

10.1 Dielectric Mirror structure 139
10.2 Optimisation results for simple Bragg Mirror 142
10.3 Interleaved layers of a Bragg Mirror 143
10.4 Optimisation results on a Chirped Mirror 146
10.5 Interleaved layers of a Chirped Mirror 147
10.6 Scales found on the wings of butterflies in the *Morpho* family 150
10.7 Pattern of the scales of butterflies and its computed reflection 151
10.8 Refraction spectrum obtained for a *Morpho*-like structure. 153

List of Tables

5.1 Errors obtained by CMASPHI-QR in 10D 80
5.2 Errors obtained by CMASPHI-QR in 30D 81
5.4 Results of different variants of CMA-ES 83
5.5 Results of different variants of Nelder-Mead 84
5.6 Results of different variants of Differential Evolution 84
5.7 Results of different variants of PSO 85

6.1 Sieves Method for DE 96
6.2 Sieves Method with Linear increase on the Big case, with the membership function 6.1 97
6.3 Sieves Method with Linear increase on the Small case, with the membership function 6.4 98
6.4 Sieves Method with Linear increase on the Big case, with the membership function 6.4 99
6.5 Sieves Method with Linear increase on the Small case, with the membership function 6.4 100
6.6 Sieves Method with Linear increase on the Big case, with the membership function 6.1 100
6.7 Sieves Method with Linear increase on the Small case, with the membership function 6.4 100
6.8 Sieves Method with Linear increase on the Big case, with the membership function 6.4 101
6.9 Sieves Method with Linear increase on the Small case, with the membership function 6.4 101

7.1 Summary of optimizers performances 108
7.2 Optimizers direct comparisons (5 stocks, 25 time-steps) 110
7.3 Optimizers direct comparisons (15 stocks, 50 time-steps) 111

9.1 Average fitness results and SP1 measure for CMA-ES and CMA-ES(50, 100) 127

9.2 Average fitness results and SP1 measure for DE and PDE 131
9.3 Success rate for CMA(50,100), DE and PDE 133

10.1 Optimization results for a simple Bragg Mirror 144
10.2 Chirped Mirror layers' thickness 148
10.3 Optimization results for a Bragg Mirror 149
10.4 Optimization results for a a structure mimicking the scales of butterflies of the *Morpho* family. 154

List of Algorithms

6.1 Sieves Method for (μ,λ)-SAES 88
6.2 Sieves Method for $(\mu/\mu,\lambda)$-SAES 89

Chapitre 1

Résumé

1.1 De l'impact des variables inutiles sur l'optimisation

Dans la plupart des problèmes étudiés, le poids de chaque variable diffère. C'est notamment le cas sur la fonction ellipsoïde $f(\mathbf{x}) = 10^6 x_1^2 + \sum_{i=2}^{D} x_i^2$ où la première variable x_1 a un poids un million de fois plus important que les autres sur l'évaluation finale. Dans certains cas en revanche, il y a des variables qui n'interviennent absolument pas dans le résultat final. En dépit de ceci, on souhaite évidemment pouvoir procéder à l'optimisation. Pour ce faire, il convient de choisir attentivement l'algorithme utilisé.

Une étude théorique des taux de convergence à l'optimum étendant les résultats généraux pour la limite basse [Fournier and Teytaud, 2011] et la limite haute [Jamieson et al., 2012] montre que le nombre de variables inutiles n'a pas d'impact sur le taux de convergence des algorithmes.

Pourtant, les variables inutiles ont un impact clair dès l'initialisation des algorithmes, et ce à deux niveaux. Le premier concerne les algorithmes pour lesquels la taille de la population dépend de la dimension. Ainsi par exemple, la population nécessaire au fonctionnement de Newuoa [Powell, 2008] a une taille de $2d+1$, tandis que celle de Nelder-Mead [Nelder and Mead, 1965] est de $d+1$, où d est la dimension du problème considéré. La plupart des algorithmes à stratégie évolutionnaire ont une population dont la taille est indépendante de la dimension, ou, au pire, qui augmente logarithmiquement en fonction de celle-ci. Cependant, certains de ces algorithmes reposent sur l'utilisation d'une matrice de covariance. Stocker ces matrices en mémoire prend beaucoup de place, beaucoup trop lorsque la dimension dépasse une certaine taille : en dimension un million, on stocke alors une matrice carrée d'un million par un million de doubles, ce qui représente seize teraoctets de mémoire vive. Même s'il était possible de stocker ces matrices, ces algorithmes

nécessitent d^2 paramètres, qu'il n'est statistiquement pas possible de déterminer. À l'inverse, des algorithmes plus simples (tels que $(1+1)$-ES) ne souffrent pas de ce problème.

L'impact d'un grand nombre de variables peut également se faire sentir à l'exécution des optimiseurs. En effet, la supposition faite consistant à dire que le temps de calcul nécessaire à l'algorithme pour passer d'une génération à l'autre est négligeable par rapport au temps nécessaire pour évaluer un individu, ne tient plus lorsque le nombre de variables dépasse les dix mille dans le cas des algorithmes qui ont besoin de recalculer un matrice de covariance. En fait, le temps de calcul devient si long qu'il est impossible d'obtenir des résultats corrects dans un temps raisonnable. C'est pourquoi dans ce travail, la comparaison se fait sur la base du temps de calcul global.

Les expérimentations effectuées sur le jeu de test BBOB [Hansen et al., 2010a], avec un nombre de variables inutiles allant de cent à un million montrent qu'un faible nombre de variables inutiles ne change pas démesurément les performances des algorithmes par rapport à un fonctionnement normal. Plus le nombre de variables inutiles augmente en revanche, et plus les performances d'algorithmes tels que CMA-ES [Hansen and Ostermeier, 2003] ou CMSA-ES [Beyer and Sendhoff, 2008] se dégradent, jusqu'à ce qu'ils ne soient tout simplement plus capables de traiter le problème par défaut d'espace mémoire. Des algorithmes tels que Differential Evolution [Storn and Price, 1997], $(1+1)-ES$ ou Self-Adaptive Evolution Strategy [Beyer, 2001] (dans ses versions isotropique ou anisotropique) ne sont pas impactés par le nombre de variables inutiles. En fait, il est même possible de noter que même si les temps de calculs augmentent forcément, ces algorithmes parviennent tout de même à atteindre le même optimum que dans les cas sans variables inutiles.

1.2 Étude de performances sur des problèmes mal conditionnés en grande dimension

Les systèmes complexes tels que les systèmes météorologiques ou les réseaux électriques, sont un domaine faisant fréquemment appel à l'optimisation boîte noire. Les algorithmes à même de traiter des problèmes mal conditionnés ou au mieux partiellement séparables ne devraient *a priori* guère rencontrer de difficultés.

Or, si de nombreux travaux ont été publiés à propos des performances des algorithmes d'optimisation tels que SA-ES [Beyer, 2001], CMA-ES [Hansen and Ostermeier, 2003], Differential Evolution [Storn and Price, 1997], *etc.*, dans la grande majorité de ces travaux, les tests sont effectués sur des problèmes portant sur moins d'une centaine de variables. Les systèmes complexes cités précedemment sortent cependant de ce domaine : il n'est là pas rare de rencontrer des problèmes où plusieurs milliers de variables doivent être

optimisées, sans qu'il soit possible de réduire le nombre de paramètres.

Le travail effectué ici vise à déterminer quelles sont les performances de ces algorithmes dans les cas extrêmes où le nombre de variables atteint le million. En outre, du fait de l'augmentation du nombre de variables, de nouveaux critères de comparaison entre les algorithmes doivent être pris en compte : lorsque les temps de calculs internes à l'optimiseur augmentent, il convient non plus d'exprimer les budgets alloués aux optimiseurs en terme de nombre d'évaluations, mais en temps.

Cette étude est effectuée sur les algorithmes d'optimisation stochastique les plus représentés dans la littérature, à savoir $(1+1)-ES$, SA-ES [Beyer, 2001] (y compris sa version avec matrice de covariance [Rechenberg, 1973b]), CMA-ES [Hansen and Ostermeier, 2003], CMSA-ES [Beyer and Sendhoff, 2008], DE [Storn and Price, 1997], NM [Nelder and Mead, 1965] et PSO [Kennedy and Eberhart, 1995] et porte sur trois fonctions différentes (Ellipsoïde, Cigar et Schwefel), en utilisant trois types de rotation plus ou moins complètes, impliquant divers degrés de non-séparabilité.

Ce que l'on peut retenir des résultats sur ces problèmes, pour des dimensions allant jusqu'à un million, est que DE, PSO et $(1+1)-ES$ obtiennent globalement les meilleures performances. Tout particulièrement dans le cas de DE pour une séparabilité intermédiaire, PSO pour un budget faible. $(1+1)-ES$ est quant à lui le plus stable.

Lorsqu'il est possible d'utiliser des algorithmes utilisant une matrice de covariance, les résultats sont intéressants, mais pour cela la dimension ne doit pas être trop importante sinon les temps de calcul explosent, voire il devient impossible de traiter le problème. CMA-ES est par exemple particulièrement bon pour surmonter les soucis de précision numérique, atteignant des précisions que les autres algorithmes ne peuvent pas atteindre.

L'un des points les plus importants à retenir de ce travail est que l'on voit à nouveau l'illustration du fait qu'un seul et même optimiseur ne peut pas répondre à tous les problèmes de manière optimale : c'est toute la puissance d'un portfolio d'optimiseurs, qui permet de déterminer automatiquement quel est l'optimiseur le plus adapté au problème qui lui est soumis, et de l'utiliser pour atteindre l'optimum de la manière la plus efficace possible.

1.3 Impact de l'utilisation des mutations Quasi-Aléatoire sur les performances de CMA-ES

L'utilisation de processus aléatoires est une composante importante du fonctionnement des algorithmes d'optimisation boîte noire. De fait, elle est si importante que c'est dans certains cas la seule manière de garantir la convergence [Gelly et al., 2007].

Tel qu'illustré par l'image 2.1a, le tirage de nombres purement aléatoires peut conduire à exploiter abondamment une certaine région de l'espace tout en en ignorant une vaste partie. Une manière de limiter ce problème consiste à utiliser des nombres générés quasi-aléatoirement tel que proposé dans [Teytaud and Gelly, 2007]. Ainsi, on peut voir sur l'image 2.1b que l'espace est bien plus uniformément couvert : aucune région n'est plus ou moins exploitée que le reste.

L'utilisation de mutations quasi-aléatoires est le sujet du travail présenté dans cette section, et est appliqué à la compétition "Expensive" de CEC2015 [Chen et al., 2014]. Si, en règle générale sur les benchmarks tels que BBOB, le budget alloué aux algorithmes est de l'ordre de $budget = 10 \times D^2$ (avec D la dimension du problème), on se place ici dans un cas où les évaluations sont beaucoup plus chères : on ne dispose que de $budget = 50 \times D$.

Un optimiseur souvent acclamé, à juste titre, pour sa capacité à atteindre une solution optimale en un faible nombre d'évaluations est CMA-ES [Hansen and Ostermeier, 2003], ce qui sera également le cas dans les problèmes qui nous intéressent ici. Nous allons cependant introduire une variante de CMA-ES, pour laquelle nous utiliserons des mutations quasi-aléatoires.

Un autre moyen de pallier à ce manque de budget, proposé par [Baudis and Posik, 2014] consiste à construire un portfolio, un ensemble de plusieurs optimiseurs, et de leur partager le budget selon des règles plus ou moins complexes [Pulina and Tacchella, 2009, Gagliolo and Schmidhuber, 2005, Gagliolo and Schmidhuber, 2006]. En plus de CMA-ES, les autres optimiseurs utilisés dans les portfolios sont $(1+1)-ES$, SA-ES [Beyer, 2001] (y compris sa version avec matrice de covariance [Rechenberg, 1973b]), CMA-ES [Hansen and Ostermeier, 2003], CMSA-ES [Beyer and Sendhoff, 2008], DE [Storn and Price, 1997], NM [Nelder and Mead, 1965] et PSO [Kennedy and Eberhart, 1995]. Tous les portfolios étaient construits de manière à ce que 25% du budget soit réparti équitablement entre chaque optimiseur, puis sur les 75% restants du budget, seul le meilleur optimiseur était utilisé pour tenter d'atteindre l'optimum.

Le premier enseignement des tests effectués est que globalement, les portfolios "restart" ont été plus performants que les optimiseurs simples. Cela s'explique par le fait qu'en dépit d'un budget plus réduit, les restarts permettent d'éviter les écueils d'une mauvaise initialisation. Bien que CMA-ES fasse preuve de performances tout à fait honorables comme escompté, il est souvent battu par NM, qui semble moins souffrir d'une mauvaise initialisation. Ceci est confirmé par le fait que les portfolio "restart" de CMA-ES obtiennent de meilleures performances que ceux de NM.

La deuxième batterie de tests effectués a consisté à créer une grande quantité de variante de CMA-ES, en utilisant une stratégie élitiste ou non, un pas de mutation plus ou moins grand que ce soit pour sa limite basse ou sa valeur initiale ou une taille de popula-

tion plus ou moins grande. Enfin, chaque variante était dédoublée, l'une utilisant des mutations aléatoires normales, et l'autre des mutations quasi-aléatoires. Il est particulièrement intéressant de noter que selon les critères utilisés pour l'évaluation des résultats sur ce jeu de test donné, à chaque fois, la version quasi-aléatoires l'emporte sur la version normale. En fait, en y regardant de plus près, on se rend compte que bien que les plus mauvais runs des variantes quasi-aléatoires sont plus mauvais que les variantes normales, on remarque sur les moyennes et médianes, que ce sont les versions quasi-aléatoire qui dominent nettement.

1.4 Introduction de la "Sieves Method" dans le déroulement de l'optimisation

Une technique bien connue des statisticiens pour rendre un problème plus facile consiste à commencer par considérer un sous-ensemble de variables, puis petit à petit d'introduire de nouvelles variables. Bien que soutenue par un vaste éventail de travaux aussi bien théoriques qu'empiriques, cette technique est pour ainsi dire inconnue dans le monde de l'optimisation stochastique. Dans ce travail, une adaptation de cette technique à l'optimisation stochastique est proposée et testée sur des problèmes artificiels puis basés sur le monde réel. On l'appellera Élargissement Progressif.

Les algorithmes évolutionnaires fonctionnent en générant des descendants à partir d'un ou plusieurs parents, en les faisant muter ou se croiser selon des règles plus ou moins complexes en fonction des algorithmes. Chaque individu est alors évalué, puis va remplacer ou non la génération précédente. Tandis que dans le cas général l'ensemble des variables sont modifiées d'une génération à l'autre, il suffit, pour appliquer l'Élargissement Progressif de ne modifier que les N premières variables. Ensuite, tout se joue dans la façon dont N est calculé et mis à jour.

Si N est augmenté trop souvent, on se trouve finalement dans une situation où toutes les variables sont optimisées dès le début ou presque. En revanche, si N n'est pas suffisamment augmenté, on se retrouve dans une situation où l'on prend le risque de consommer tout le budget sans avoir touché une seule fois à certaines variables, ce qui n'est pas une solution souhaitable dans le cas général.

Les tests effectués avec SA-ES [Beyer, 2001] sur un problème artificiel montrent que cette méthode peut permettre d'améliorer sensiblement les performances lorsque le problème est mal conditionné. En revanche la paramétrisation de l'évolution de N dépend du problème considéré : il n'existe pas une valeur unique qui permette de faire fonctionner l'Élargissement Progressif dans tous les cas.

Des tests plus poussés sur un problème de recherche de politique floue sur un problème

de gestion de stocks de production électrique en utilisant DE [Storn and Price, 1997] permettent de renforcer cette première impression. Bien que l'amélioration des performances d'un optimiseur utilisant l'Élargissement Progressif se fait souvent de manière asymptotique, cette amélioration est parfois très nette. Il est en outre intéressant de noter qu'une vaste famille de fonctions contrôlant l'évolution de N permet d'obtenir cette amélioration : ainsi que l'on incrémente N linéairement ou logarithmiquement en fonction du temps ou du nombre de génération, les résultats sont meilleurs (de manière plus marquée dans le cas logarithmique toutefois), sans même chercher à configurer les paramètres de contrôle de manière particulièrement fine. Cette technique a toutefois un coût, visible au début du processus d'optimisation : le nombre de variables optimisées étant faible, les performances sont moindres qu'une optimisation normale.

1.5 Application sur un problème de gestion de stocks

Lorsque de nouveaux algorithmes d'optimisation stochastiques sont conçus, ou que des algorithmes existants sont modifiés, ils sont testés sur des jeux de tests connus comme BBOB [Hansen et al., 2010a], CEC [Suganthan et al., 2005], Cute/Cuter/Cutest [Gould et al., 2003], *etc.* Ces jeux de tests sont bien évidemment particulièrement importants pour déterminer les performances et lacunes de chacun des optimiseurs considérés, notamment de par le vaste éventail d'écueils qu'ils contiennent : multimodalité, non séparabilité, *etc.* Pour autant, ces jeux de tests ne sont pas sans défauts, et ce notamment de par le fait que pour l'essentiel, les problèmes qu'ils proposent sont des fonctions mathématiques artificielles sans lien avec le monde réel.

Pourtant, ces algorithmes étant destinés à être utilisés pour répondre à des problèmes plus ou moins complexes du monde réel, il est tout aussi intéressant de les comparer sur des jeux de tests qui en sont inspirés. Ici, la comparaison des principaux algorithmes d'optimisation stochastique de la littérature se fait sur un problème de gestion de stocks de production électrique. Ce problème, complexe, est d'autant plus intéressant que cette complexité est paramétrable : le nombre de stocks, le nombre de pas de temps, les conditions de remplissage et de production sont autant de paramètres qui permettent de rendre un problème plus ou moins complexe. Enfin, l'utilisation de plusieurs familles de politiques de décision (fonction expert, réseau de neurones, logique floue ou planification pas-à-pas) permet de passer de trois à plusieurs milliers de paramètres à optimiser.

Bien que ne remettant pas complètement en cause les résultats des jeux de tests artificiels, les tests effectués ici permettent de dégager quelques enseignements intéressants. Ainsi, on remarque que DE [Storn and Price, 1997] est l'optimiseur le plus stable, offrant le plus souvent les meilleures performances sur ce type de problème, ou que

PSO [Kennedy and Eberhart, 1995] fonctionne particulièrement bien en grande dimension. En revanche, les performances de CMA-ES [Hansen and Ostermeier, 2003] sont globalement décevantes - tout particulièrement en grande dimension - de même que celles de NM [Nelder and Mead, 1965] ou $(1+1)-ES$, à l'exception de quelques cas particuliers.

1.6 Combinaison de politiques de contrôles

La recherche directe de politique de contrôle consiste à déterminer les paramètres d'un ensemble de règles qui permettent de répondre au mieux à un problème, le plus souvent de planification : par exemple la production électrique en fonction d'un certain nombre de contraintes, telles que la météo ou la consommation.

Les règles en question peuvent être issues de l'expertise humaine, ou bien être des méthodes génériques comme un réseau de neurones ou des fonctions de logique floue. Les premières sont particulièrement efficaces lorsqu'elles sont disponibles, mais ont souvent l'inconvénient d'être très limitées par leur structure même. Quant aux dernières, elles sont évidemment beaucoup plus libres, mais cela les rend par là même beaucoup plus difficiles à configurer.

L'idée derrière ce travail est de parvenir à tirer profit de la robustesse de l'une et de la capacité de généralisation de l'autre en les combinant toutes deux pour former une méta-politique. Bien qu'il existe déjà des méthodes tirées de la Programmation Dynamique [Bellman, 1957] pour parvenir à déterminer quelle politique est optimale, ces méthodes ont l'inconvénient d'être coûteuses en temps de calcul mais aussi de fournir une réponse incomplète si pour une raison ou pour une autre le processus d'optimisation venait à être interrompu avant sa conclusion normale.

Dans ce travail, on considère au contraire l'optimisation directe de deux politiques différentes, C_1 et C_2, dont le poids respectif est contrôlé par un paramètre indépendant α. La politique résultante est alors

$$C = \alpha C_1 + (1-\alpha)C2$$

Les tests effectués sur des problèmes de gestion de stocks, en présence ou non de bruit démontrent que lorsque les politiques combinées sont suffisamment différentes l'une de l'autre (*i.e.* sont orthogonales), les résultats obtenus sont au pire aussi bons que la meilleure des deux politiques. Le plus souvent en revanche, les résultats de la combinaison sont nettement supérieurs aux deux politiques individuelles, et ce d'autant plus que le budget augmente. En outre, l'un des gros avantages de cette technique est qu'il est possible d'arrêter le processus d'optimisation à tout moment, et de tout même obtenir une politique raisonnable.

1.7 Application de la “Sieves Method” sur un problème réel

Parmi les rares jeux de tests basés sur le monde réel, on trouve un ensemble de trois problèmes de clustering [Gallagher, 2016], où le but est de parvenir à optimiser la position d’un ensemble de noyaux dont le nombre varie de deux à dix, de manière à minimiser la somme de la distance entre un point et le noyau le plus proche. Ces problèmes sont intéressants à plus d’un titre car, en plus d’être inspirés du monde réel, leur complexité varie en fonction du nombre de noyau et de la dimension de chaque point de donnée et ils sont faciles à comprendre. Un autre point particulièrement important ici, est que les solutions optimales sont connues [du Merle et al., 1999], ce qui rend l’évaluation des performances des optimiseurs possible dans l’absolu, et pas seulement les uns par rapport aux autres.

Dans l’article d’origine, les performances d’un ensemble d’algorithmes sont étudiées : CMA-ES, Nelder-Mead, la recherche aléatoire et la méthode des k-means. Dans ce travail, la première chose qui est faite est de valider l’implémentation utilisée en comparant les performances de CMA-ES. Ce faisant, l’indicateur SP1 [Auger and Hansen, 2005] qui représente le nombre d’évaluations nécessaires pour atteindre l’optimum est calculé, dans le but de permettre une comparaison plus précise de chaque optimiseur.

Dans un second temps, ces problèmes sont soumis à Differential Evolution [Storn and Price, 1997] qui offre des performances nettement supérieures à CMA-ES, bien que ne parvenant pas toujours à atteindre l’optimum. Une étude des raisons de ces échecs montre qu’ils sont le plus souvent dûs à un problème de minima locaux desquels DE ne parvient pas à sortir. Afin de tenter de contourner le problème, est alors introduite une variante de DE nommée Progressive Differential Evolution. Cette variante consiste à introduire l’Élargissement Progressif du nombre de variables optimisées.

Cette modification n’est pas sans coût : jusqu’à 15 000 évaluations - sur un budget de 300’000 - les performances de PDE sont nettement inférieures à celles de DE. Cela s’explique simplement par le fait que jusqu’à ce moment là, toutes les variables n’étaient pas encore optimisées. En revanche, après 15 000 évaluations, les résultats déjà excellents de DE sont encore améliorés, parfois très nettement, grâce à cette technique, que ce soit en terme de fitness moyenne que de taux d’atteinte de l’optimum (à l’exception de cinq cas particuliers, et dans seulement deux cas la différence est importante).

1.8 Reproduire un résultat de l'évolution biologique avec des optimiseurs stochastiques

De nombreux algorithmes d'optimisation stochastiques sont inspirés de l'évolution naturelle, qu'elle soit d'origine sexuée (c'est par exemple le cas pour Differential Evolution) ou non (par exemple la famille SA-ES). D'autres algorithmes sont inspirés d'autres phénomènes biologiques, comme le vol des oiseaux (ou la nage des bancs de poissons) dans le cas de Particule Swarm Optimisation.

Ici, le but va être de tenter de reproduire une structure naturelle, fruit de plusieurs millions d'années d'évolution, que l'on retrouve sur la chitine de scarabées ou les ailes des papillons de la famille *Morpho*. Ces structures physiques appelées Miroirs de Bragg ont la particularité d'être constituées de fines couches de matière transparente, capables de réfléchir une partie du spectre lumineux grâce à un phénomène de physique quantique.

Les tests les plus simples, visant à concevoir un miroir de Bragg capable de réfléchir la lumière à une longueur d'onde donnée (600nm, correspondant à du orange-rouge) se révèlent concluants : c'est finalement relativement facile, dans la mesure où les différents algorithmes parviennent souvent à obtenir la solution optimale. Lorsque l'on complique le problème, en demandant non plus une réflexion à une longueur d'onde donnée mais sur l'ensemble du spectre visible, les choses se corsent, mais surtout du point de vue physique : là encore, avec un nombre limité de couches (jusqu'à une vingtaine), les optimiseurs se mettent assez facilement d'accord sur l'optimum. Au delà de trente couches en revanche, il n'y a plus guère que Differential Evolution qui parvienne à obtenir l'optimum, pour un taux de réflexion d'environ 80% sur l'ensemble du spectre visible, ce qui est un très bon résultat.

Le dernier test consiste à tenter de reproduire la réflexion particulière trouvée sur les ailes des papillons de la famille *Morpho*. Cette réflexion a de particulier que la lumière n'est pas directement réfléchie, mais l'est selon deux angles différents : ce phénomène est à l'origine des reflets irisés bleutés. Là encore, les résultats obtenus sont extrêmement bons, puisque l'on parvient à obtenir un taux de réflexion de 100%, qui se retrouve, finalement, être trop "bon" par rapport à ce que l'on peut trouver dans la nature. Toutefois, le taux de réflexion et l'architecture des structures peuvent être améliorées pour coller au plus près à ce que l'on s'attend à obtenir en ajoutant de nouvelle contraintes sur les structures, comme prendre en compte leur poids par exemple. D'un point de vue général, on observe là aussi d'excellents résultats pour Differential Evolution.

En fait, il est intéressant de noter que sur l'ensemble de ces problèmes, l'algorithme qui fonctionne le mieux est Differential Evolution, l'algorithme le plus proche d'une évolution "naturelle". Enfin, les tests effectués sur ce problème montrent encore une fois

la supériorité des mutations quasi-aléatoires pour CMA-ES, comparées à des mutations purement aléatoires.

1.9 Conclusion

Les travaux présentés dans cette thèse peuvent être articulés en deux axes :

Dans la première partie, nous avons dans un premier temps étudié le comportement des principaux algorithmes d'optimisation boîte noire dans des conditions rarement considérées : en présence d'un grand nombre de variables "inutiles" ou en très haute dimension sur des cas mal conditionnés. Dans un cas comme dans l'autre, la théorie veut que les algorithmes soient à même de gérer ces problèmes, mais la pratique prouve qu'il en est autrement que ce soit pour des raisons de temps de calcul bien trop longs ou tout simplement une impossibilité de stocker en mémoire tous les paramètres nécessaires au fonctionnement d'un algorithme (les matrices de covariance par exemple, ou tout simplement la population dans le cas de Nelder-Mead). D'autres algorithmes quant à eux (Differential Evolution essentiellement, mais aussi $(1+1)-ES$ et Particle Swarm Optimisation) obtiennent des performances à la hauteur des attentes : à même d'ignorer le variables inutiles quel qu'en soit le nombre, ou de traiter des problèmes mal conditionnés même en très haute dimension.

Dans un deuxième temps, nous avons proposé deux améliorations des algorithmes d'optimisation boîte noire : la première concerne l'utilisation des mutations quasi-aléatoires pour CMA-ES, qui a permis d'en améliorer les performances grâce à une meilleure exploration de l'espace des mutations. La seconde est l'utilisation d'une méthode tirée du domaine des statistiques, la "Sieves Method", technique qui a fait ses preuves en permettant d'obtenir de meilleurs résultats sur les cas tests considérés.

Dans la deuxième partie de cette thèse, nous avons cherché à appliquer ces enseignements sur des problèmes venant du monde réel ou en étant inspirés : des simulations de réseaux électriques, des problèmes de clustering tirés du monde réel, et la reconstitution de structures naturelles. L'objectif était double : déterminer s'il existait des différences notables entre les performances des algorithmes sur des jeux de tests artificiels et des problèmes plus réels d'une part ; d'autre part, tester si l'impact des mutations quasi-aléatoires et de la "Sieves Method" sur ces problèmes est du même ordre que sur des cas artificiels.

L'un des principaux enseignement à tirer de ceci est que si CMA-ES est le leader inconstesté des jeux de tests artificiels, dans les cas considérés ici c'est Differential Evolution qui s'impose sans conteste : bien qu'il soit parfois battu, cet algorithme est non seulement

le plus souvent le meilleur, mais fait également montre d'une bonne régularité en n'étant jamais particulièrement mauvais.

Enfin, malgré un coût certain (dans le sens ou un arrêt prématuré du processus d'optimisation conduit à un effondrement des performances), la "Sieves Method" a fait montre de résultats intéressants puisqu'améliorant sensiblement les résultats obtenus sans, que ce soit en terme de qualité de la solution que de probabilité d'atteindre l'optimum. De même, l'utilisation des mutations quasi-aléatoires pour CMA-ES montre une nouvelle fois que même dans les rares cas où les performances ne sont pas améliorées, elles ne sont pas non plus diminuées mais que bien souvent on peut observer une nette amélioration des résultats.

Chapter 2

Introduction

2.1 Generalities

> **Optimization**: The action of making the best or most effective use of a situation or resource.
>
> *Oxford Dictionary*

Optimization is found everywhere, as soon as we want to improve a process, an object, a cost, *etc.* In mathematics, optimization is the process through which we will find the minimum (or maximum) of a given function: given a function $f : \Omega \subset \mathbb{R}^D \mapsto \mathbb{R}$ we want to find x^*, such that $\forall x \in \mathbb{R}^D$ where D is the dimension of the problem then $f(x^*) \leq f(x)$ if we want to find the minimum, or $f(x^*) \geq f(x)$ if we want to find the maximum.

In trivial cases, this can be done through a simple analysis, and every high-school student learns to do it. In some cases however, the function is far too complex to allow such analysis. Specialized optimizers must be used.

2.1.1 Families of optimizers

Based on Hessians Optimizers have been around for a long time. In fact, in the algorithms used to this day, one dates back from the seventeenth century: the Newton's Method (also known as Newton-Raphson method). Introduced in its primitive form by Isaac Newton in his *De analysi per aequationes numero terminorum infinitas* written in 1669, later refined by Joseph Raphson's *Analysis Aequationum Universalis* in 1690, this method is designed to find successively better approximations of the roots (optimums) of a function.

The simple fact that this method is still used today is testament to its efficiency. In fact, when it can be used, it is one of the fastest ways to find the optimum of a given function. The problem however is that it is necessary to compute the first and second order derivatives of the optimized function. Even in cases where it would be theoretically feasible, computing the Hessian - or worse, inverting it - would take far too long to be efficient, if at all possible.

Based on gradients In order to address this problem was developed an evolution of the Newton's Method, the Quasi-Newton Method [Davidon, 1959]. Since then, multiple variants of the Quasi-Newton Methods were proposed such as Broyden–Fletcher–Goldfarb–Shanno (BFGS), a variant independently published by four authors [Broyden, 1970, Fletcher, 1970, Goldfarb, 1970, Shanno, 1970] which now has variants of its own, the Symmetry Rank One variant [Conn et al., 1991], *etc.*

While much slower than Newton's Method (which has a quadratic rate of convergence, versus a superlinear rate of convergence for Quasi-Newton's Methods), the main advantage of Quasi-Newton over its parent is that it doesn't need to compute the Hessian: instead, the Hessian is updated with the gradient vectors successively obtained through the optimization. By not requiring to compute the inverse of the Hessian, Quasi-Newton algorithms can be used on problems where the Newton Method is far too expensive.

Based on fitness values After that came optimizers only using the value (or fitness) of the function at some sampling points in order to reach the optimum. One of such methods is NEWUOA [Powell, 2008], which works by establishing a model by quadratic interpolation in a given "trust region". After a minimization of the surrogate model, it samples some points, and, based on the *a posteriori* interpolation errors, it updates the "trust" region or the current best point.

An interesting point of such a method is that it is derivative free, all it needs is to sample and evaluate the function. The results of those evaluations will enable it to perform its interpolation.

Based on comparisons At the bottom of this stack, we find optimizers working by comparing the evaluation of two or more samples - individuals - to reach the optimum. While there are many algorithms in this family, most of them have in common that they are inspired by the natural world: Particle Swarm Optimization (or PSO) [Kennedy and Eberhart, 1995, Shi and Eberhart, 1998b] for example is inspired by

the flight of birds or the behavior of a fish school. More often however, those algorithms are, in some way, trying to reproduce natural evolution. At each step - or generation - those optimizers mutate and/or recombine the current population to produce offsprings. Each offspring is evaluated and the best ones are selected in order to become the new population. Eventually, the offsprings converge to the solution. Or at least *a* solution.

One of the main strengths of such algorithms is that they don't make any assumption on the function and don't need to know anything about it. All they require, is the ability to evaluate each individual sample and compare them. While in algorithms like NEWUOA [Powell, 2008] the fitness themselves are important, in comparison based algorithms the only thing that matters is to know how they compare, which one is better than the other.

The distinction between fitness based and comparison based algorithms is important in some applications. When there are human appreciations of an individual or for some problems originating from games, we only have a relative appreciation of two solutions: one is better, the other is worse, but it isn't necessarily easy or even possible to quantify by how much they differ. In addition, some theorems (for example those derived from [Fournier and Teytaud, 2011]) only deal with comparison-based methods.

Optimizers used in the thesis In this work, we only used optimizers based on comparisons:

- Self-Adaptive Evolution Strategies (SA-ES [Rechenberg, 1973a]), which come in three main flavours: isotropic, where there is only one mutation parameter and anisotropic with one mutation factor for each parameter (both from [Schwefel, 1977]) and anisotropic with covariance matrix[Schwefel, 1981a].
- Covariance Matrix Adaptation Evolution Strategy (CMA-ES[Hansen and Ostermeier, 2003]) where the mutation step sizes are guided by cumulative step-size adaptation and also features full covariance matrix adaptation.
- Covariance Matrix Self-Adaptation, CMSA-ES [Beyer and Sendhoff, 2008], the extension of SA for invariance w.r.t. rotations through the use of a covariance matrix.
- The "simple" $(1+1)$-ES, where the step size is updated according to the success or failure to improve (the $\frac{1}{5}$ rule)
- Nelder-Mead [Nelder and Mead, 1965], a simplex method where one point is moved at each generation.

- Particle Swarm Optimisation[Kennedy and Eberhart, 1995, Shi and Eberhart, 1998b], where the speed of the particles is influenced by the best solutions found by the particle itself and its neighborhood (a fixed subset of all existing particles).

- Differential Evolution[Storn and Price, 1997], an algorithm where where mutation is done by adding the difference of two or more solutions in the population to another one, and crossover is done coordinate by coordinate between the parent and one mutant.

In most cases, unless otherwise stated, none of the optimizers were tweaked to try to achieve the best possible results. While it would of course have been possible, the goal was to compare each optimizer as they came, in their default configuration (or one of their default configurations if there are many different).

This means that in most cases in the following chapters, the parameters of the optimizers were:

- $(1+1)-ES$: step-size multiplied by 1.5 in case of success and one-fifth rule (*i.e.*division by $1.5^{\frac{1}{4}}$ in case of failure)

- SA-iso: population size $\lambda = 12$, parent population size $\mu = 3$, mutation rate for step-sizes $\tau = \frac{1}{\sqrt{2N}}$

- SA-aniso: population size $\lambda = 12$, parent population size $\mu = 3$, and mutation rate $\tau_{global} = \frac{1}{\sqrt{2N}}$ and $\tau_{local} = \frac{1}{\sqrt{2\sqrt{N}}}$.

- SA-Cov: population size $\lambda = 12$, parent population size $\mu = 3$, and mutation rates $\tau_{global} = \frac{1}{\sqrt{2N}}$ and $\tau_{local} = \frac{1}{\sqrt{2\sqrt{N}}}$ with $\beta = 0.0873$

- CMA-ES: population size $\lambda = \lfloor 4+3\log(N) \rfloor$, and parent population size $\mu = \lambda/2$

- CMSA: $\lambda = 12$, $\mu = 3$, $\tau = \frac{1}{\sqrt{2N}}$, and a learning rate for the covariance matrix $\tau_C = \frac{1+N(N+1)}{2\mu}$

- DE: variant DE/Curr-to-best/1 [Price, 1999] with a population size of 30, and parameters $Cr = .5$, $F_1 = F_2 = .8$

- NM: $\alpha = 1$, $\gamma = 2$, $\rho = -0.5$, and $\sigma = .5$.

- PSO: population of size 30 organized in a social neighbourhoods of size 10. $\omega = 1/2\log(2)$, $\phi_g = \phi_p = \frac{1}{2} + \log(2)$, initial velocity 1 and maximum velocity $\frac{3}{2}$.

Throughout this thesis, there are many cases where each individual optimizer could have obtained better performances had their parameters been optimized. The choice however has been to not do so for a very simple reason: as can be seen in chapters 7 through 10, one of the important part of this thesis concerns applications to the real world. In many cases, people who would have to use optimizers to solve their problems wouldn't be able to tweak, much less optimize the parameters of the optimizers they plan on using. As such, the goal of the work done here was to assess the performances of the optimizers in as close a setting as they would be used later on.

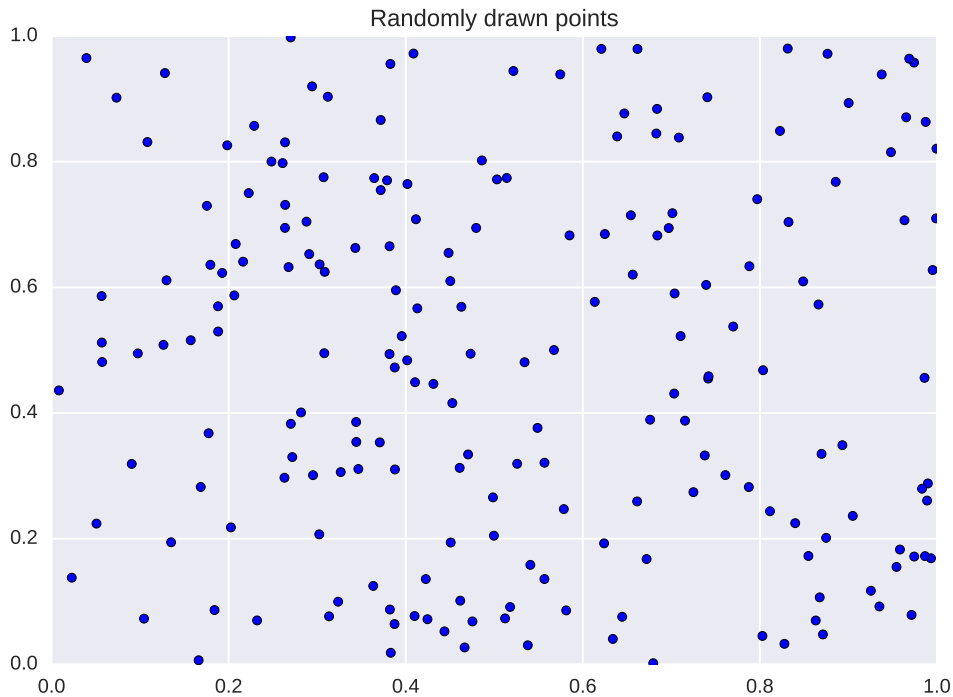
2.1.2 Randomization

Randomization is an important part of optimization. In gradient-based algorithms, it can be used to change the step-size, to select a subset of variables to optimize or to select the samples to use. In evolutionary algorithms, randomization is at the heart of the optimizer: it's through randomization that the current population produces offsprings, which will lead to an evolution of the population. In fact, randomization is so important in this process that it's the only way convergence can be guaranteed for certain forms of robustness [Gelly et al., 2007].

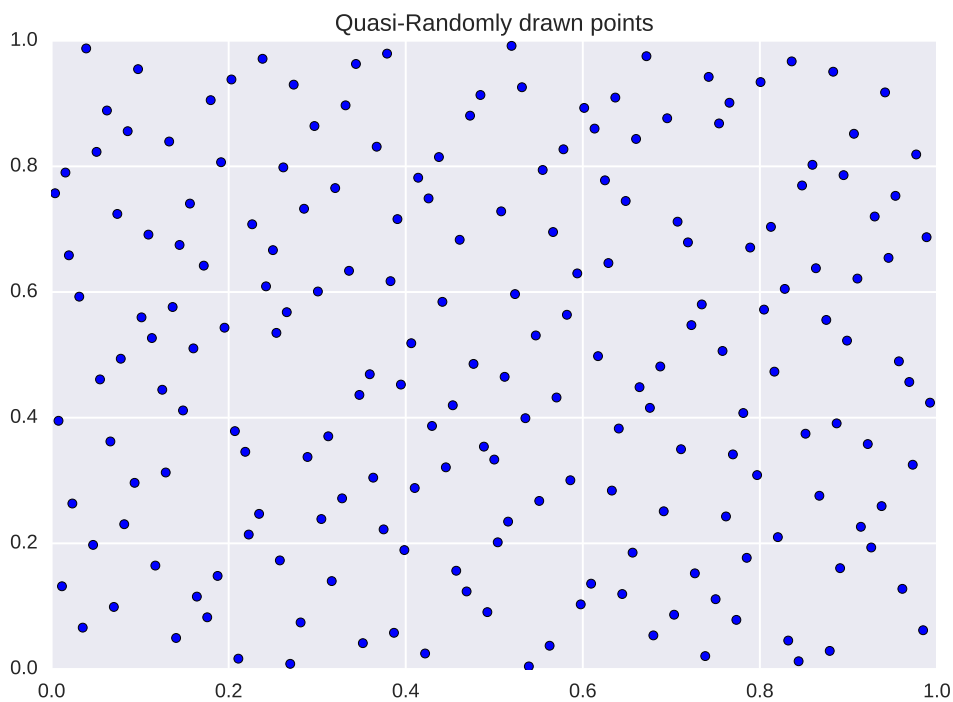
Purely random evolution is not without problems however. As can be seen in Figure 2.1a, for one hundred points uniformly drawn in $[0,1]^2$, there are some redundancies, points that are on top of one another. On the contrary, there are regions of space with not a single point. What if the solution we're looking for is in one of those spaces? Of course, we will probably get there at some point. But this is far from optimal.

On the contrary, with points drawn quasi-randomly as proposed in [Teytaud and Gelly, 2007], we can see in Figure 2.1b that the space is much better covered and that there is no overlap. Of course, quasi-random is not random. But among the quasi-random points generated, we can randomly select one. As can be seen in Figure 2.1c, while the space is a bit less covered and overlap more of a risk compared to pure quasi-random, it is still better than pure randomness. That way, we get the best of both world. Robustness through randomization; low redundancy through quasi-randomness. Of course, there are other and possibly better options to use quasi-random: the simplest one could be to shift the selected values by a random vector drawn on $[0,1]^d$ and apply a modulo 1, which has the advantage to preserve low discrepancy, to ensure low variance [Tuffin, 2004] and to avoid introducing any bias.. Another option could be to use the scrambling method [Ökten, 2002, Kocis and Whiten, 1997, Matoušek, 1998].

(a) Random



(b) Quasi-random



(c) Random + Quasi-Random

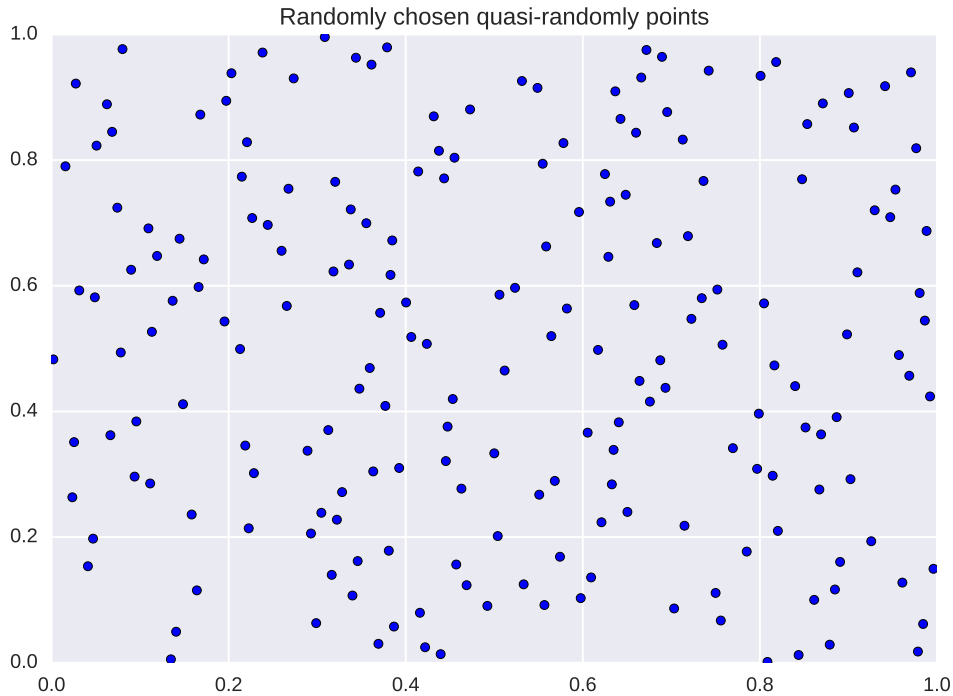


Figure 2.1 – Comparison of 100 points, drawn randomly, quasi-randomly, or randomly chosen among 50'000 quasi-randomly generated points.

2.1.3 Invariances

One of the many properties of optimizers to keep in mind is what their invariances are. For example, given a specific problem, Newton, BFGS or NEWUOA will exhibit the same performance on the base problem as in all cases where the problem is translated, rotated or its variables re-scaled (with the caveat that the initialization may need to be adjusted).

On the other hand, things are more complicated when dealing with comparison based algorithms. While they are invariant with regard to an increasing function [Gelly et al., 2007], translations (if we do not consider the initialization of the algorithms since the probability distribution for the initial points cannot be invariant), things are more complicated when we are dealing with rotations. PSO [Hansen et al., 2008] isn't invariant with regards to rotations, in the case of Differential Evolution it depends on one of its parameters [Auger et al., 2009] but others are able to deal with rotations, such as Covariance Matrix Adaptation Evolution Strategy [Hansen, 2008] or Nelder-Mead [Nelder and Mead, 1965].

A problem is said to be separable if it is possible to optimize each variable in turn by fixing all the others to a given value. Rotating a separable problem gives a non-separable problem, hence the importance of invariance with regard to rotation.

2.1.4 Benchmarks

In order to assess the performances and properties of any given optimizer, it needs to be tested on one or more standardized testbeds. Only by comparing their results in a fair way is it possible to improve anything. There exist many different testbed, each trying to propose difficult challenges to the optimizers: ill-conditioning, non-separability, valleys, non-convexity, multi-modality, *etc.*

Among them, we can cite BBOB [Hansen et al., 2010a], CEC2005 [Suganthan et al., 2005], Cute/Cuter/Cutest [Gould et al., 2003] and BBComp[1]. Despite the many properties tested by those benchmarks, they are mostly restricted to low dimension (few problems have more than 64 dimensions), but the main issue is probably the lack of real world problems.

In fact, while there are some real world problems in Cute/Cuter/Cutest, it is by far the exception. In all other cases, the problems are mathematical functions. More or less complex, but always mathematical functions. This, of course, is very useful, since it is easy to implement and easy to know if an optimizer reaches the optimum or not.

There is a risk however in only assessing the performances of optimizers on artificial benchmarks: can we be sure the properties we look for on an artificial benchmark, such

[1] The competition site can be found at `http://bbcomp.ini.rub.de/`

as multi-modality, non-separability, ill-condition, *etc.* can be found in real-problems in much the same way? How can we be reasonably sure that we didn't lose sight of the end goal? Can we say that those artificial benchmarks give a reasonable estimate of the relative performances we can expect when we want to design a piece of machinery or a concert room, to decide on the best trajectory for the space shuttles? The only way to be able to answer those questions is to experiment on real-world problems, or at the very least, on benchmarks whose problems are inspired or taken from the real-world.

2.2 Outline & Contributions

Chapter 2 introduces the domain and some of the important notions found throughout this Ph.D. thesis. After that, Part I will present some new interesting properties of existing algorithms as well as present some improvements. Part II applies existing optimizers as well as the improvements previously presented to real world problems in order to assess their performances outside of an artificial benchmark. Finally Part III concludes in Chapter 11 and gives some perspectives in Chapter 12.

2.2.1 Methods

Part I is dedicated to the study of up to now passed by properties or the inclusion of improvements to existing algorithms. Chapter 3 studies the impact of useless variables on the optimization process and introduces the notion of invariance with regard to useless variables. While ill-conditioning is well studied, it has most often only been the case in low dimension. Chapter 4 will show a new picture in very high dimension.

Chapter 5 will show the impact of using Quasi-Random mutations for CMA-ES as well as some interesting results involving porfolios, whereas Chapter 6 introduces Progressive Widening, known as the Sieves Method to statisticians.

2.2.2 Applications

Part II is all about using optimizers on real world problems, or at least inspired by the real world. Chapter 7 will compare optimizers on a unit commitment problem, where the goal is to be able to meet a power production demand by using water stocks. On the same type of problems, chapter 8, inspired by portfolios [Baudis and Posik, 2014, Gagliolo, 2010] combines two control policies into one to take advantage of the strengths of each one.

Chapter 9 tests the performances of an optimizer modified with Progressive Widening on a real world inspired benchmark, while Chapter 10 reproduces a naturally evolved

structure, comparing the performances of several optimizers in the process, including a quasi-random variant of CMA-ES.

Part I

Methods

Chapter 3

On the codimension of the set of optima: large scale optimization with few relevant variables

This chapter's content comes from the paper Berthier, V. and Teytaud, O. (2015a). 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. Its abstract was:

The complexity of continuous optimisation by comparison-based algorithms has been developed in several recent papers.Roughly speaking, these papers conclude that a precision can be reached with cost $\Theta(nlog(1))$ in dimension n within polylogarithmic factors for the sphere function. Compared to other (non comparison-based) algorithms, this rate is not excellent; on the other hand, it is classically considered that comparison-based algorithms have some robustness advantages, as well as scalability on parallel machines and simplicity. In the present paper we show another advantage, namely resilience to useless variables, thanks to a complexity bound $\Theta(mlog(1))$ where m is the codimension of the set of optima, possibly $m \ll n$. In addition, experiments show that some evolutionary algorithms have a negligible computational complexity even in high dimension, making them practical for huge problems with many useless variables.

3.1 Introduction

In many, if not most, optimization problems, different variables have different weight in the evaluation of the fitness function: one such example is the simple ellipsoid $f(\mathbf{x}) =$

$10^6 x_1^2 + \sum_{i=2}^{D} x_i^2$, where one variable ($x_1$) has a "weight" one million times more important than the other variables. We say that the condition number of the problem is of one million.

In some cases though, some variables do not only have a far lesser impact on the evaluation function than others, their impact is nil. By opposition to the other "critical" variables, they are called "useless". One such case can be seen when optimizing a neural network controller with a sparsity criteria where many weights are set as zero: all variables linked to neurons with those weights have no impact on the fitness function. More importantly, this phenomenon can be seen in parameter estimation problems or in genetic programming where many variables may be useless due to some other variables.

This phenomenon can be observed in reinforcement learning [Sutton, 1996, Ratitch and Precup, 2004, Kearns et al., 1999], evolution of trees [Zhang et al., 1997], Nash equilibrium [St-Pierre et al., 2011] or Support Vector Machines [Girosi, 1998]. [Powell, 2008] also mentions very flat directions as a key point in some optimization problems. An important question is then to know how and when those useless variables impact the optimization process, and if it is possible to overcome it.

Notations. We here introduce some notations that will be used throughout this chapter. d is the dimension of the search space; we consider optimization in $D = (0,1)^d$. m is the codimension of the set of optima, *i.e.*$m = d - u$ where u is the dimension of the set of optima.

x^* is an optimum of the objective function. The objective function, also known as fitness function, is $f : D \to \mathbb{R}$. $\tilde{O}$ denotes an upper bound within polylogarithmic factors.

Impact of useless variables on algorithms initialization. Some optimizers have a population size linear in the number of variables: Newuoa [Powell, 2008] generates an initial population of size $2d+1$. Newuoa uses this population for building a first approximation of the Hessian. Nelder-Mead generates an initial population of size $d + 1$. Only when this initial population is generated, points which depend on the fitness values are generated based on the ranking of this initial population. Finite-differences methods will generate an initial population of size $d+1$ for estimating the gradient.

For those optimizers, we can easily see that a small number of useless variables is not an issue, but it soon becomes one as their number increases. In practice it is often unfeasible: a population of one million individuals of one million double variables requires tera-bytes of RAM (the exact number depending on double precision on the considered system).

Many Evolution Strategies have a dimension-independent population size, or at worse a logarithmically increasing one. However, those that rely on covariance matrix adaptation

(*e.g.* CMA-ES, CMSA-ES, *etc.*) suffer from the same kind of problem: at some point, the resources needed to store this matrix become insufficient. Other algorithms, not suffering from either of those problems, can be said to be robust w.r.t. useless variables.

Runtimes in the presence of useless variables. When assessing the performances of an optimizer, two measures can be used. The first and arguably most used one is to compare them by the number of function evaluations required to reach the optimum. As it is independent of implementation, it is easier to use. However, there are huge gaps between the "internal costs" of different optimization algorithms: this cost can be very high for algorithms based on covariance matrix adaptation. In fact, it can be so high that those algorithms are unable to deal with problems of dimension 10'000 or more. On the other hand, some algorithms (*e.g.* Differential Evolution, Particle Swarm Optimization, *etc.*) can be used with a hundred times more variables without problem.

The second possible measure is to compare algorithms on their runtimes: in some cases, the number of function evaluations is not important, as long as we can get the result fast. This however is a difficult measure to use: it is implementation dependent, making indirect comparisons (*e.g.* from two different chapters) at best suspect; it does not make any difference between the time needed to perform a function evaluation, and the time needed by the algorithm itself. In most cases, the later is supposed negligible compared to the former. With a high number of variables, this assumption does not hold anymore in some cases: CMSA-ES and CMA-ES which need to compute the eigen values and eigen vectors of the covariance matrix require a lot of time, far more than necessary for a function evaluation.

One possible way to avoid this problem would be to use a diagonal covariance matrix instead of the full matrix. The goal here however is to assess the impact of a high number of useless variables on the performances and behavior of standard algorithms, not to find ways to make it work.

Outline of the chapter. Section 3.2 investigates theoretically the impact of the codimension in evolutionary computation. Section 3.3 discusses existing algorithms from the point of view of the codimension of the set of optima, and from the point of view of their invariance properties. Section 3.4 shows experimentally that optimizing in very high dimensional search spaces is possible for evolutionary computation.

3.2 Theoretical analysis: impact of the codimension on the required number of function evaluations

We first summarize the state of the art.We then study lower bounds (Section 3.2.1) and upper bounds (Section 3.2.2). We first discuss the case of a codimension m equal to the dimension d, *i.e.*a set of optimum reduced to a single point. Sections 3.2.1 and 3.2.2 will discuss the extension of these results to codimension $m < d$. [Fournier and Teytaud, 2011] has shown that the number of function comparisons for finding the optimum with precision ε is $\Theta(d\log(1/\varepsilon))$ for algorithms based on comparisons. The upper bound on the number of iterations for reaching a given precision is for some specific comparison-based algorithm on the sphere function and the lower bound of the same quantity is in the case of any family of functions with unique optimum, when the optimum can be anywhere in the domain (optimum uniformly randomly drawn in the domain, or worst case over optima in the domain), and for a precision (stopping criterion) defined either in terms of distance to the optimum, or in terms of fitness values, if the fitness values $f(x) - f(x^*) = \Omega(||x - x^*||^\alpha)$ for some $\alpha > 0$.

These results are based on information theory. Basically, a comparison provides one bit of information, so if we need a precision such that the optimum should be described with M digits (in binary), we need M comparisons. More generally, a ranking of λ offspring provides at most $\log_2(\lambda!)$ bits of information, and detailed results for algorithms using a selection operator of μ individuals over λ can be derived in a similar manner. [Jamieson et al., 2012] obtained a more general result (including various models of noise), at the expense of a different dependency in ε; they get: (i) a lower bound on the number of comparisons $\Omega(d\log(1/\varepsilon))$ on the number of iterations before reaching an expected precision ε. (ii) an upper bound on the number of comparisons $O(d\log(1/\varepsilon)^2)$ on the number of comparisons before reaching an expected precision ε, reached by an explicit algorithm.

3.2.1 Lower bound

The lower bound in [Fournier and Teytaud, 2011] can be adapted to our setting as follows:

Theorem 1 (corollary of [Fournier and Teytaud, 2011]): *Consider a fixed $\delta < 1$. Consider the function $f_{x^*,R,d,m} : x \mapsto \sum_{i=1}^{m} (R(x - x^*))_i^2$ where R is a rotation of $\mathbb{R}^d$ and $x^* \in D$. Consider F_m the set of such functions. Consider a comparison-based algorithm A. Then, there is a universal constant K (depending on δ only), such that if for all functions in F_m, with probability at least $1 - \delta$, A outputs $\hat{x}$ such that $||\hat{x} - x^*|| \leq \varepsilon$ after n comparisons,*

then

$$n \geq K \times m \times \log(1/\varepsilon).$$

Proof: Consider $F'_{m,d}$ the restriction of $F_{m,d}$ to the identity matrix for R. Consider the optimization in $(0,1)^d \times \{0\}^{d-m}$. Then by [Fournier and Teytaud, 2011], the number n ensuring precision ε is at least $K \times m \times \log(1/\varepsilon)$, for some universal K depending on δ only. $F'_m \subset F_m$, hence a lower bound for F'_m also holds for F_m. This yields the expected result. □

This result shows that the the rate depends on the codimension rather than on the dimension itself.

3.2.2 Upper bound

The result from [Jamieson et al., 2012], for the upper bound and in the noise-free case, is as follows:

Theorem 2 (corollary of [Jamieson et al., 2012]): *Consider a fixed $\delta < 1$. Consider the function $f_{x^*,R,d,m} : x \mapsto \sum_{i=1}^{m} (R(x-x^*))_i^2$ where R is a rotation of $\mathbb{R}^d$ and $x^* \in D$. Consider $F_{m,d}$ the set of such functions, for a given d and a given m. Then, there is a universal constant K (depending on δ only) and an optimization algorithm A, such that for all functions in $F_{m,d}$, with probability at least $1-\delta$, A outputs $\hat{x}$ such that $||\hat{x}-x^*|| \leq \varepsilon$ after n comparisons, where*

$$n = \lceil K \times m \times \tilde{O}(\log(1/\varepsilon)^2) \rceil. \tag{3.1}$$

Proof: The algorithm in [Jamieson et al., 2012] uses coordinate-wise line search, which can not be applied directly for our rotated framework. However, as pointed out in [Jamieson et al., 2012] (Section 5.1: " an analysis with the same result can be obtained with [...] chosen uniformly from the unit sphere"), the same result holds with randomly rotated search directions. The algorithm with randomly rotated search direction applied to $f_{x^*,R,d,m}$ exactly mimics the behavior of the algorithm on $f_{x^*,R,m,0}$. This yields the expected result. □

We point out that evolution strategies (usually) also have this invariance property. However, we did not use evolution strategies in the proof because there is no formal proof of convergence for every variant of evolution strategies. Nonetheless, [Auger, 2005] is

close to such a result for one specific version of evolution strategies (up to the sign of the constant), Theorem 2 shows an upper bound for comparison-based methods, and there is a big hope that Theorem 2 could be adapted to evolution strategies if the constant in [Auger, 2005] is proved negative.

The gap with the lower bound is the exponent 2 on $\log(\frac{1}{\varepsilon})$ in Eq. 3.1. We do not reduce the gap in the general case, but we propose the following partial result, using $F''_{m,d} = \{f_{x^*,R,d,m}; \forall i x_i^* \neq 0, R$ has all coefficients in $\{0,1\}\}$.

Theorem 3: *Consider a fixed $\delta < 1$. Consider the family $F''_{m,d}$ of objective functions. Then, there is a universal constant K (depending on δ only) and an optimization algorithm A, using the parameter m as input, such that for all functions in $F_{m,d}$, with probability at least $1-\delta$, for ε sufficiently small, A outputs $\hat{x}$ such that $||\hat{x}-x^*|| \leq \varepsilon$ after n comparisons, where $n = \lceil K \times m \times O(\log(1/\varepsilon)).\rceil$*

Remarks: We prove the upper bound for permutations of coordinates, and not for the complete set of rotations. We assume that m is known; we conjecture that this assumption can be removed. The result is for ε sufficiently small.

Proof:

Step 1: consider many algorithms. Consider $I = \{(i_1,\ldots,i_m) \in \{1,\ldots,d\}^m; i_1 < i_2 < \cdots < i_m\}$. The cardinality of I is $z = d!/(m!(d-m)!)$. For each i, consider the algorithm A_i realizing the upper bound in [Fournier and Teytaud, 2011] with probability $1-\delta/(3z)$, for some number of function evaluations w, for any sphere function restricted to m components $i = (i_1,\ldots,i_m)$. By union bound, all the algorithms reach this bound, with probability at least $1-\delta/3$.

Step 2: a portfolio of algorithms, and algorithm selection. Consider now the algorithm A running all the A_i concurrently, in a round. However, the algorithm spends half his computational effort on the A_i which has found the best point up to now, and distributes the remaining computational power evenly over the other A_j. So a round of A is as follows:

(i) Spend one function evaluation on each A_j, $j \in I$. This costs z function evaluations.

(ii) Spend z function evaluations on the A_{j^*}, with j^* the index of the algorithm which has proposed the best search point (randomly break ties).

The overall algorithm repeats (i) and (ii) up to the available budget.

Step 3: eventually, only the right algorithm is selected. Consider the solver A_{k^*} where k^* is the family of the $R^{-1}(e_j)$ for $j \leq m$. Only this solver, among the A_j, can converge to the optimum. Hence, for ε' sufficiently small, A_{k^*} always wins the comparison after it reaches optimality within precision ε'. The upper bound states that such a precision is reached with probability at least $1-\delta/3$ when the number of rounds is at least w.

When this precision ε' is reached, $j^* = k^*$, and from now on A_{k^*} spends half of the computation budget.

Step 4: the budget. We have seen that A^{k^*} spends half of the computation budget, except possibly for the early rounds (before reaching precision ε', see step 3). Let us now show that A^{k^*} spends one fourth of the whole computation budget, when the requested precision is small enough.

Let us choose $\varepsilon < \varepsilon'$ such that the required number of rounds for A_{k^*} to reach precision ε with probability at least $1-\delta/3$ is at least twice more (i.e. $2w$) than the budget w.Such an ε exists by the lower bound. With probability $1-\delta$, when A_{k^*} reaches such a precision ε,

- the overall number of rounds is at least $2w$ (by the use of the lower bound, above);
- and during the second half of these $\geq 2w$ rounds at least one half of the evaluations have been spent for A_{k^*} (by Step 3).

Therefore it has spent at least one fourth of the budget when this number of rounds $2w$ is reached.

Step 5: concluding. With probability at least $1-2\delta/3$, one fourth of the budget has been devoted to A_{k^*} when the number of rounds is $\geq 2w$. With probability at least $1-\delta/3$, A_{k^*} has the rate provided by the upper bound. This provides the expected result.

□

Theorem 2 provides a bound on the runtime, linear as a function of the codimension m, but quadratic as a function of $log(\frac{1}{\varepsilon})$. We show in theorem 3 that, for sets of optima with codimension m aligned with axes, the quadratic dependency in $log(\frac{1}{\varepsilon})$ becomes linear. We conjecture that this holds in the general case of codimension m.

3.3 Algorithms & their invariances

Section 3.3.1 discusses invariance in optimization algorithms. Section 3.3.2 presents the optimization algorithms we consider.

3.3.1 Old and new invariances

Invariance is a classical consideration in optimization. Let us distinguish several kinds of invariance (the fifth one is a new kind of invariance in which we are particularly interested in the present chapter):

(i) Invariance w.r.t. translations is hard to achieve, due to the initialization; a probability distribution for the initial search point(s) can not be translation invariant. However, up to the initialization issue, many algorithms are invariant by translations of the objective function. It is sufficient to prove lower bounds for evolution strategies [Jägersküpper and Witt, 2005, Jägersküpper, 2006].

(ii) Invariance by composition with increasing functions is at the heart of extensions of these lower bounds to a more general setting, using information theory [Fournier and Teytaud, 2011] - basically, a comparison can provide only one bit of information, hence there is a limited rate for comparison-based algorithms.

(iii) Invariance w.r.t. rotations does not always hold, as discussed below for various algorithms. Most algorithms are invariant w.r.t. permutations of indices. Anisotropic evolution strategies [Beyer, 2001] provide invariance w.r.t. rescaling of variables (up to the initialization), but not w.r.t rotations.

(iv) Invariance w.r.t. linear transformation (not only rotations) is addressed in e.g. the Newton method in mathematical programming. It is approximated without expensive computation of the Hessian in the BFGS [Broyden, 1970, Fletcher, 1970, Goldfarb, 1970, Shanno, 1970] method. Up to the initialization, black-box counterparts of the quasi-Newton methods ensure similar invariances [Powell, 2008]. In the field of evolution strategies, the most well known methods which ensure invariance w.r.t. linear transformations are CMAES [Hansen and Ostermeier, 2003] and CMSA [Beyer and Sendhoff, 2008] both providing invariance with respect to rotations.

(v) This chapter discusses another kind of invariance: the fact that an algorithm is invariant w.r.t. addition of useless variables. This invariance is related to the codimension: when k variables are irrelevant in dimension d, then the codimension is at most $d-k$. An algorithm is said to be invariant w.r.t. addition of useless variables if this addition has no impact on the performances of the algorithm: the best obtained fitness with and without useless variable is the same, and it is reached after the same number of function evaluations.

3.3.2 Algorithms used in our experiments, and their invariances

Parameters used for the nine algorithms in our comparison are (with d as the dimension presented below).

The optimizers used have some invariance properties:

- CMAES has some invariance properties w.r.t. rotations and translations [Hansen, 2008], except (as most algorithms) for the initialization which, as discussed above, can not be translation invariant. CMAES is asymptotically invariant by rescaling of variables. On the other hand, CMAES is not invariant by addition of useless variables.
- SAiso and SAaniso are not invariant by rotation. They are invariant for rescaling of variables, up to the initialization.
- CMSA has the same kind of invariances as CMAES. CMSA is the extension of SA for invariance w.r.t. rotations. The computational cost of CMSA is higher than the one of SA. As CMAES, it is not invariant by addition of useless variables.
- SAcov is invariant for all invariance criteria discussed here.
- DE is invariant for all invariance criteria discussed here when $Cr = 1$; but not w.r.t rotations when $Cr < 1$.
- $(1+1)-ES$ is invariant for all invariance criteria discussed here.
- NM has the same kind of invariances w.r.t. rotations and translations as CMAES and CMSA.
- PSO is not invariant for rotations [Hansen et al., 2008].

For all algorithms, the initialization is as follows. Each coordinate of each individual is randomly drawn according to a Gaussian random variable with zero mean and standard deviation 6 in order to cover the domain of each function in the BBOB testbed which is $[-5,5]^D$.

3.4 Experiments

Test cases & criteria. We use the functions from the BBOB test set, and perform experiments with additional useless variables, *i.e.*we have codimension $m = 40$, and dimension $d = m + u$ with $u = 100, 1000000$ useless variables.

Other experiments have been performed with $m = 2,3,4,5,8,10,16,20,32,64$, and also with $u = 10000$; results were in agreement with results presented below with $m = 40$ and $u \in \{100, 10000\}$.

We consider the expected fitness value (y-axis; the fitness at the optimum is subtracted as all our algorithms are invariant by addition of a constant to the optimum), for given computation times (x-axis).

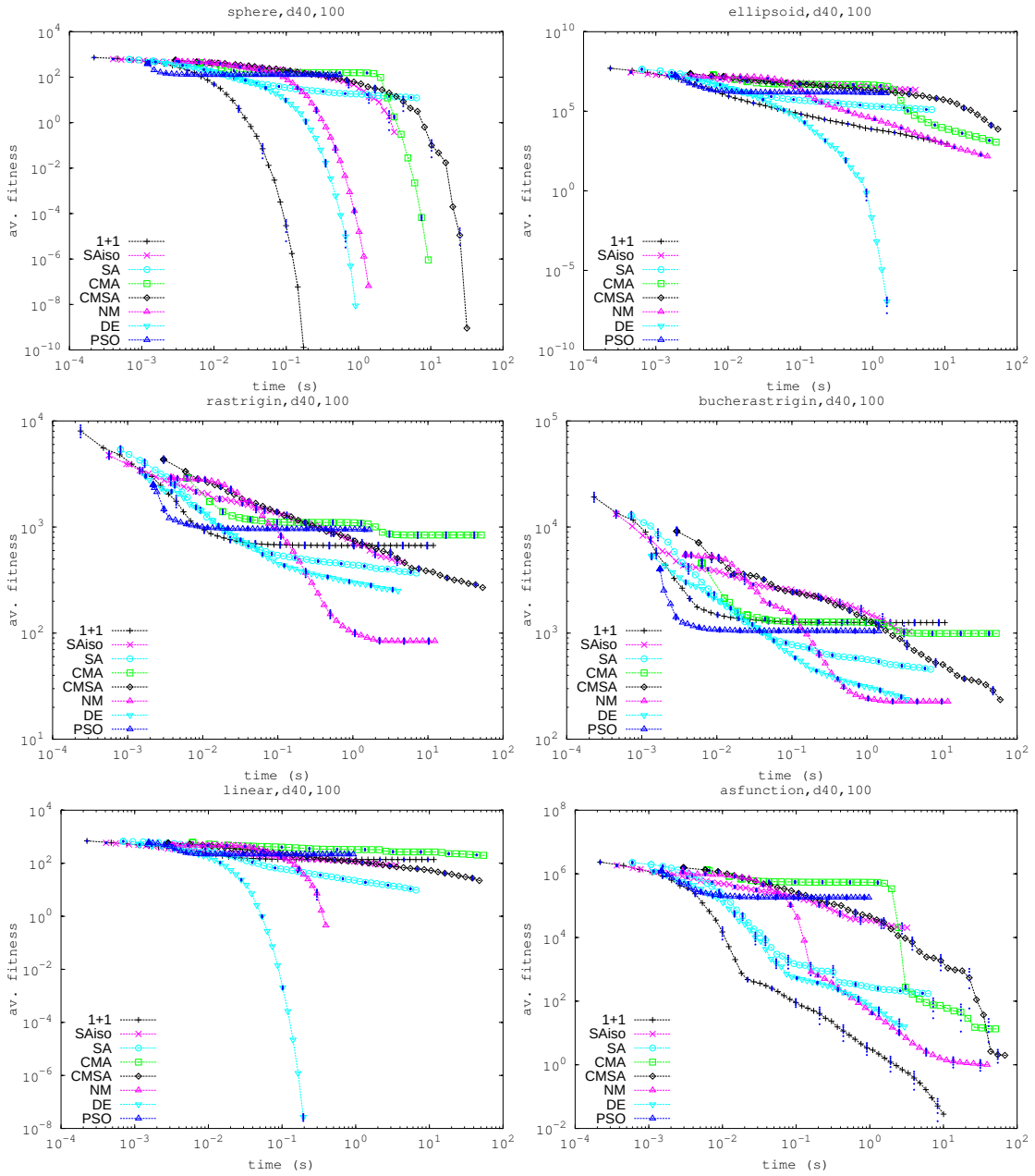


Figure 3.1 – Expected fitness value w.r.t computation time, for functions f1 to f6 in Bbob, respectively, in the case of 100 useless variables. A zoomable and colored version is available at `http://www.lri.fr/~teytaud/uv.pdf`.

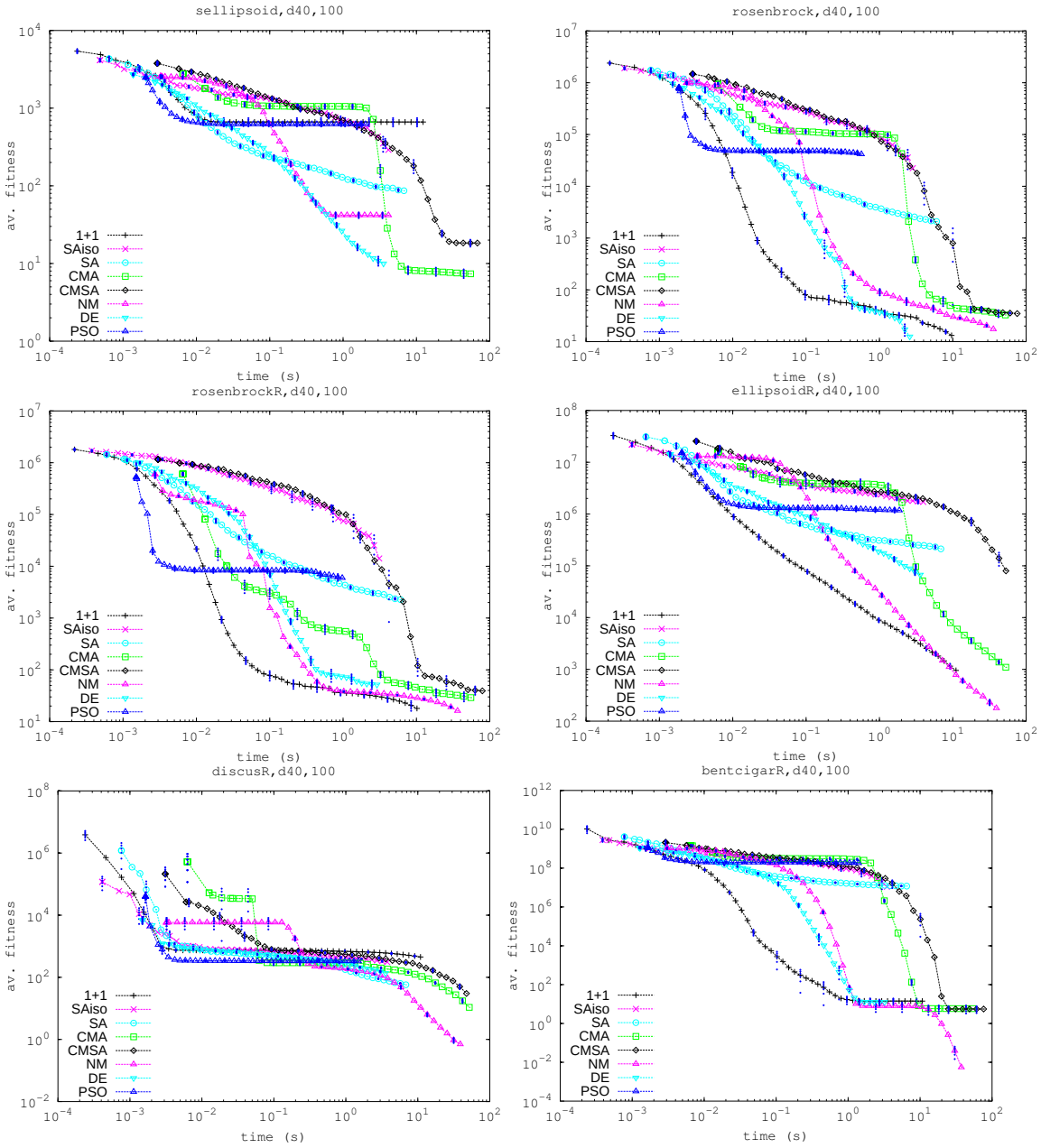


Figure 3.2 – Expected fitness value w.r.t computation time, for functions f7 to f12 in Bbob, respectively, in the case of 100 useless variables. A zoomable and colored version is available at `http://www.lri.fr/~teytaud/uv.pdf`.

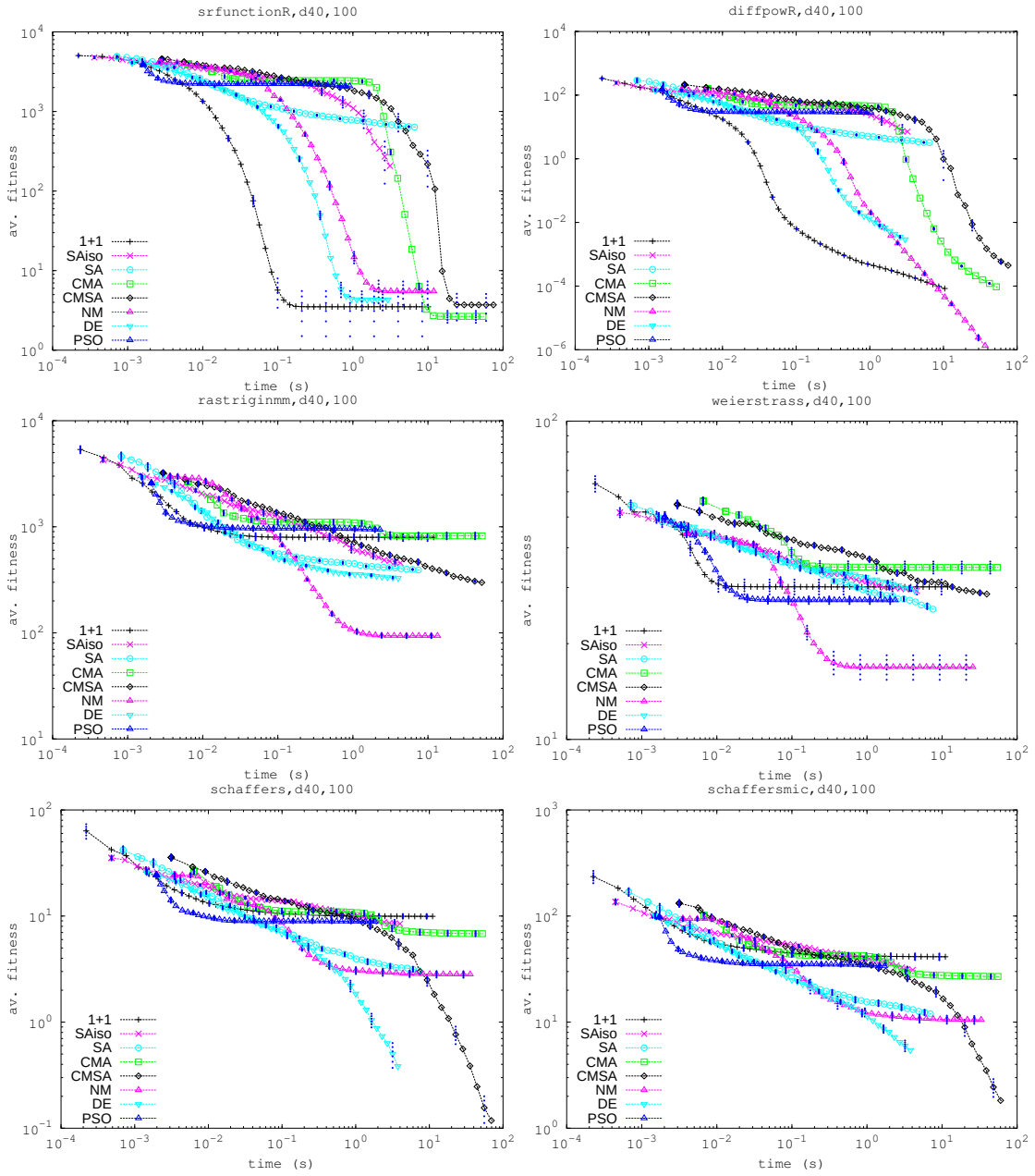


Figure 3.3 – Expected fitness value w.r.t computation time, for functions f13 to f18 in Bbob, respectively, in the case of 100 useless variables. A zoomable and colored version is available at `http://www.lri.fr/~teytaud/uv.pdf`. Confidence intervals are displayed for one point out of four; they are very small and almost invisible.

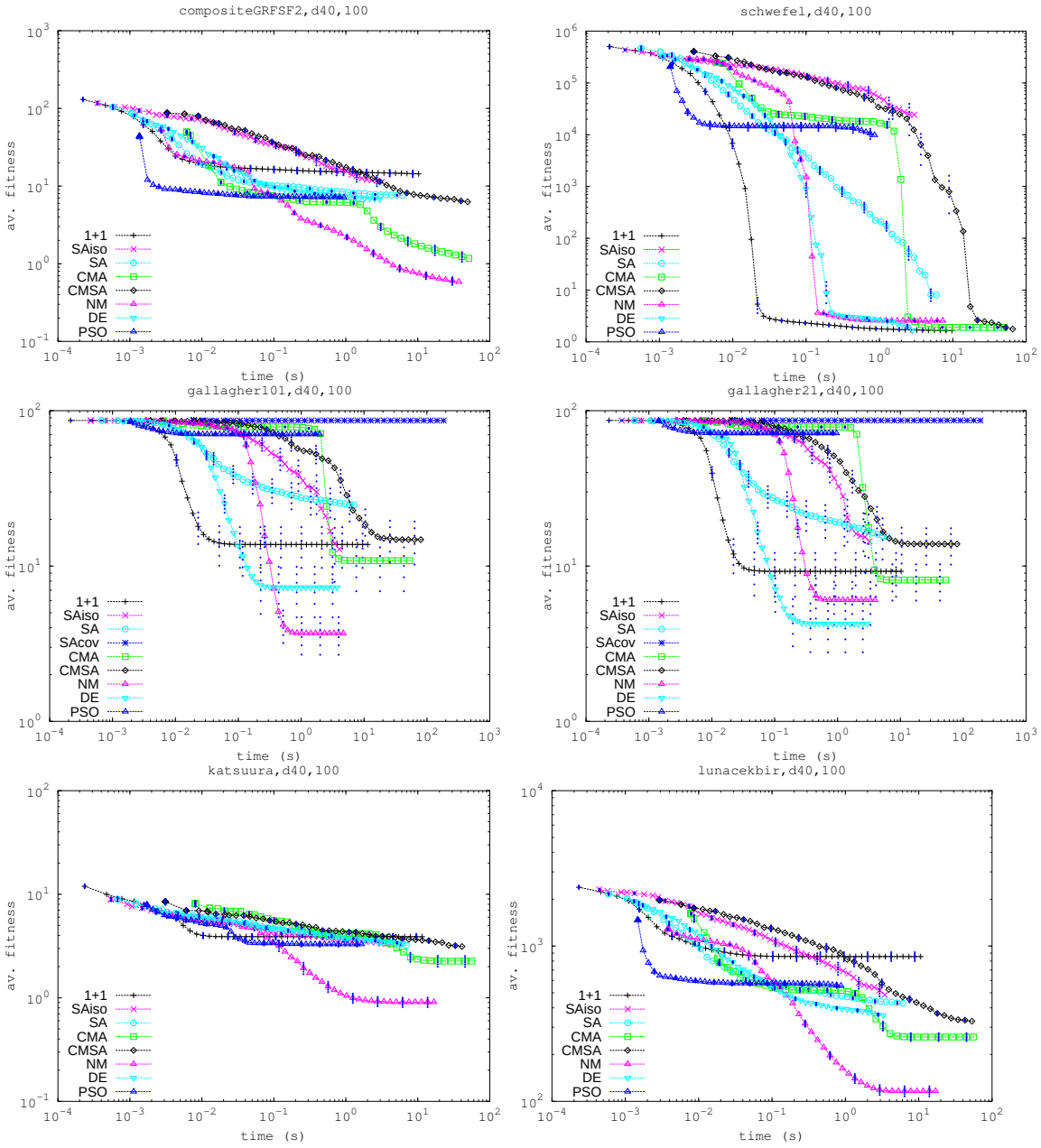


Figure 3.4 – Expected fitness value w.r.t computation time, for functions f19 to f24 in Bbob, respectively, in the case of 100 useless variables. A zoomable and colored version is available at `http://www.lri.fr/~teytaud/uv.pdf`. Confidence intervals are displayed for one point out of four; they are very small and almost invisible.

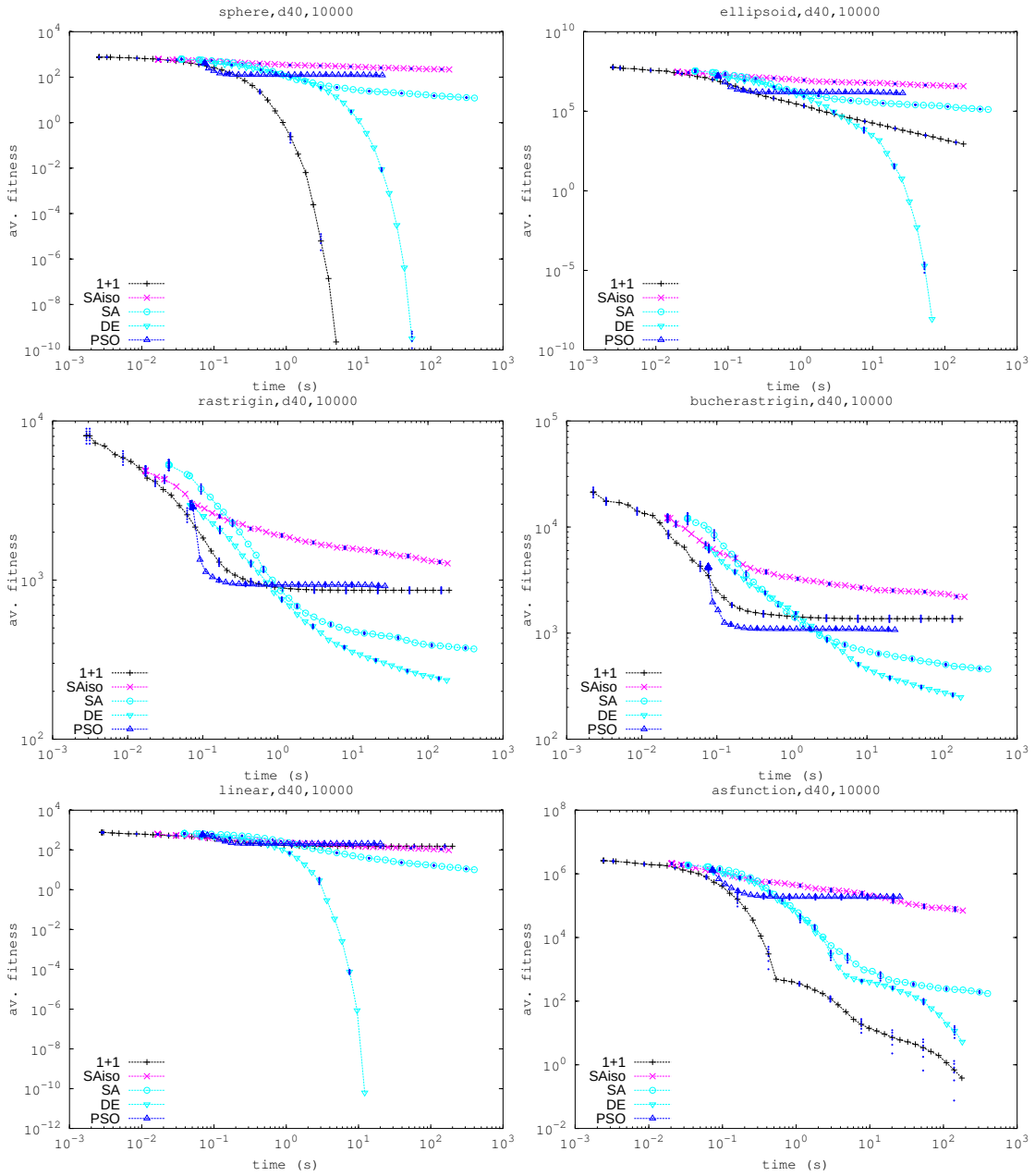


Figure 3.5 – Expected fitness value w.r.t computation time, for functions f1 to f6 in Bbob, respectively, in the case of 10000 useless variables. A zoomable and colored version is available at `http://www.lri.fr/~teytaud/uv.pdf`.

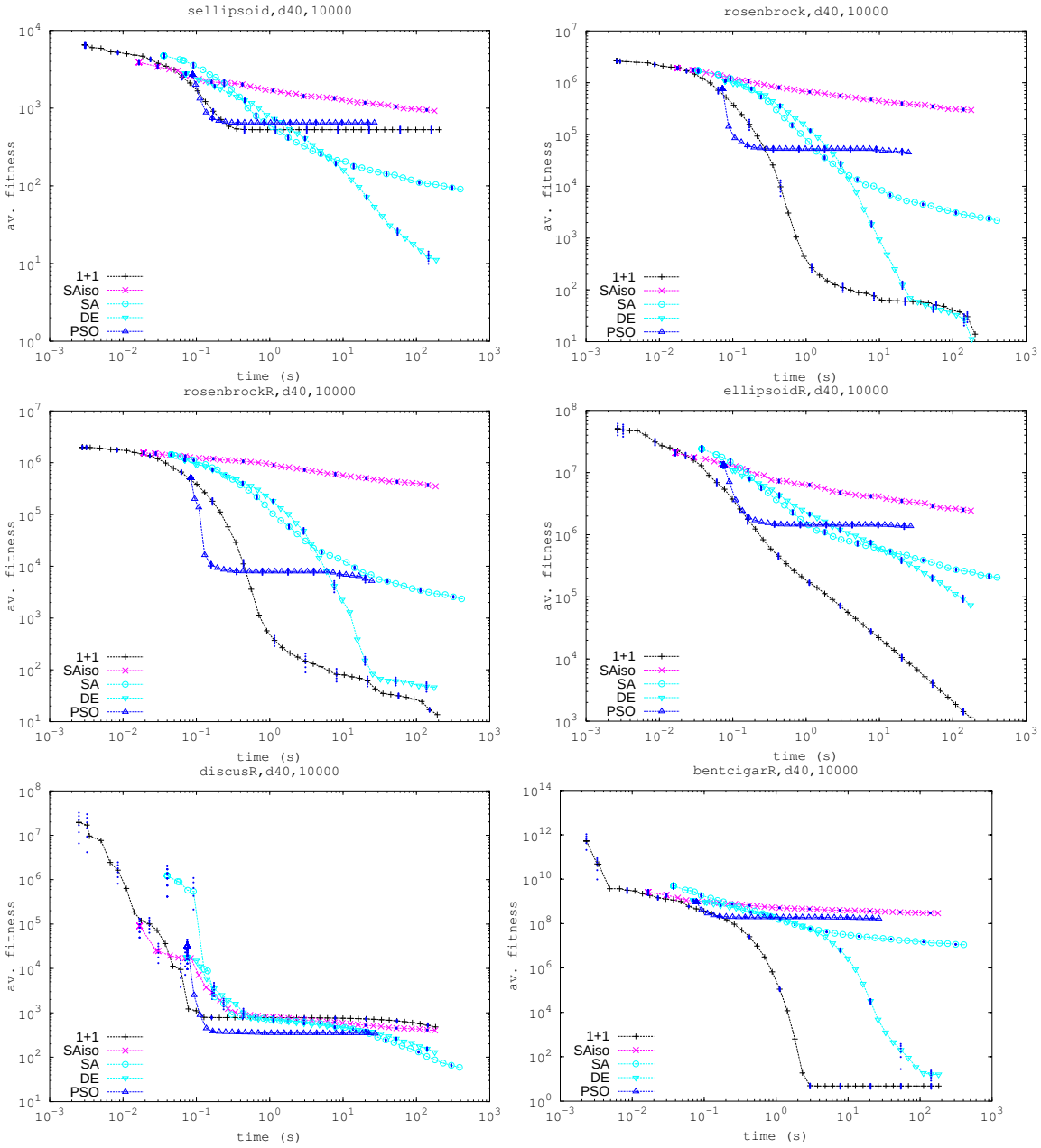


Figure 3.6 – Expected fitness value w.r.t computation time, for functions f7 to f12 in Bbob, respectively, in the case of 10000 useless variables. A zoomable and colored version is available at `http://www.lri.fr/~teytaud/uv.pdf`.

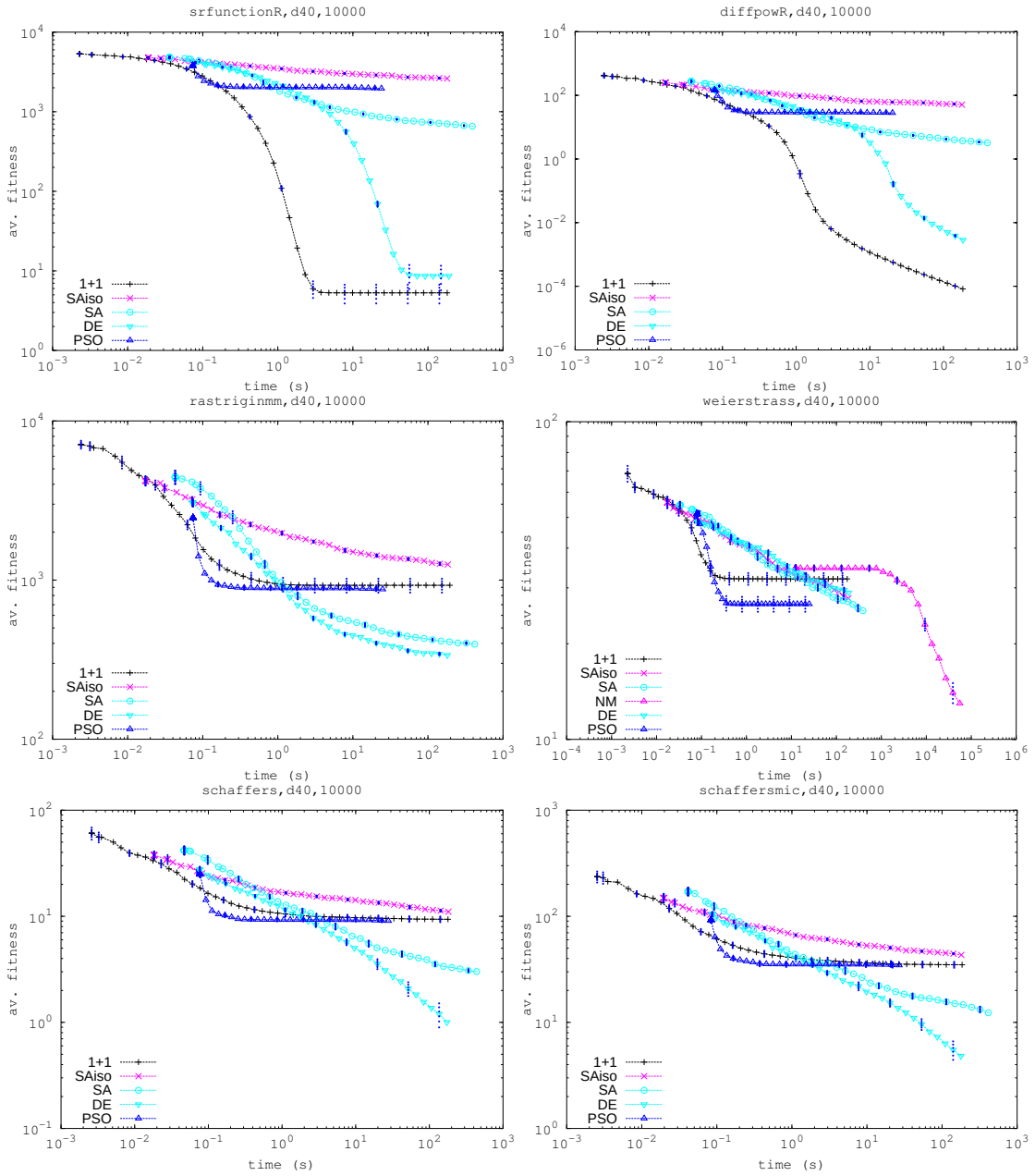


Figure 3.7 – Expected fitness value w.r.t computation time, for functions f13 to f18 in Bbob, respectively, in the case of 10000 useless variables. A zoomable and colored version is available at `http://www.lri.fr/~teytaud/uv.pdf`.

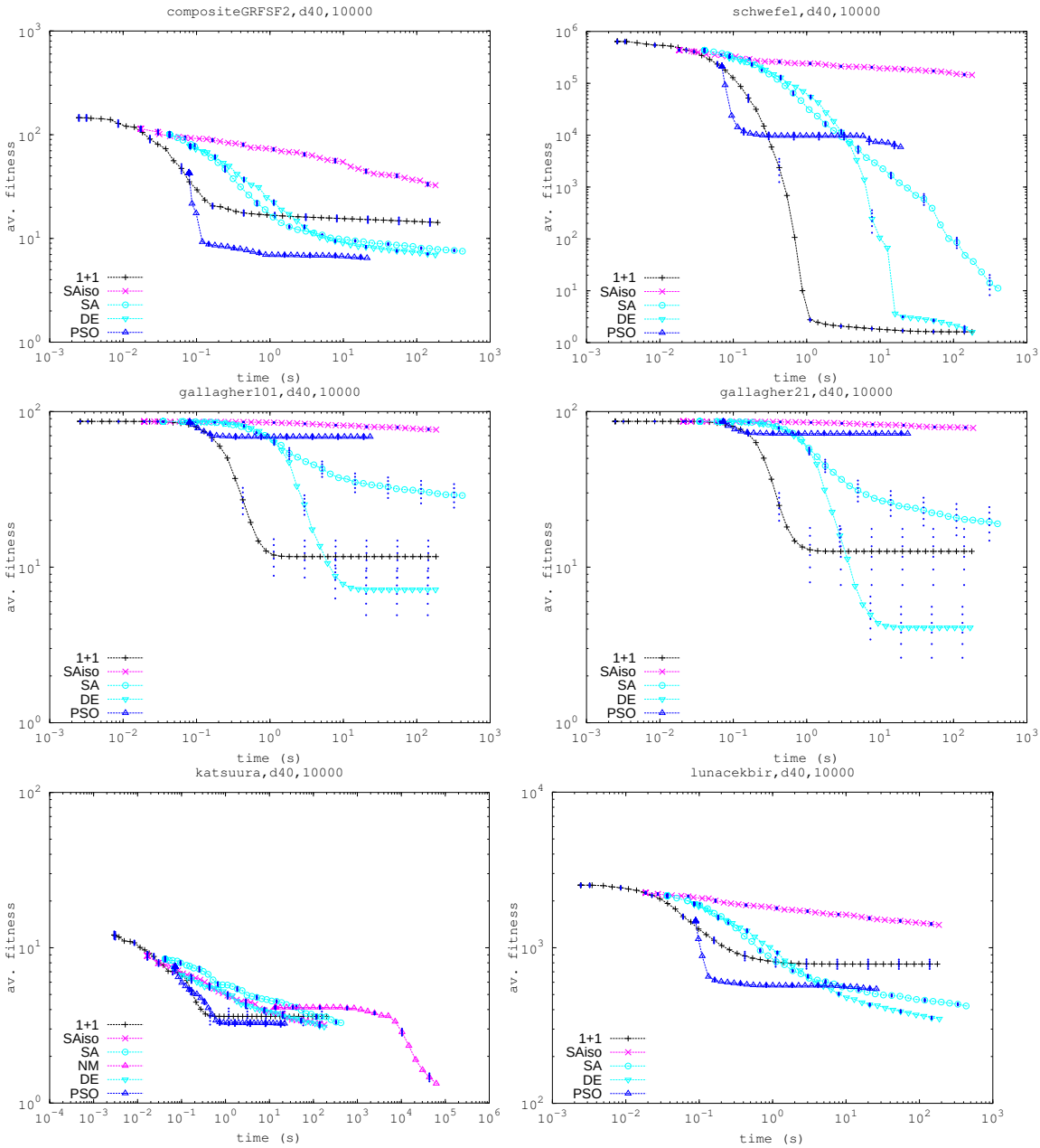


Figure 3.8 – Expected fitness value w.r.t computation time, for functions f19 to f24 in Bbob, respectively, in the case of 10000 useless variables. A zoomable and colored version is available at `http://www.lri.fr/~teytaud/uv.pdf`.

The x-axis is computation time, because for large number of variables the internal computation time of considered algorithms is not negligible. In fact, many algorithms could not run at all with such high dimension.

We did not permute coordinates, so that the useless variables are always the last ones. However, all considered algorithms are invariant by permutation of variables, so that this is not an issue.

Results. In all results, confidence intervals are presented for one point out of four; they are almost invisible because they are very small. Results are presented in Fig. 3.1, 3.2, 3.3, 3.4 for 100 useless variables, and in Fig. 3.5, 3.6, 3.7, 3.8 for 10000 useless variables.

Roughly speaking, many algorithms can compete for dimension 140 (codimension 40, 100 useless variables), though the simple $(1+1)$-ES and DE perform best overall (recall that we consider time on the x-axis, and not the number of evaluations). With 10000 useless variables, only fast algorithms (DE/SA/SAiso) can compete; DE performs best in case of ill-conditioning; SA performs well in case of ill-conditioning and no rotation. Algorithms which are not presented in the comparison are those who could not provide results in the given time limit.

3.5 Conclusion

This chapter emphasizes useless variables as a key for understanding the practical behavior of evolutionary algorithms on high dimensional problems. On the theoretical side, we extend known runtime analysis from the case of a set of optima with dimension 0 to a set of optima with dimension > 0 for quadratic problems with full rank, leading to a codimension m possibly much lower than the dimension d. The lower bound extends the known lower bound, from $dimension = codimension$ to more general cases. The upper bound holds for permutation of coordinates and not for the whole family of rotations (Theorem 3), or, in the case of full rotations, with a quadratic dependency in the log-precision (Theorem 2).

Practically speaking, whereas many methods rely on a linear number of function evaluations (typically just for the initialization), evolutionary algorithms use a logarithmic or constant initial population size.

In addition, an algorithm such as DE or SA or SAaniso or the simple $(1+1)$-ES will just ignore unimportant variables and optimize the remaining ones.

Therefore, evolutionary algorithms can handle very large problems, provided that the problem has a a special structure - in particular, when many variables are useless; and this is far from being trivial as some state of the art optimization methods such as Newuoa, CMAES or CMSA can not do that. In fact, a more general case might be true - when, up

to a rotation, many variables are useless; in particular, DE is invariant by rotation when cross-over is disabled (*i.e.*Cr=0), and (1+1)-ES is invariant by rotation, so that rotations of problems with many useless variables can be tackled. Importantly, rotations of problems with useless variables are not problems with useless variables - therefore, our results show that some high-dimensional problems can be tackled whenever they have no useless variables, but are rotations of problems with useless variables.

Experimentally, we successfully optimized BBOB functions with up to a million of useless variables. Unsurprisingly, for algorithms which are invariant w.r.t. useless variables, the best fitness for a given number of evaluations is exactly the same as with no useless variables. On the other hand, results become worse for algorithms which do not have this invariance and can even become impossible to obtain in a timely fashion due to computation time constraints.

Further work. (a) On the mathematical side, we conjecture that Theorem 3 also holds with F_m instead of F''_m, *i.e.*with full rotations and not only with permutations of coordinates.

(b) On the experimental side, we might study the same question empirically: what happens with random rotations of the BBOB testbed embedded in a large set of useless coordinates. For algorithms which are invariant per rotation (not DE, not PSO) this does not make any difference.

(c) Adaptive methods for choosing parameters might be tested for PSO or DE [Yu and Zhang, 2011, Liu and Lampinen, 2005, Poaík and Klema, 2012, Brest et al., 2006] as they could maybe handle better the extreme size of our problems.

(d) We tested the addition of completely useless variables. In fact, since full separability and fully rotated problems are extreme cases, we might consider variables with very low but not zero impact. We might use tricks similar to those used in the Cute testbed for partial separability [Gould et al., 2003].

(e) Recently, an effort has been made for developing real world test functions in the evolutionary computation community [Gallagher, 2016]. This provides an example of test case in which the real world decided the level of separability and the level of useless variables in a test case. Extending [Gallagher, 2016] to a high dimension case might be a good experiment.

Chapter 4

Large scale ill conditioned functions: when criteria change the whole picture

4.1 Introduction: optimization for artificial intelligence applications

When complex simulators are involved in weather or power systems for example, the gradient is not necessarily available. To overcome this difficulty, we then have the choice between reducing the number of parameters, and/or finding tools for high-dimensional black-box optimization. These problems may be partially separable and/or ill-conditioned. We will here focus on ill-conditioned problems.

Various algorithms, such as SA-ES, CMA-ES and CMSA-ES have been proposed to handle ill conditioning in evolutionary continuous optimization [Rechenberg, 1973b, Hansen and Ostermeier, 2003, Beyer, 2001, Beyer and Sendhoff, 2008]. We here compare some of the main evolutionary algorithms methodologies on ill conditioned functions, in the case of high-dimensional problems with partial separability. In particular, we extend the analysis in [Hansen et al., 2008, Auger et al., 2009] in the direction of higher dimensions, partial separability, and a set of different criteria..

The goal is to show that ill conditioned functions, in a partially separable world, lead to very different results depending on simple things: criteria (often overlooked), partial separability (there is an entire world between full separability and complete rotational invariance), taking into account the computational cost or not.

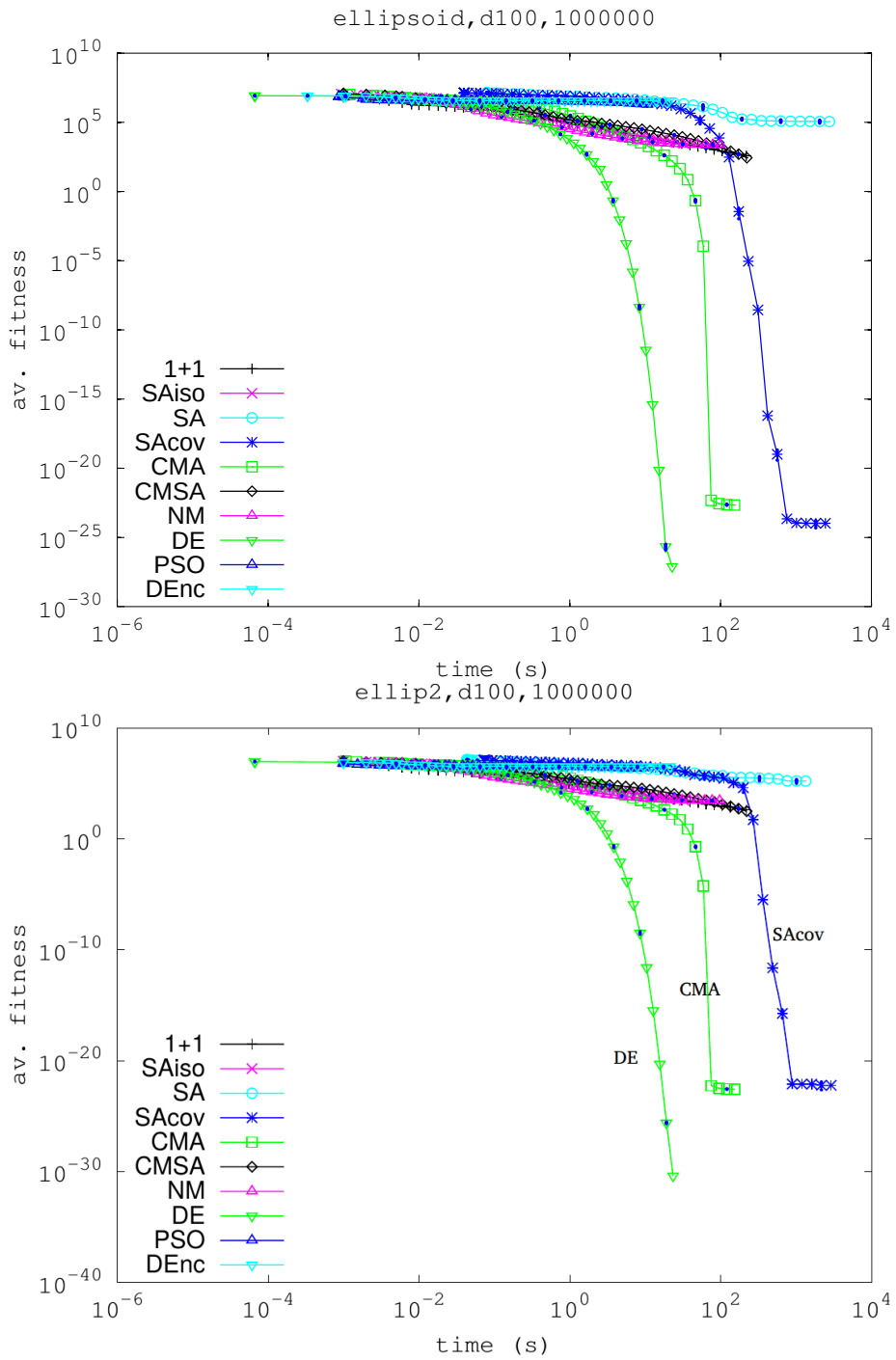


Figure 4.1 – Ellipsoid function (not rotated) in dimension 100 (top) and slightly rotated ellipsoid function in dimension 100 (bottom). Algorithms good for ill-conditioned functions perform well, including those working with full covariance such as CMA-ES and CMSA-ES. There is not much rotation, hence the good performance of anisotropic SA and PSO which can tackle rescaling of variables. Overall, DE performs best.

4.2 Experiments

Second order approximations of difficult smooth functions, quadratic ill conditioned functions have been widely used as a testbed. This is not a violation of the non-linear nature of the test, because our tests are based on comparison-based algorithms; therefore, a wide family of non-quadratic functions are handled as well - all functions with same-center ellipsoids as level sets are concerned. For example, [Auger et al., 2009] considers ellipsoid functions raised to some exponent, which is deceptive for Newuoa [Powell, 2008] due to its usage of fitness values (and not only comparisons), but does not disturb our algorithms here.

In our experiments, in order to have both ill-conditioning and variable levels of separability, we used some functions and three optional rotation schemes.

4.2.1 Test functions

The functions we used in our experiments are the ellipsoid function f_e, the cigar function f_c and the Schwefel function such that:

$$f_e(\mathbf{x}) = \sum_{i=1}^{N} \alpha^{\frac{i-1}{N-1}} \mathbf{x}_i^2$$

$$f_c(\mathbf{x}) = (\mathbf{x}_1^2) + 10^6 \sum_{i=2}^{N} \mathbf{x}_i^2$$

$$Schwefel(\mathbf{x}) = \sum_{i=1}^{N} \sum_{j=1}^{i} x_j^2$$

In the general case for f_e we have $\alpha = 10^6$, which gives us an ill-conditioned function, and the $Schwefel$ function is naturally ill-conditioned in high dimension. We also use the special case of f_e with $\alpha = 1$, which is the sphere function $f_s(\mathbf{x}) = \|\mathbf{x}\|_2^2$. This allows us to have a baseline, not ill-conditioned. For all algorithms, each coordinate of each initial individual in the population is randomly drawn according to a Gaussian random variable with zero mean and standard deviation 1, on a domain of $[-10, 10]^D$

4.2.2 Rotations

To obtain different levels of separability, we considered three different rotation schemes, plus of course no rotations at all. The first and simplest one is the pair-wise rotation, where

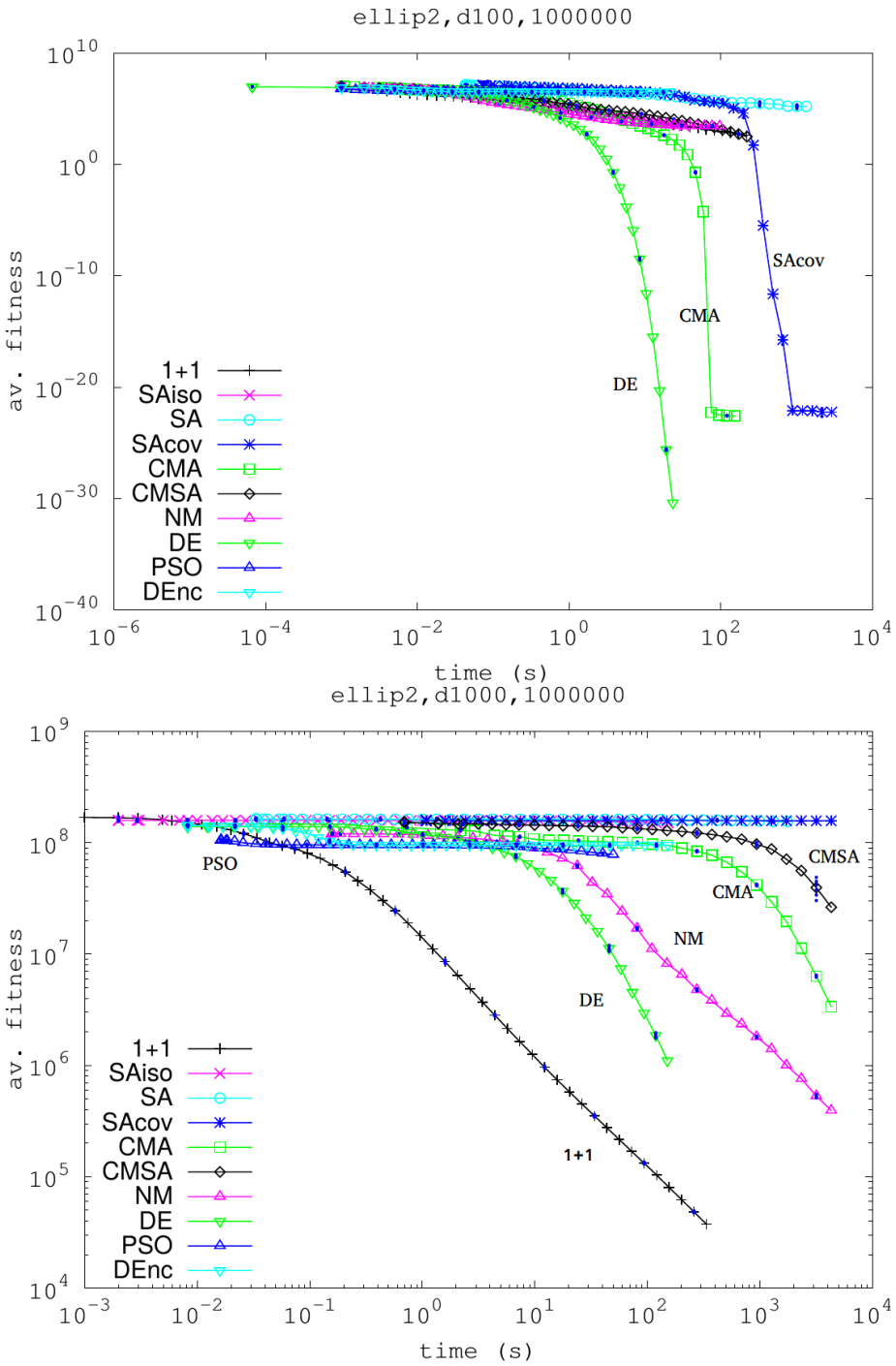


Figure 4.2 – Slightly rotated ellipsoid function in dimension 100, 1000. The best algorithm depends on the budget and dimension; $(1+1)$-ES is surprisingly strong, DEnc, DE, PSO, also perform well. Standard deviations are plotted but so small they are barely visible.

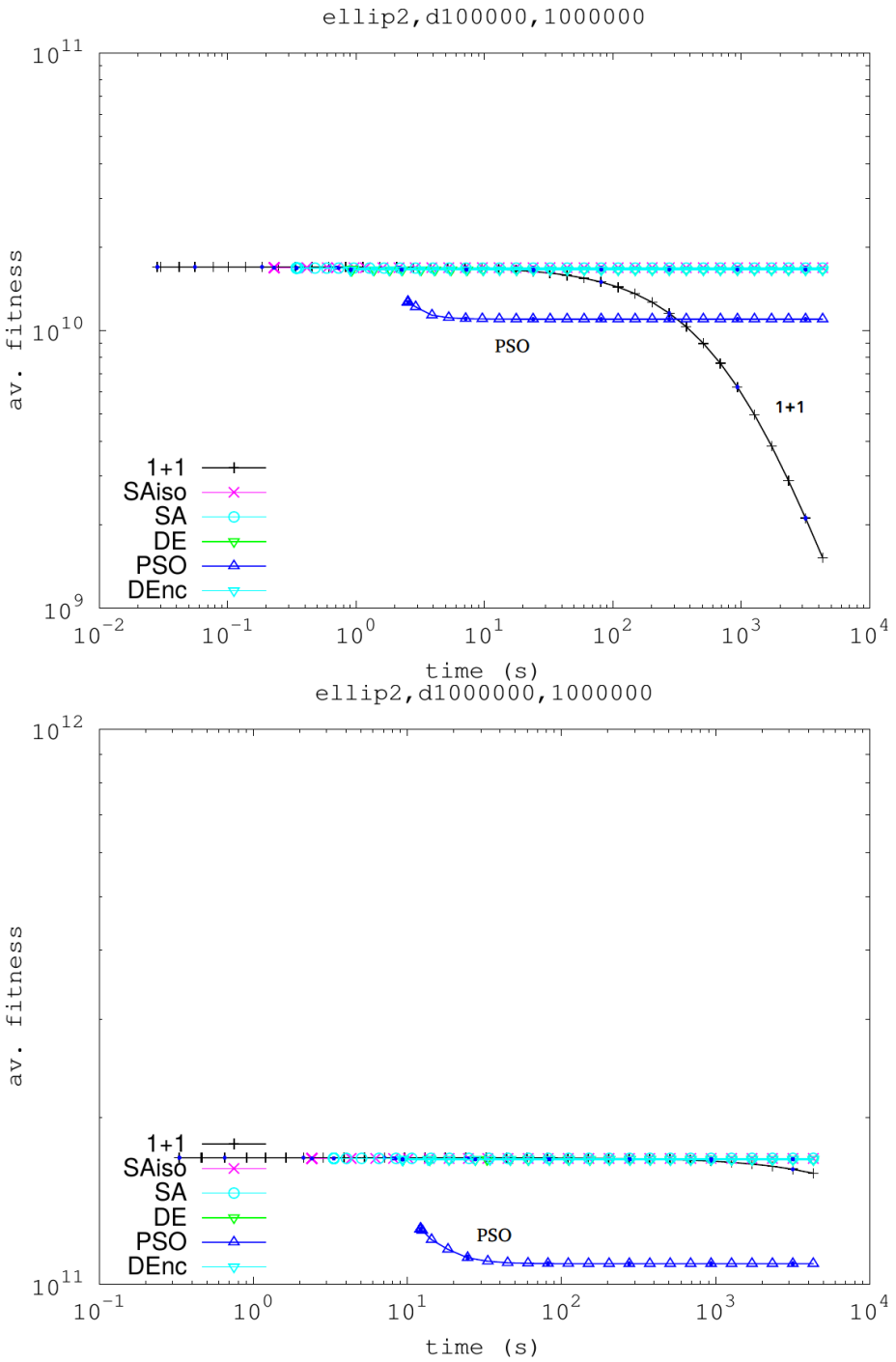


Figure 4.3 – Slightly rotated ellipsoid function in dimension 10000, 100000. The best algorithm depends on the budget and dimension; $(1+1)$-ES is surprisingly strong, DEnc, DE, PSO, also perform well. Standard deviations are plotted but so small they are barely visible.

each pair of two axes is rotated by a 45 degrees angle such that $f_2 = f(R(\mathbf{x}))$ where

$$R = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & \dots & 0 & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & \dots & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 0 & 0 & \dots & 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}.$$

This type of rotation, by just mixing two variables together, gives a low level of non-separability. It is quite fast, and doesn't use any matrix whatsoever, allowing us to use it at any desired dimension.

A more complex rotation, called sparse rotation, is a generalization to random angles of the pair-wise rotation: for $i \in \{1, \cdots, \frac{N}{2}\}$, given P_i a randomly drawn permutation of indices, R_i is the composition of randomly drawn rotations for each pair of axes (*i.e.*1 and 2, 3 and 3, *etc.*), we define $f_{sparse} = f \circ R_1 \circ P_1 \circ \cdots \circ R_{\frac{N}{2}} \circ P_{\frac{N}{2}}$. Each variable i is therefore mixed with approximately $2^{\lceil \sqrt{log(N)} \rceil}$ other variables.

Finally, we have the full rotation, where $f_R = f(R_f \mathbf{x})$, with R_f being uniformly drawn in the rotations space. This gives us a fully non-separable problem.

By applying each of those rotations to the different functions, we are able to define functions with different levels of separability and ill-condition (*e.g.* $ellipsoid_2, cigar_{sparse}, ellipsoid_R$, *etc.*)

4.2.3 Invariances

In the white-box case, using the Hessian and gradient, the Newton algorithm is invariant per translation, rotation, rescaling of variables, except for the initialization. BFGS [Broyden, 1970, Fletcher, 1970, Goldfarb, 1970, Shanno, 1970] has similar properties with only the gradient, and NEWUOA [Powell, 2008] has similar properties with gradient and Hessian approximated thanks to successive objective function values. We now discuss invariances for comparison-based optimization algorithms, which are also invariant for compositions with monotonic functions.

It is well known that PSO is not invariant for rotations [Hansen et al., 2008]. But in the present chapter, we do not consider only full rotation. Full rotation is extremely costly and fully destroys separability; it is nonetheless an interesting testbed, but the point is the

present chapter is to investigate what is going on with moderate rotations of variables, *i.e.*partial separability.

DE is invariant or not, depending on the choice of the Cr parameter. [Auger et al., 2009] concluded that for their test case, $Cr = 1$ (full invariance) was the best choice. We will include this in our test, but due to partial separability, it makes sense to also keep $Cr < 1$ in the comparison.

[Auger et al., 2009] compared algorithms from the point of view of a limited dimension, high ill conditioning, full rotation, and neglecting the internal computation time of algorithms - only numbers of evaluations are considered there. In contrast, we compare the performance of various algorithms in high-dimension, with partial separability and taking into account the internal computation time of algorithms, by comparing the performances of the different algorithms not only based on the number of evaluations performed, but also by the elapsed time.

4.2.4 Experimental results

Numerical results

The ellipsoid function. We first consider the ellipsoid function (Fig. 4.1). In dimension 100, for small budget, DE performed best; with higher budget, PSO becomes better; then it is outperformed by Nelder-Mead, later joined by CMSA-ES and more surprisingly $(1+1)$-ES. In higher dimension, PSO and DE get the best performances.

The slightly rotated ellipsoid function. We now slightly rotate the ellipsoid function, function f_{2r}; Fig. 4.2 and4.3. In dimension 100, the best results for small budgets are obtained by DE and PSO, but they are quickly outperformed by Nelder-Mead and then SA with isotropic mutations, and later joined by CMSA-ES and the simple $(1+1)$. In high-dimension (1000 and 1000000) PSO outperforms all other algorithms by far.

The sparsely rotated ellipsoid function. We now consider the sparsely rotated ellipsoid function (Fig. 4.11 and4.11). PSO, DE and Nelder-Mead perform well in moderate dimension. In high-dimension, PSO still performs well, but with larger budget it is clearly outperformed by DE, the only algorithm which managed to benefit from second order information in spite of the large dimension.

An interesting fact is that $(1+1)$-ES comes back as the best algorithm, in spite of its simplicity; $(1+1)$-ES is not invariant by rescaling of variables, but it is invariant per rotation and it is extremely fast, and succeeds with this degree of rotations which is, contrarily to the “slightly” rotated ellipsoid, too hard for most algorithms.

The fully rotated ellipsoid function. We now fully rotate the ellipsoid function, function $ellip_R$; Fig. 4.4. In dimension 100, the best results for small budgets are obtained by DE and PSO, but they are quickly outperformed by Nelder-Mead and then SA with isotropic mutations, and later joined by CMSA-ES and the simple $(1+1)$. In high-dimension (1000 and 1000000) PSO outperforms all other algorithms by far.

Schwefel function. Fig. 4.5 and4.6 presents results for the Schwefel function in various dimensions; NM performs best in small dimension, and then it is outperformed by CMA-ES (dimension 100); in high dimension $(1+1)$-ES is the best algorithm but PSO, DE, DEnc and SAcov have interesting results and SA-iso is good for a fast approximation - but eventually fails probably because of isotropy.

Cigar function. Finally, Fig. 4.7 and4.8 presents results for the Cigar function with various rotations, in dimension 1000.

Results are then reproduced in dimension 10000. The $(1+1)$-ES performs remarkably well.

Criteria

Various criteria can be used when considering a minimization problem:

- A simple criterion, when an algorithm output an approximate $\hat{x}$ of the optimum x^* of an objective function f, is the simple regret $f(\hat{x}) - f(x^*)$. This can be averaged over multiple runs.
- Another criterion is the probability of $f(\hat{x}) - f(x^*) < \varepsilon$ for some ε. The choice of ε is important and it is not invariant by rescaling of the optimization algorithm; for example, various optimization testbeds have used $\varepsilon << 1$ (such as $\varepsilon = 10^{-9}$) for functions with values of order 10^7; this is a very asymptotic behavior, and sensitive to machine precision.
- The previous criterion can be adapted (SP1, Expected Running Time - ERT, both in [Auger and Hansen, 2005]) without deeply modifying the nature of results.
- Less common criteria include $||\hat{x} - x^*||$.

Besides criteria, important experimental conditions include the dimension, the tuning of the algorithms under analysis, the translation or rotation of the objective function.

In this work, we will focus on the first simple criterion: average regret as a function of time.

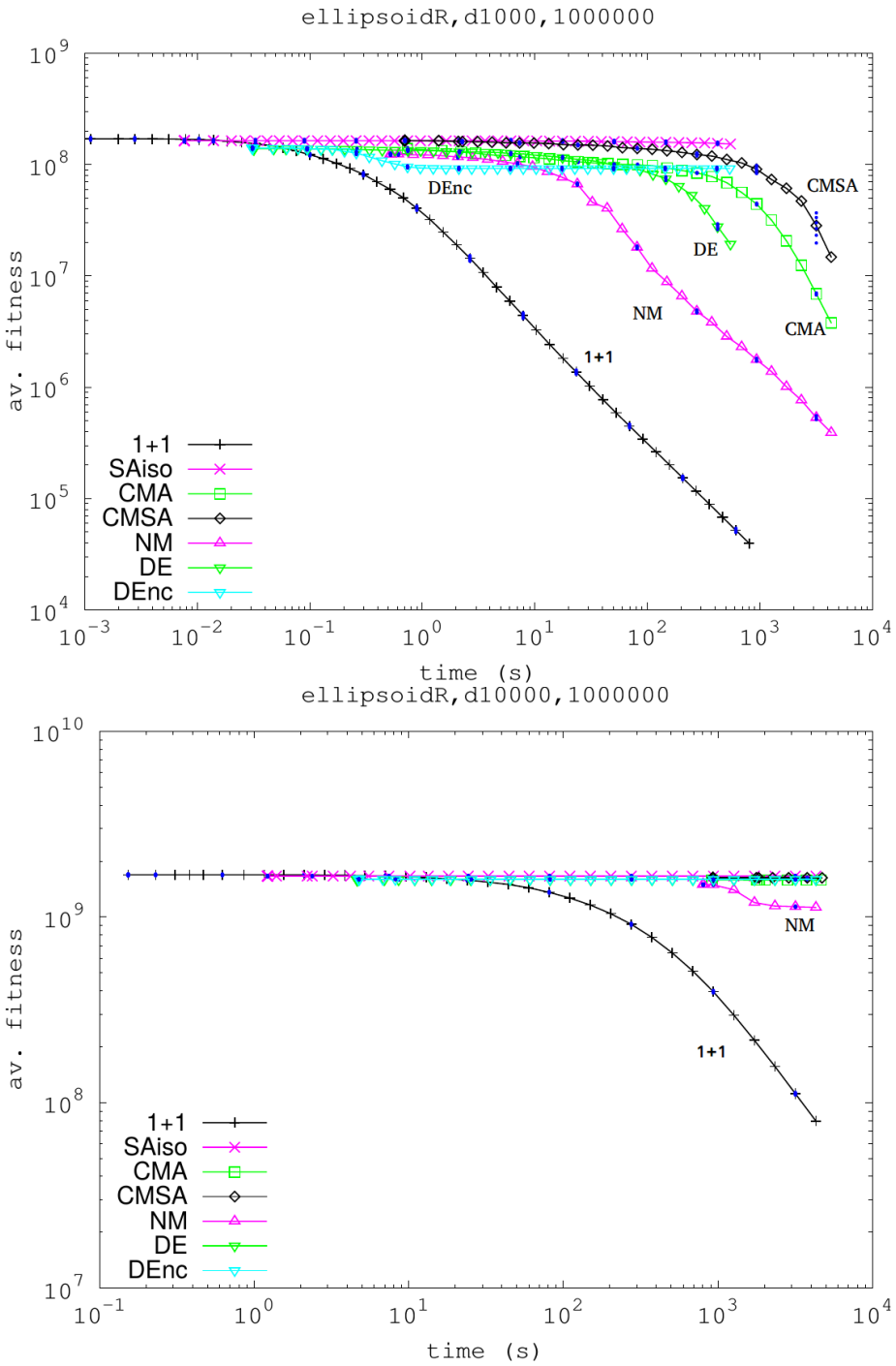


Figure 4.4 – Fully rotated ellipsoid function in dimension 1000 and 10000. Somehow surprisingly, the simple $(1+1)$ performs best, though in small dimension CMA-ES might catch up. In high dimension, sophisticated algorithm pay the price of their computational complexity - contrarily to many published chapters, we take into account computation time. Standard deviations are plotted but so small they are barely visible.

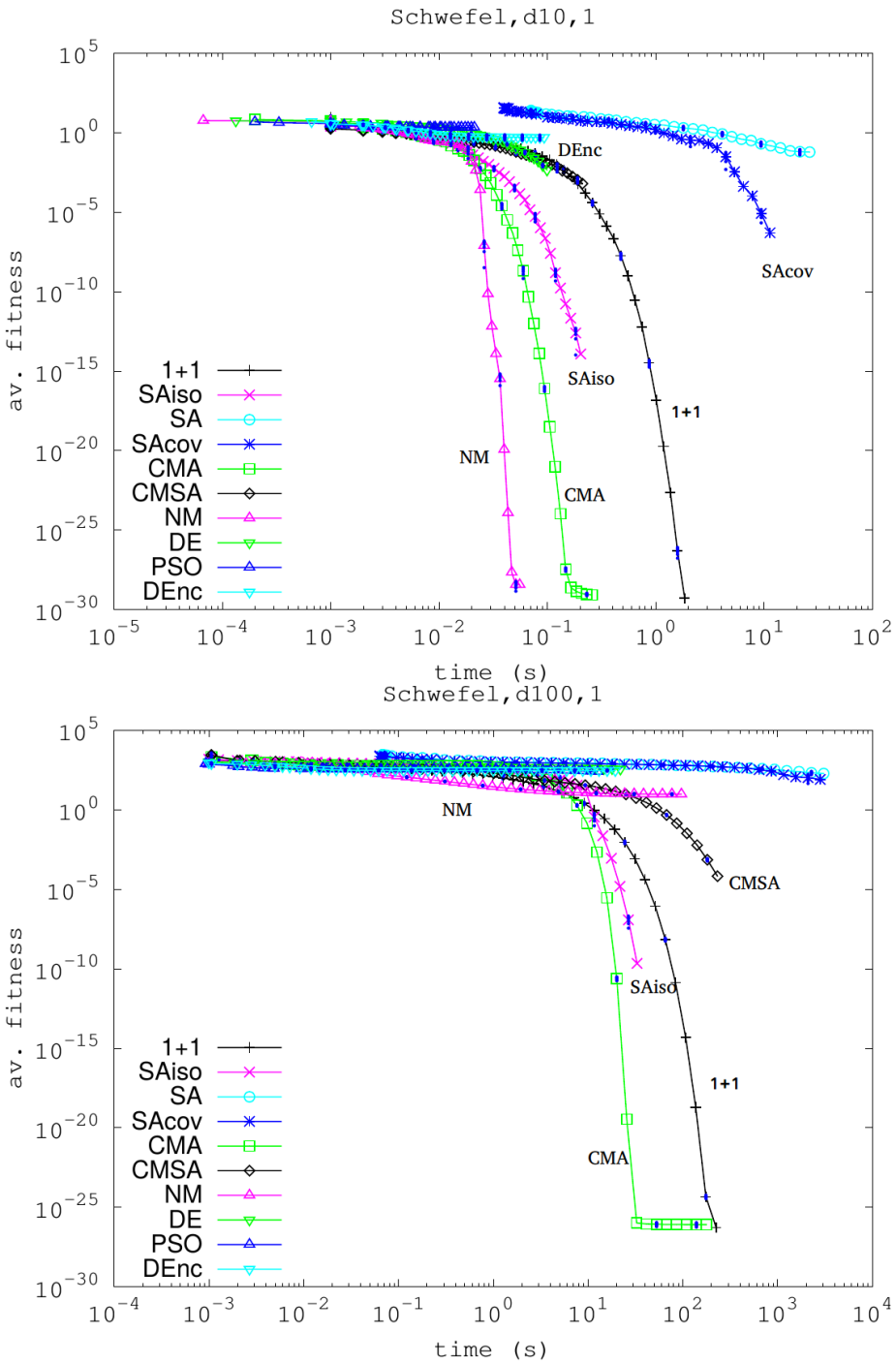


Figure 4.5 – Schwefel function in dimension 10, 100. Somehow surprisingly, the simple $(1+1)$ performs best, though in small dimension NM is best and CMA-ES might catch up. In high dimension, sophisticated algorithm pay the price of their computational complexity (contrarily to many published chapters, we take into account computation time). Standard deviations are plotted but so small they are barely visible.

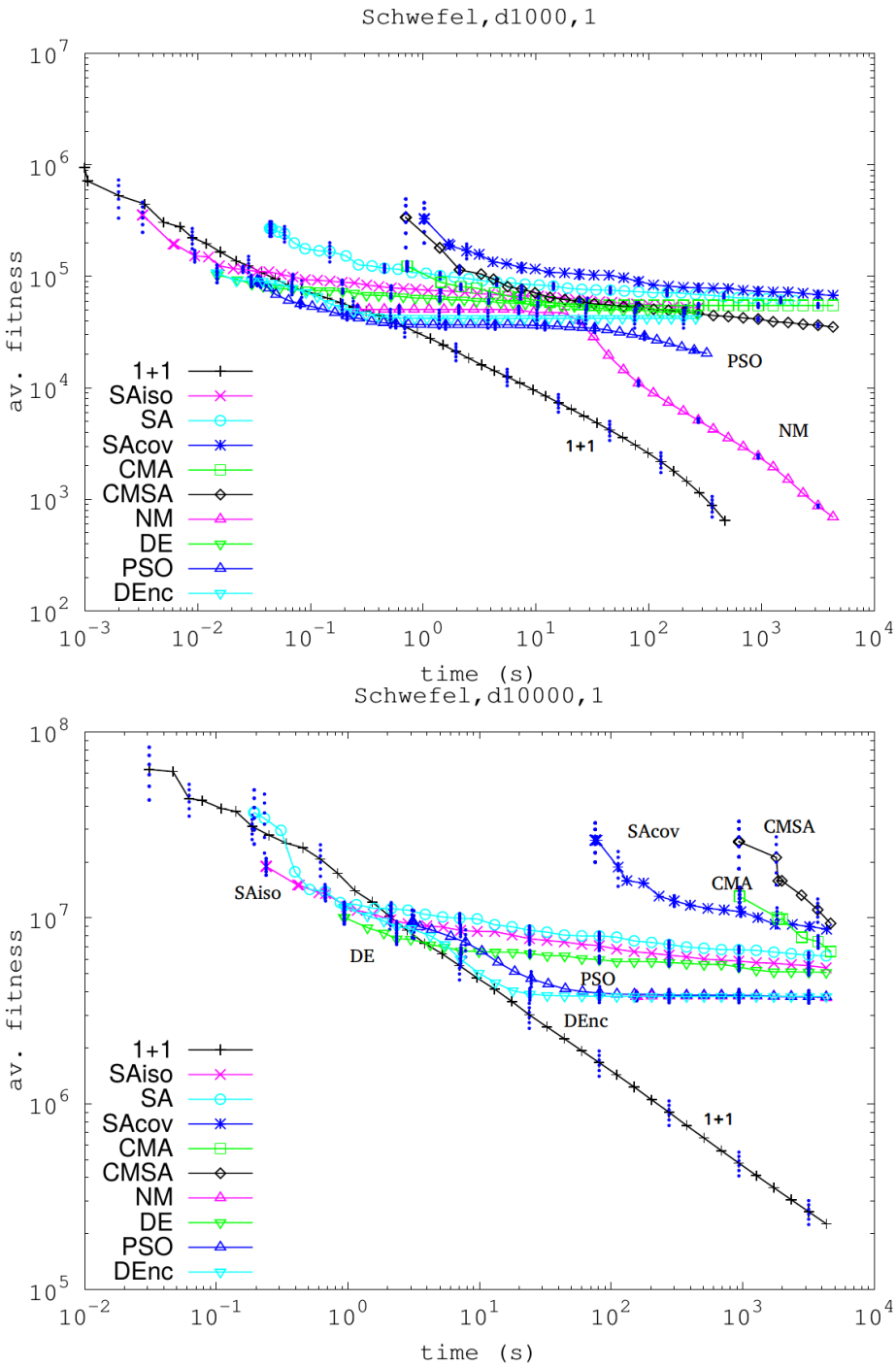


Figure 4.6 – Schwefel function in dimension 1000 and 10000. Somehow surprisingly, the simple $(1+1)$ performs best, though in small dimension NM is best and CMA-ES might catch up. In high dimension, sophisticated algorithm pay the price of their computational complexity (contrarily to many published chapters, we take into account computation time). Standard deviations are plotted but so small they are barely visible.

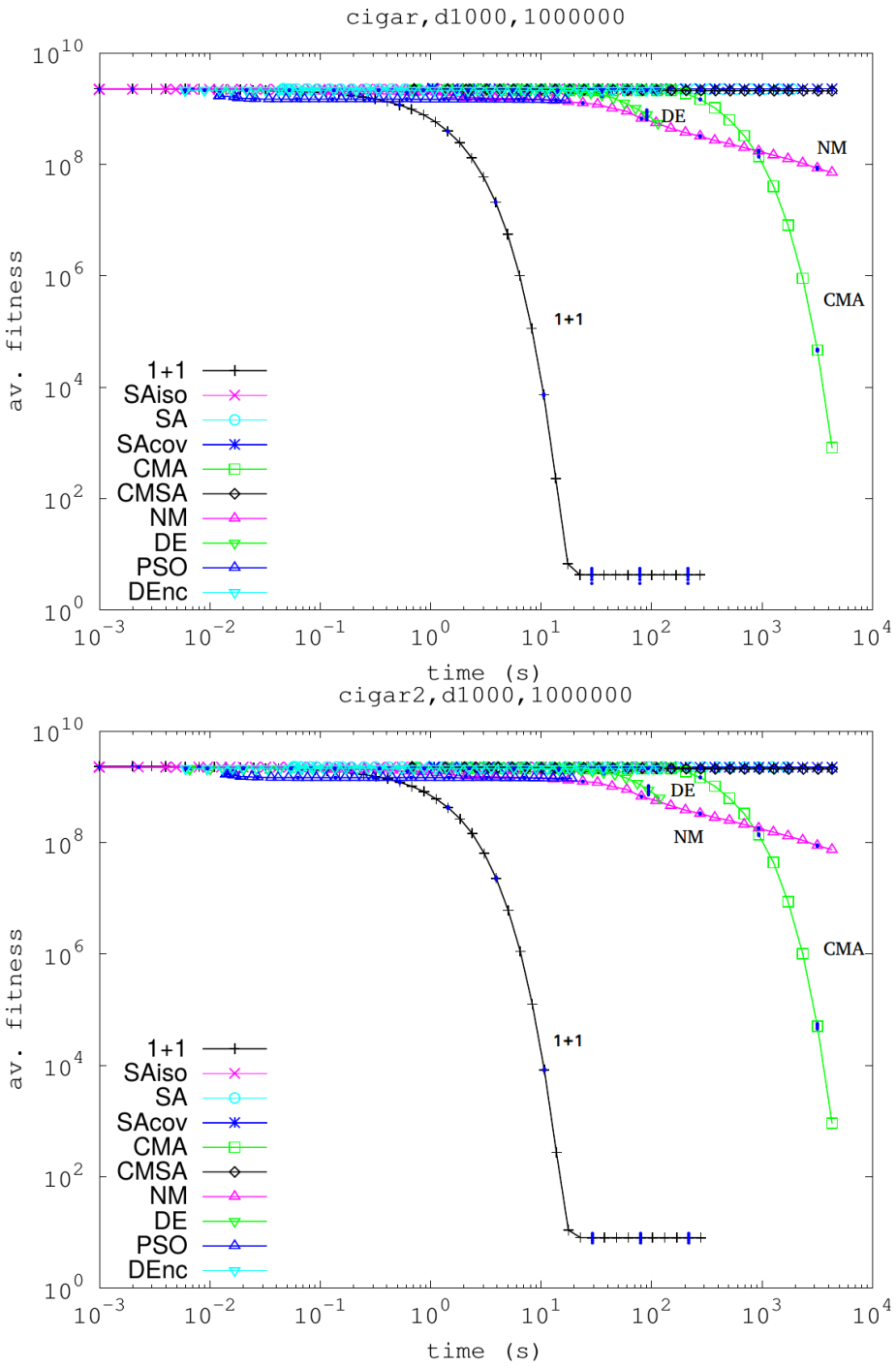


Figure 4.7 – Cigar function, in dimension 1000, with no rotation (top) and pair-wise rotation (bottom). DE becomes better and better with less rotations. Still, $(1+1)$-ES performs best. Standard deviations are plotted but so small they are barely visible.

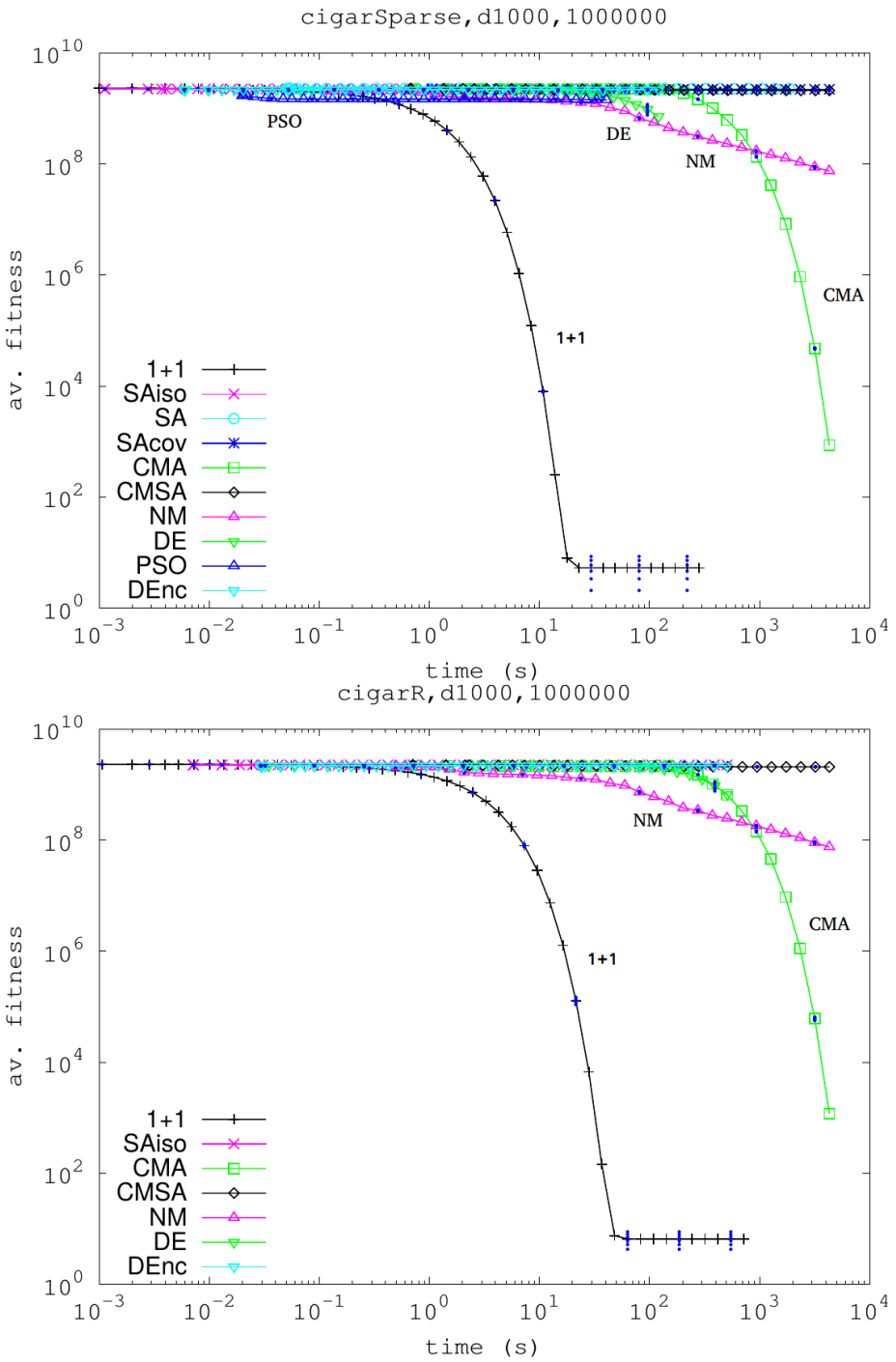


Figure 4.8 – Cigar function, in dimension 1000, with sparse rotation (top) and full rotation (bottom). DE becomes better and better with less rotations. Still, $(1+1)$-ES performs best. Standard deviations are plotted but so small they are barely visible.

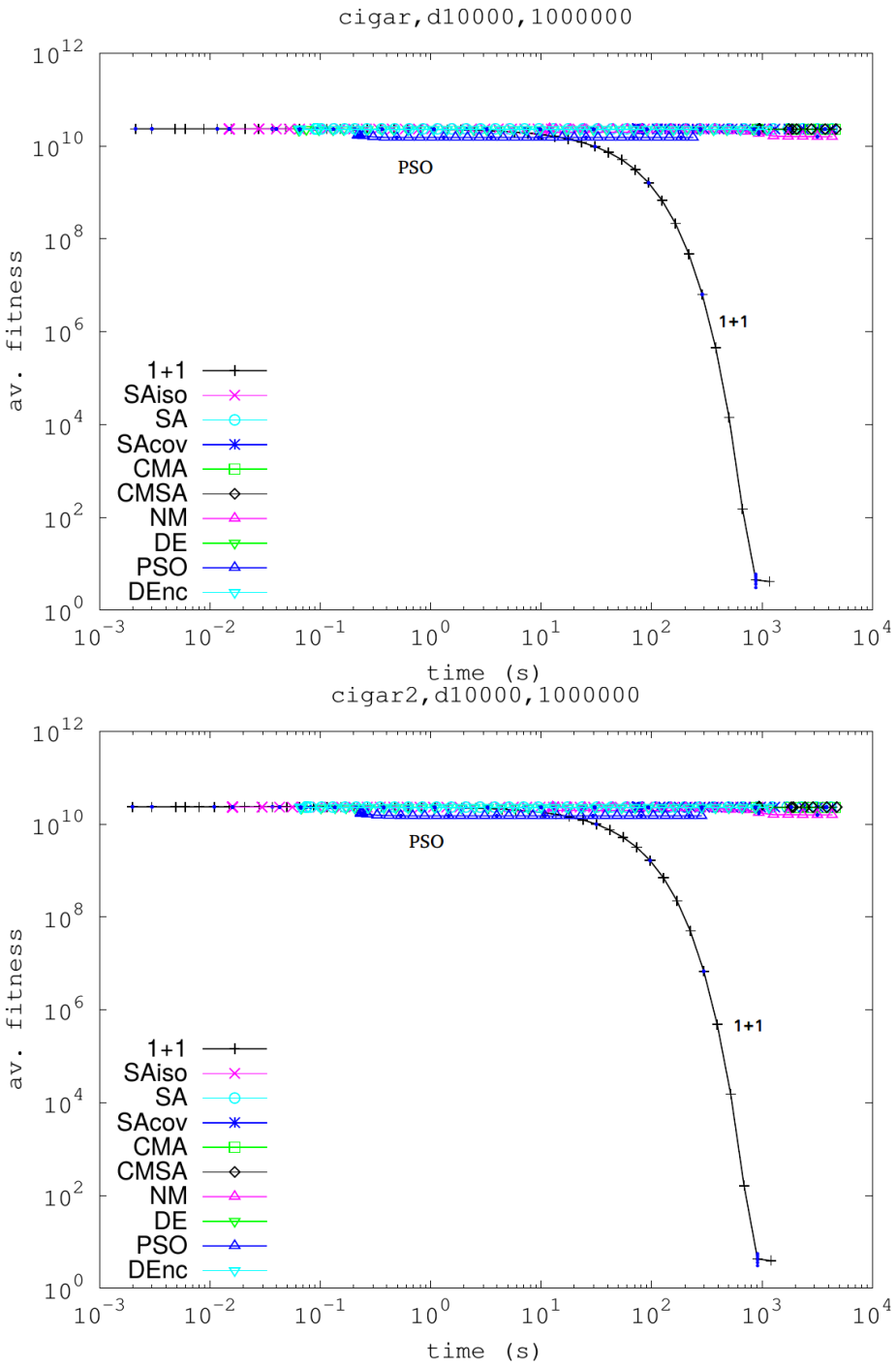


Figure 4.9 – Cigar function, in dimension 10000, with no rotation (top) and pair-wise rotation (bottom). The simple $(1+1)$-ES is the best performing algorithm; except for small budget where PSO performs well. Standard deviations are plotted but so small they are barely visible.

t

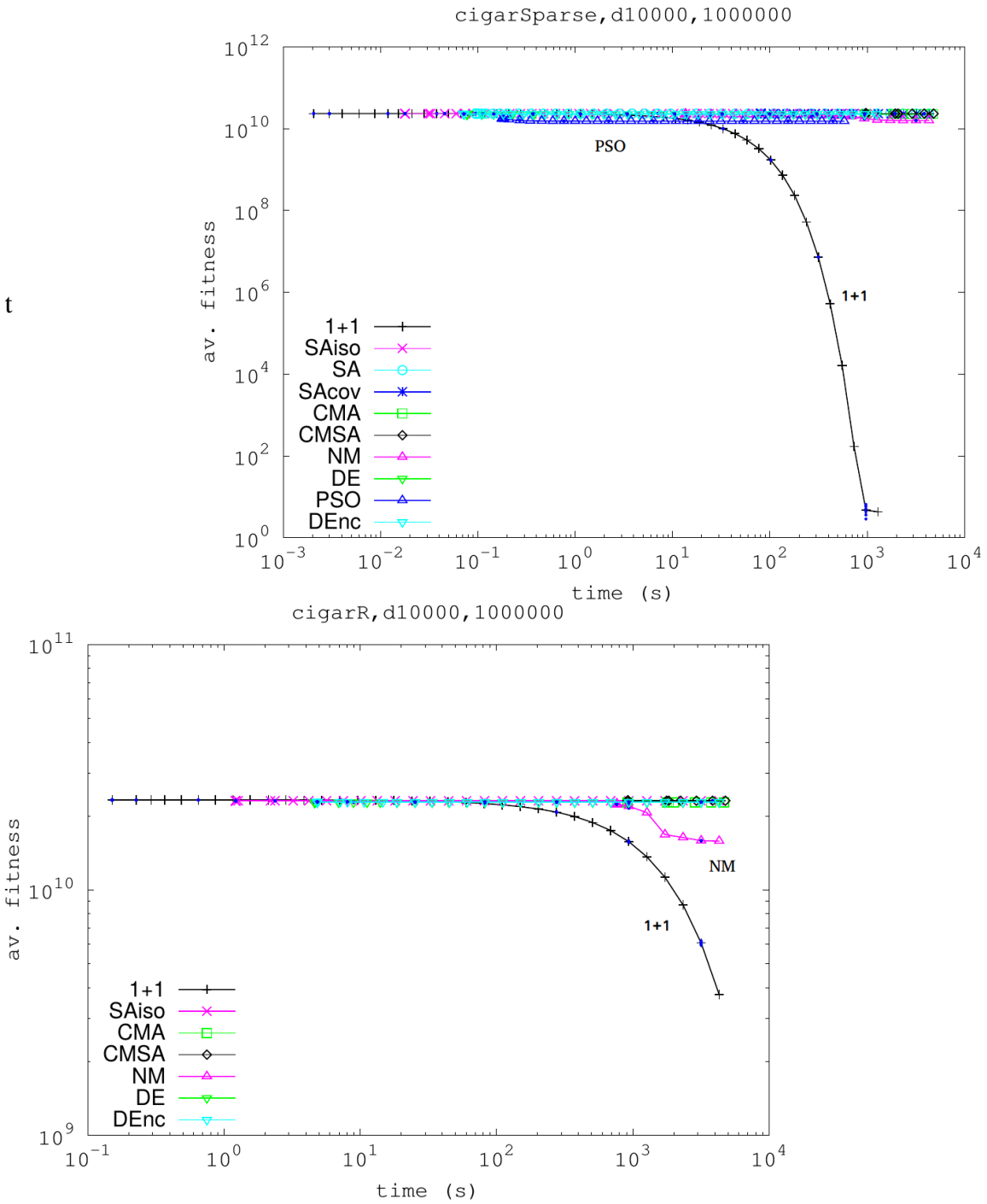


Figure 4.10 – Cigar function, in dimension 10000, with sparse rotation (top) and full rotation (bottom). The simple $(1+1)$-ES is the best performing algorithm; except for small budget where PSO performs well. Standard deviations are plotted but so small they are barely visible.

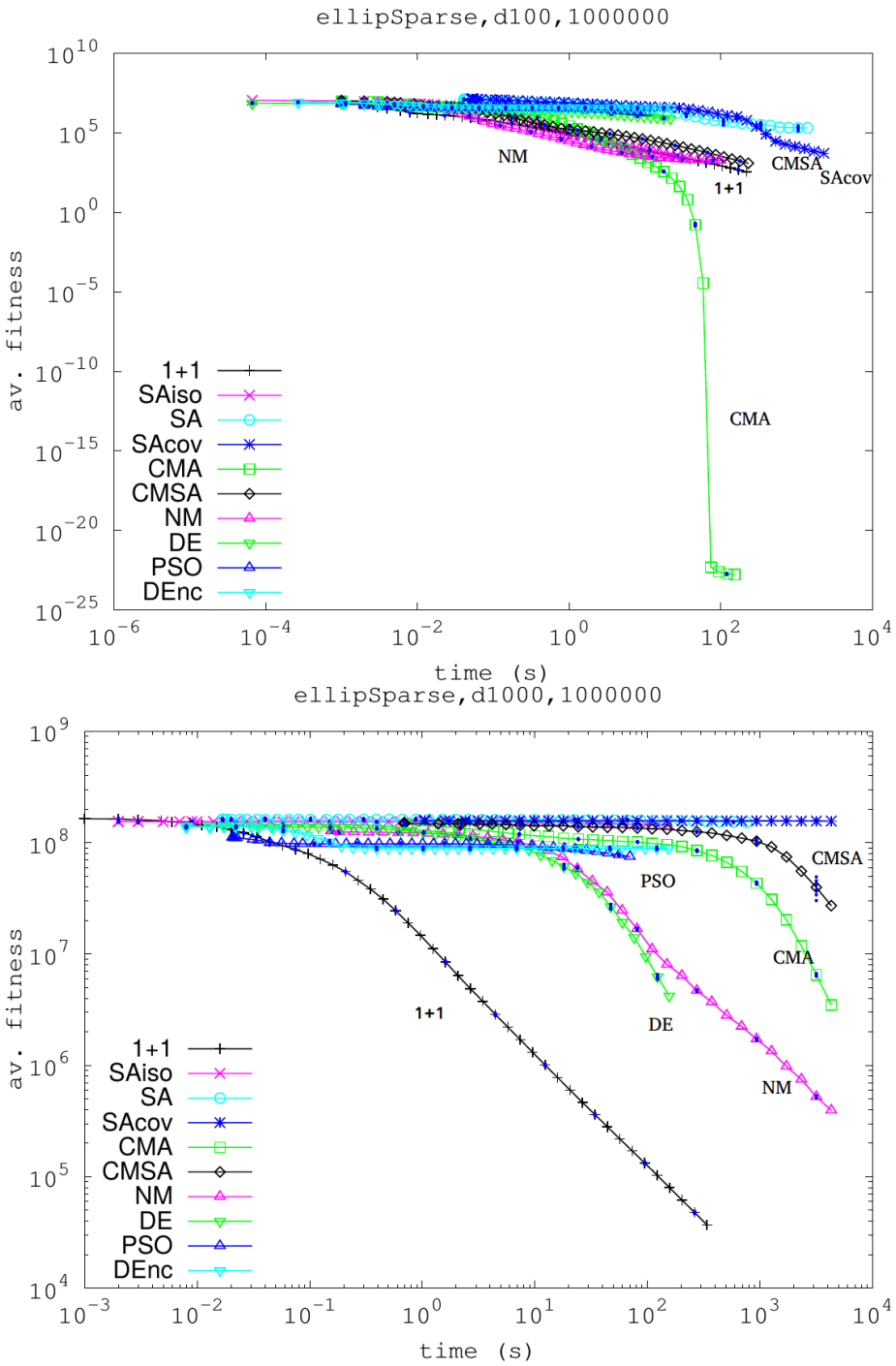


Figure 4.11 – Sparsely rotated ellipsoid function in dimension 100, 1000. CMA-ES performs very well in small dimension. Standard deviations are plotted but so small they are barely visible.

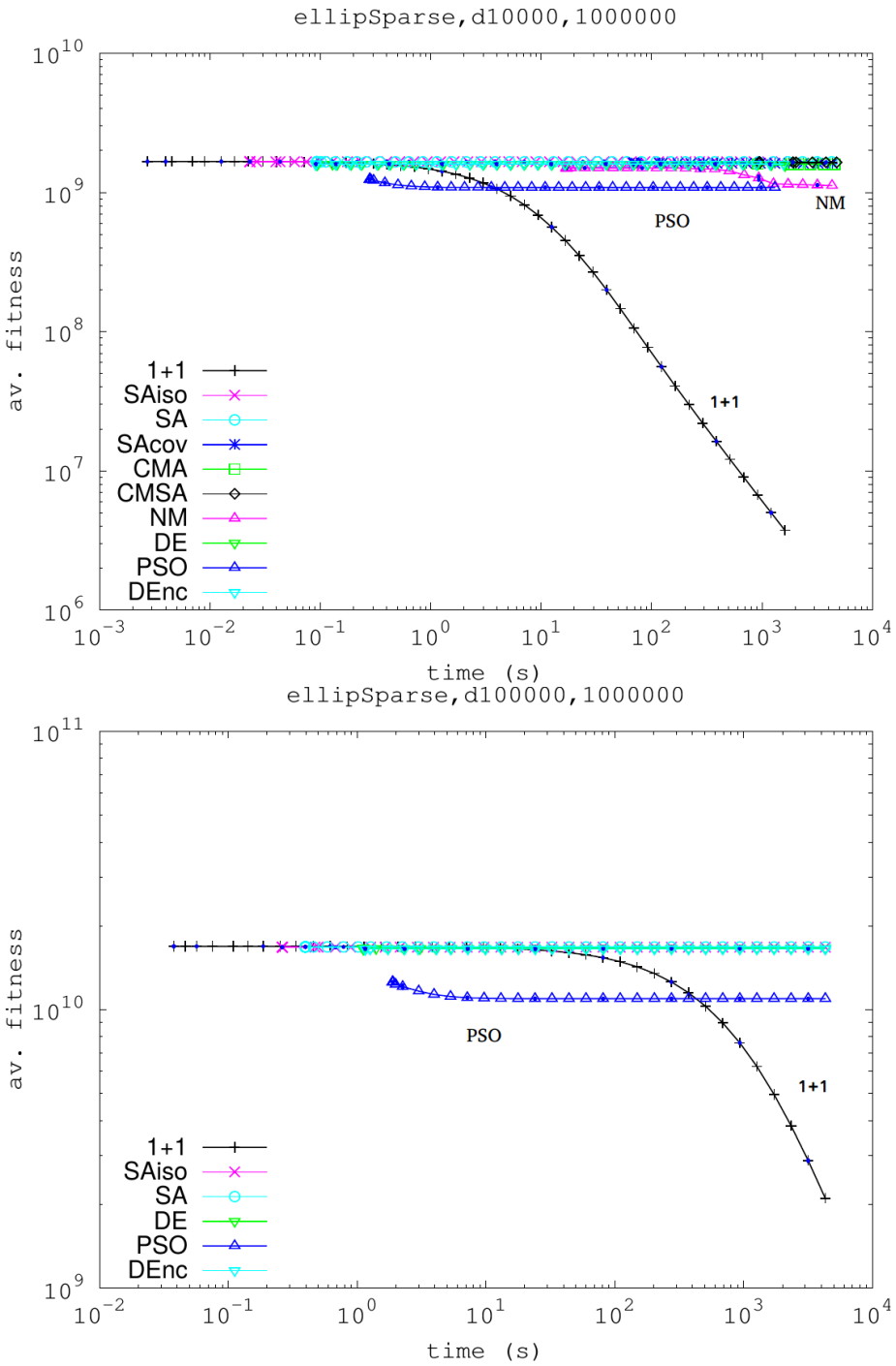


Figure 4.11 – Sparsely rotated ellipsoid function in dimension 10000, 100000. We see that in dimension 100000, $(1+1)$-ES surprisingly outperforms its competitors for sufficiently large time. Standard deviations are plotted but so small they are barely visible.

4.3 Conclusion

DE, PSO and $(1+1)$-ES performed well in many settings; PSO for small budget, DE for moderate separability, and $(1+1)$-ES as the most stable algorithm. Full covariance adaptation provides interesting results when it can be applied with a reasonable cost, *i.e.* moderate dimension. CMA-ES is excellent for handling numerical precision issues, reaching precisions that other algorithms just can't reach.

We used log scales on the y-axis; it must be pointed out that this leads to pictures in favor of algorithms which are efficient late in the runs; in many cases, PSO or DE was the best for dividing the regret (*i.e.* the difference with the optimum fitness) by 10 - this is visible only for careful readers on the graph, whereas without log the figures would emphasize this kind of quick progression towards approximate solutions.

Portfolio algorithms. This might be the most important and practical conclusion indeed. $(1+1)$-ES is computationally very fast and can work when nothing else works; full covariance algorithms such as CMA-ES and CMSA-ES can handle highly ill conditioned problems in small scale; PSO is excellent when there is enough separability; DE outperforms PSO when there is not enough separability; NM is excellent when the cost of the initialization does not make it intractable (*i.e.*when the dimension is small enough that it's possible to store the population in the RAM, and thus to apply NM on the considered problem). A portfolio [Baudis and Posik, 2014] of those five algorithms can therefore get the best of them, with just a factor five in the computation time (possibly discarded by parallelization!); such a portfolio might be the most robust and practical solution. Since all our graphs have x-axis in log-scale, a factor 5 does not change much in the results.

Some take-home messages:

- Do not overlook simple criteria such as the average obtained fitness values. Just focusing on (variants of) the probability of reaching precision $1e^{-9}$ mainly selects algorithms which are good at handling numerical problems.

- Do not overlook computation time and high dimension, which might be negligible for small scale problems but becomes a big deal in dimension 1000 and crippling in dimension 1000000, typically for algorithms which maintain a full covariance matrix.

- Do not overlook partial separability; fully rotating problems might not be a better model than full separability. Moreover, fully rotating induces huge computation times, making the internal cost of algorithms artificially negligible and pushing research towards small scale problems.

- PSO, DE and the simple (1 + 1)-ES perform well in many cases, including full rotation, when the dimension is large and the computation time is taken into account and not only the number of iterations, or when the rotation is moderate. They are weak at finding the solution up to $1e-9$ though, when the budget is large enough for allowing this.

Further work

It would be interesting to add more degrees in the sparsity of rotations / separability of functions (increasing the non separability and/or the computational cost of the objective function). A small but non-zero coupling between different groups of variables could be included [Gould et al., 2003]. Finally, quadratic forms extracted from real problems, using the Hessian at the optimum, could be used for generating real world partial separability.

Chapter 5

Experiments on the CEC 2015 expensive optimization testbed

This chapter's content comes from the paper Berthier, V. (2015b). Experiments on the cec 2015 expensive optimization testbed. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 1059–1066. IEEE, a paper presented as an entry to the Expensive Competition. Its abstract was:

We experiment various simple classical algorithms on the expensive optimization testbed from Cec2015. CMA performs best, in particular its DCMA flavor using quasi-random numbers. Nelder-Mead also performs well. Portfolios performed well for the given budget (500 evaluations in dimension 10 and 1500 evaluations in dimension 30), but not the half budget, which is also taken into account in the competition, hence we did not include them in the final version.

5.1 State of the Art

[Chen et al., 2014] proposed a testbed for expensive optimization. This means that the number of fitness evaluations is limited. In this case the budget is 500 in dimension 10 and 1500 in dimension 30. The test-case is also designed for difficult objective functions, with 2 unimodal and 13 multimodal functions including hybrid and composition functions.

[Baudis and Posik, 2014] proposed the use of portfolio methods in noise-free optimization. Portfolio are generic tools for combining optimization algorithms, most widely used in combinatorial optimization, but also appearing in noisy continuous optimization[Cauwet et al., 2014] and noise-free continuous optimization[Baudis and Posik, 2014].

We also use quasi-random numbers in mutations, as proposed in

[Teytaud and Gelly, 2007]. Basically, it makes algorithms slightly better on average, with a different distribution, as detailed later.

The computation time in portfolio algorithms can be simply divided equally among solvers, or not[Pulina and Tacchella, 2009]. [Gagliolo and Schmidhuber, 2005, Gagliolo and Schmidhuber, 2006] propose 50% for the best solver, 25% for the second best, and so on. One might also run all solvers during *e.g.* 25% of total time, and then keep 75% of the budget only for the best performing one. This is the approach we keep for all our portfolio experiments in this chapter.

Surrogate models are classical for expensive optimization[El-Beltagy et al., 1999]; population-based methods have also been widely used[Poloni and Pediroda, 1997], as well as derivative-free methods as expensive optimization is often due to heavy simulations without gradient[Booker et al., 1999]. Gaussian processes are also widely used as their internal cost becomes negligible in front of the cost of the objective function[Jones et al., 1998, Villemonteix et al., 2009]. We consider mostly population-based optimization; the comparison with the results of other methods will be outputs of the session, comparing various methods on this same testbed.

5.2 Restart / Portfolio

[Baudis and Posik, 2014] reported excellent results for sophisticated methods, and indeed also good stable results with simple methods. We decided to focus on simple methods. In all our experiments, the portfolio equally divides the computation time among the algorithms during 25% of the budget, and then uses only the solver which provided the best search point during the remaining 75% of the budget. CMA/CMA/CMA for example refer to running 3 instances of CMA during 25% of the budget, and then the best performing one during the remaining 75%.

5.3 Experiments

In the following sections, we first compare several classical algorithms, before tuning the best one and adding quasi-random numbers.

The initialization of the population of each algorithm is uniform in the domain $[-100, 100]^D$, for each algorithm.

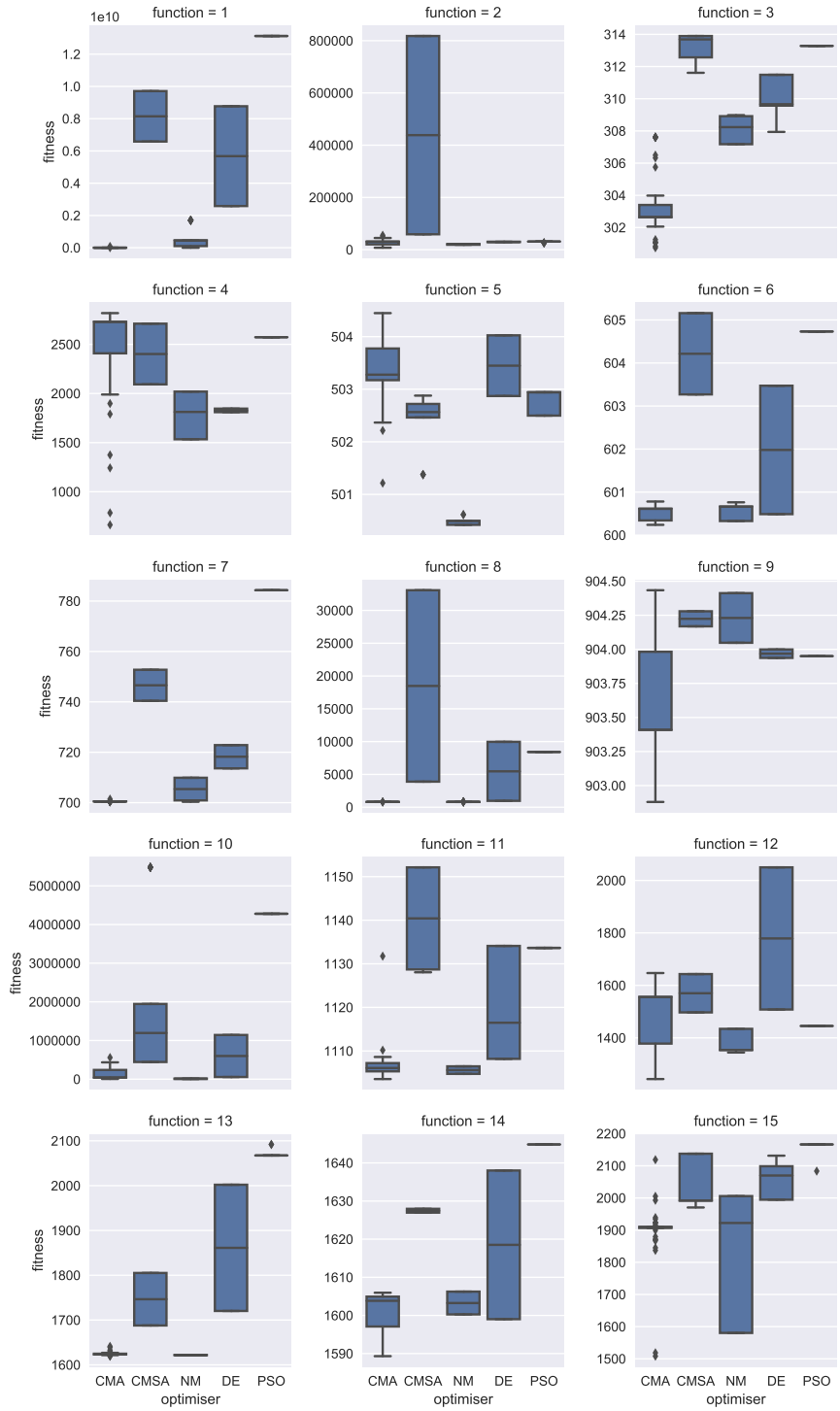


Figure 5.1 – Optimizers results on the expensive benchmark in dimension 10

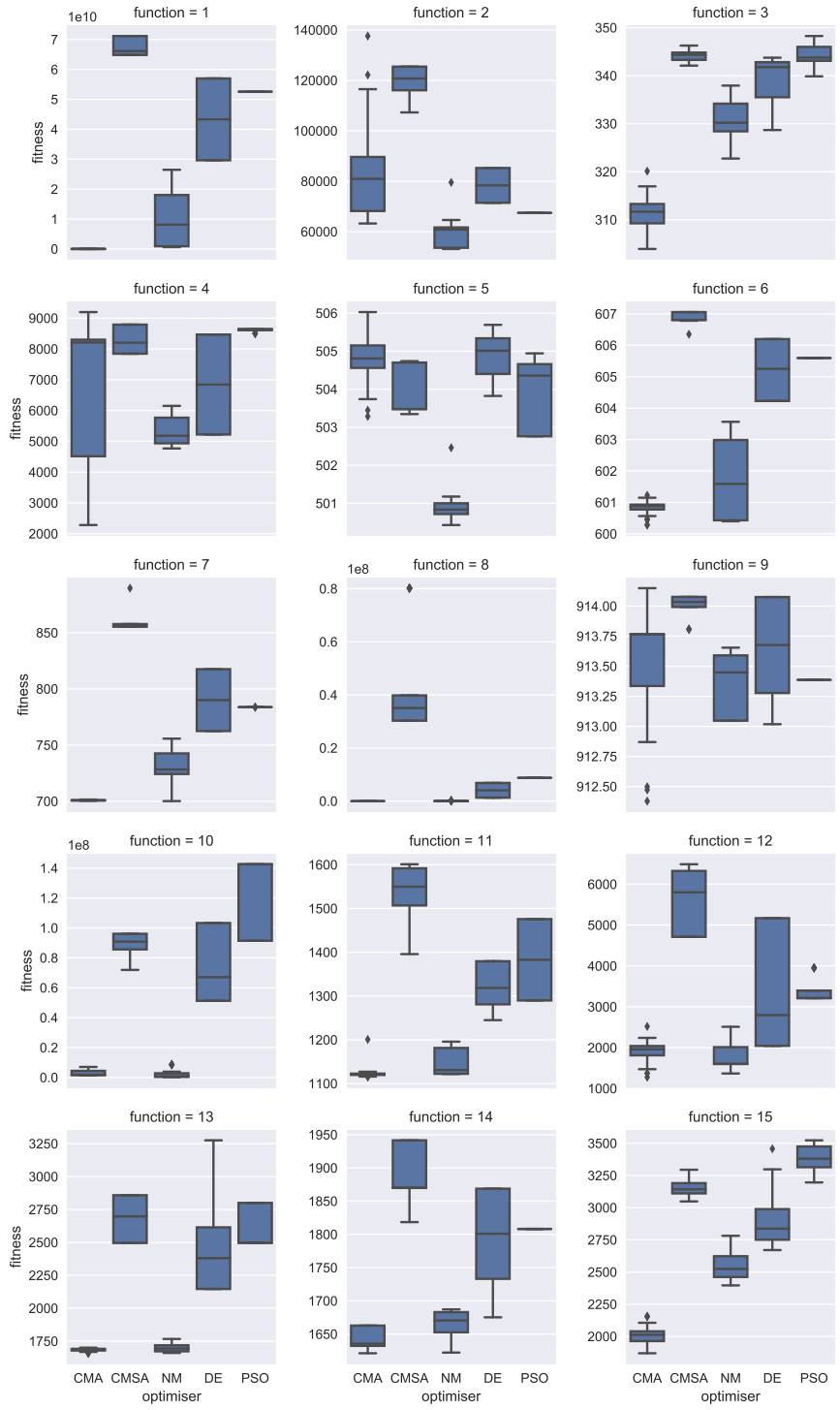


Figure 5.2 – Optimizers results on the expensive benchmark in dimension 30

5.3.1 Preliminary experiments: comparing various algorithms

Figures 5.1 and 5.2 show the results obtained by the standard algorithms on each function of the testbed in dimension 10 and 30 respectively. From this, we can with no surprise that CMA performs really well on most problems except on functions 4 and 5. It is also interesting to note that the results obtained by NM are often really good and able to compete with CMA on most functions, even getting better results in some cases.

Figures 5.3 and 5.4 show the results obtained by 9 different portfolios. Despite the lower budget allocated to a single optimizer, we can see that some of those portfolios are still able to get very good results when they include an algorithm well suited for the considered function. More importantly, we can see from functions 4 and 5 that while CMA isn't able to get good results by itself, when allowed to restart it becomes much better. While CMA takes advantage of the restarts - showing that it's very dependent on the initialization - it's much less obvious in some cases such as for PSO for example.

While the results of CMSA, DE and PSO cannot be described as good, a portfolio composed of those three algorithms (designed as '4' on the plots), is consistently able to achieve better results. This is very clearly illustrated on function 7 in dimension 30, where neither CMSA, DE nor PSO are able to get good results, while the portfolio is able to compete with the better ones.

5.3.2 Tuning, and adding quasi-random numbers

In order to see the impact of using Quasi-Random mutations on CMA, we performed tests on multiple variants of CMA: the initial step-size was set at 1 or 40 (denoted as BI), by setting the step-size lower-bound to 0.01 or 0.0001 (denoted as MS), and in addition to the normal population size, we tested with half the normal population size (denoted as SP) or twice the normal population size (denoted as BP). Finally, we also tested the elitist strategy (denoted as +). All variants where tested as is, and with Quasi-Random variations (denoted as QR).

As can be seen on Figure 5.5, with QR mutations, while the results are sometimes worse than normal, most of the time they are better. In higher dimension as seen in Figure 5.6, the QR implementations take a very clear advantage over the vanilla version.

As can be seen in Table 5.4, this improvement of QR is in fact found on each of the variants we tested.

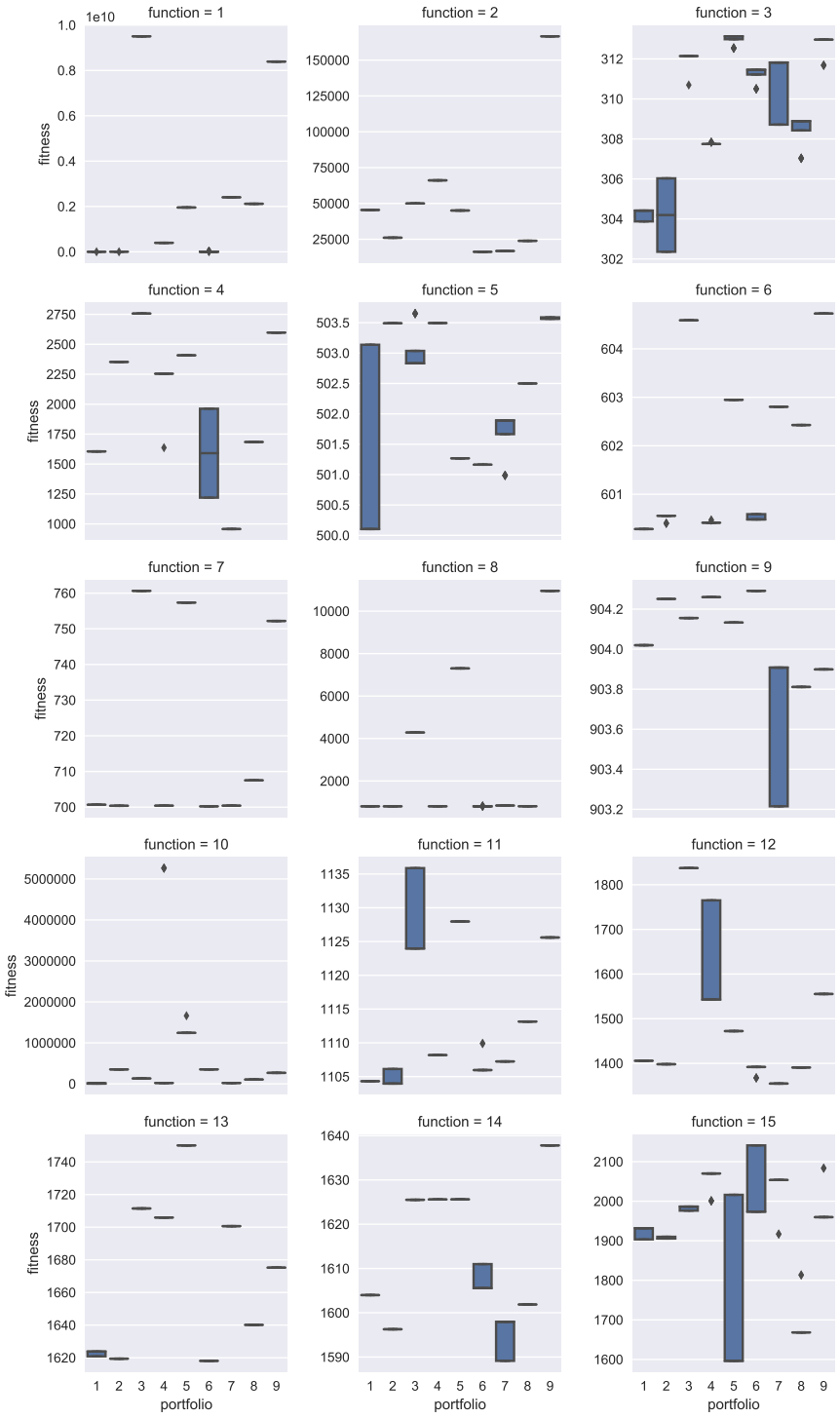


Figure 5.3 – Portfolio results on the expensive benchmark in dimension 10. The name of the portfolios where removed to keep the legend clear. Portfolios are: 1: CMA/CMA, 2: CMA/CMA/CMA, 3: CMSA/CMSA/CMSA, 4: CMSA/DE/PSO, 5: CMSA/NM/PSO, 6: DE/DE/DE, 7: NM/DE/PSO, 8: NM/NM/NM, 9: PSO/PSO/PSO

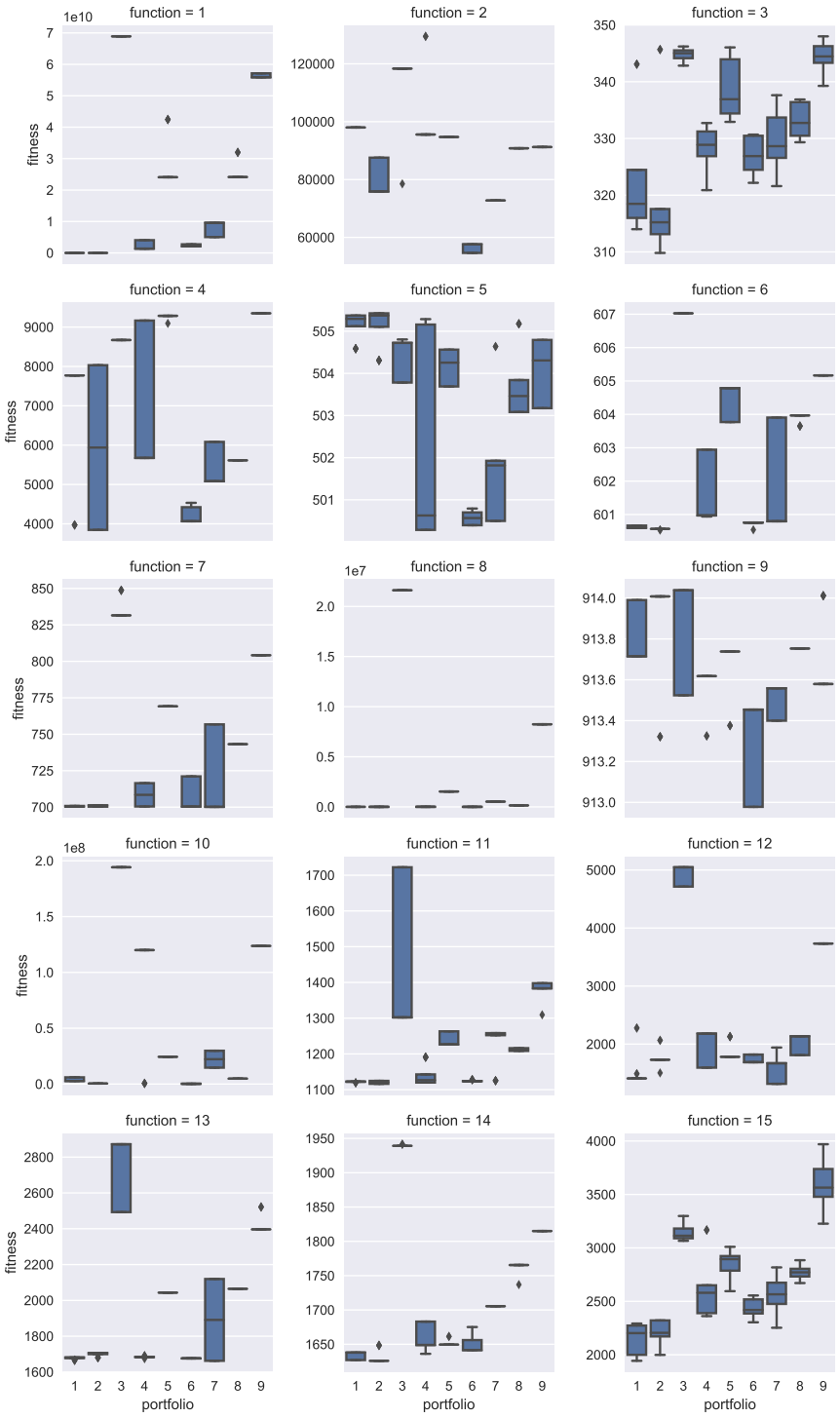


Figure 5.4 – Portfolio results on the expensive benchmark in dimension 30. The name of the portfolios where removed to keep the legend clear. Portfolios are: 1: CMA/CMA, 2: CMA/CMA/CMA, 3: CMSA/CMSA/CMSA, 4: CMSA/DE/PSO, 5: CMSA/NM/PSO, 6: DE/DE/DE, 7: NM/DE/PSO, 8: NM/NM/NM, 9: PSO/PSO/PSO

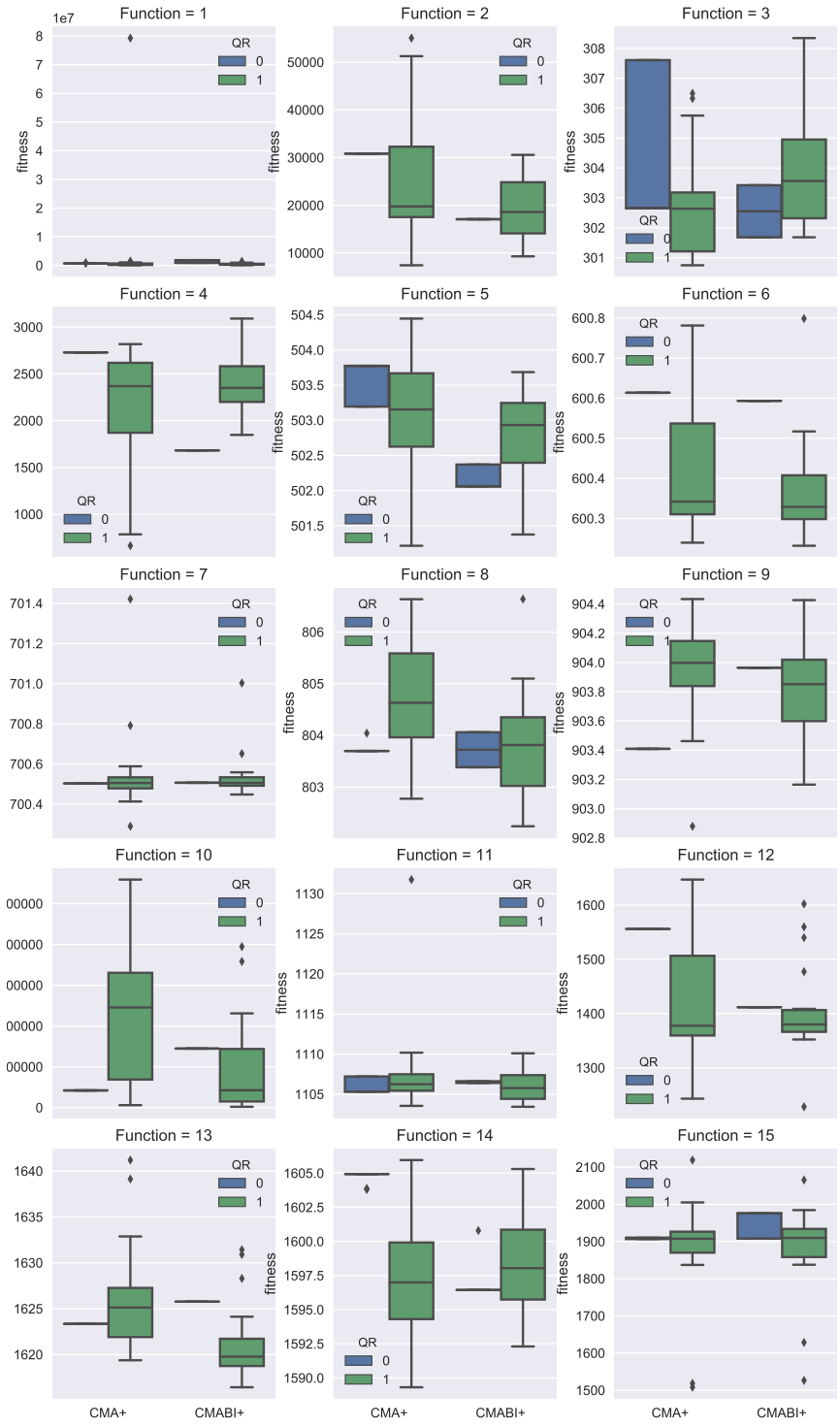


Figure 5.5 – Two CMA variants results on the expensive benchmark in dimension 10. QR = 0 denotes the Vanilla version, while QR = 1 is the version with Quasi-Random mutations.

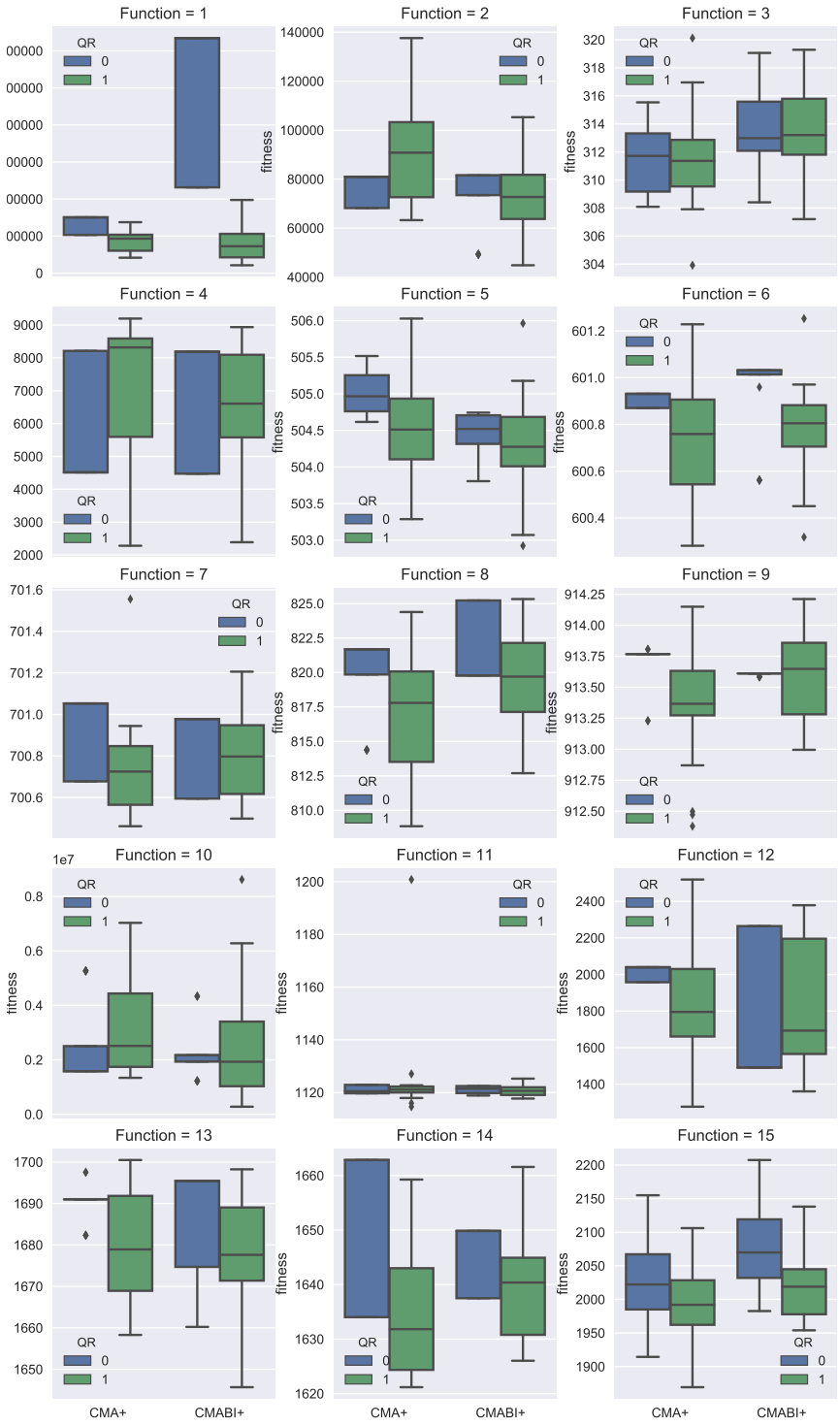


Figure 5.6 – Two CMA variants results on the expensive benchmark in dimension 30. QR = 0 denotes the Vanilla version, while QR = 1 is the version with Quasi-Random mutations.

F	Best	Worst	Median	Mean	Std
1	3066.821720	27350.414967	10303.331393	13532.288748	8015.631303
2	17946.244786	75601.839181	38585.782508	39525.522486	15424.252191
3	1.010692	15.066042	6.772117	7.342122	4.307489
4	130.551411	2754.489295	2213.214380	1919.328740	798.520928
5	0.428649	4.694434	3.378933	3.178128	1.239418
6	0.257797	0.816413	0.500261	0.494573	0.153775
7	0.288968	1.247096	0.495755	0.590027	0.302133
8	1.621837	6.750625	4.885070	4.552976	1.573973
9	3.353918	4.595982	4.222199	4.156896	0.340649
10	11346.738112	2450865.764312	263107.148179	556046.674992	730726.425838
11	2.187530	40.353662	6.419068	8.149008	7.903810
12	31.308448	610.640458	307.249692	322.513038	153.876607
13	316.642541	333.405878	320.124508	321.868712	4.217949
14	186.334995	210.410137	202.610188	201.771257	6.009089
15	10.511656	528.958620	408.019290	340.349673	170.162921

Table 5.1 – Errors obtained by CMASPHI-QR in 10D

5.4 CEC 2015 criteria

The CEC 2015 testbed uses an average between the mean and the best fitness, at the end of the budget of 1500 evaluations (in dimension 30) and 500 evaluations (in dimension 10), and also at half budget.

The lowest errors we were able to obtain with one variant of CMA-ES (termed CMASPHI-QR in the following sections) are reproduced in tables 5.1 and 5.2. The corresponding complexities are shown in tables 5.3a and 5.3b.

5.5 Other results

In this section, we present several results for all tested algorithms. We reproduced each run 10 times; the standard deviation provided in this section is the standard deviation of the score; *i.e.* they correspond to the variability of one run. Thanks to the averaging over 10 runs, our results are more significant.

Table 5.4 synthesizes the results of different variants of CMA-ES. HI refers to 80 as an initial step-size. MI refers to 20 as an initial step-size. SP refers to small population size $\frac{1}{2}(3\log(N)+4)$ instead of $3\log(N)+4$. BP refers to big population size $2(3\log(N)+4)$.

F	Best	Worst	Median	Mean	Std
1	4410.546684	2433205.172072	9909.121145	203114.035298	611688.572924
2	73838.778110	155921.584174	118489.287353	117191.061224	21987.789028
3	13.348001	37.062026	20.819096	21.305606	5.062646
4	2553.626647	9575.708781	8591.667368	7588.016926	2185.679036
5	4.662556	6.673121	5.284304	5.437349	0.591955
6	0.373039	1.045966	0.659812	0.685387	0.235695
7	0.323589	1.313365	0.535025	0.636289	0.286018
8	6.749165	56.118232	22.244325	22.911469	10.927353
9	13.241557	14.225287	13.976680	13.848334	0.303010
10	610222.482957	8220937.807046	2549366.836787	2774403.117855	2212879.888269
11	17.995147	88.531156	22.161005	27.828859	19.823562
12	224.026689	1078.687135	640.246067	671.996207	282.541119
13	354.170628	400.510809	376.774443	377.075234	11.912947
14	218.364928	271.850932	243.197289	245.710839	15.669614
15	600.342406	887.127952	771.879600	758.132119	79.104475

Table 5.2 – Errors obtained by CMASPHI-QR in 30D

(a) Computational Complexity for best results in 10D

Func.	$\hat{T}_1/T_0$
1	5.20045e-01
2	2.61135e-01
3	1.05740e+00
4	2.81510e-01
5	5.36625e-01
6	3.41910e-01
7	3.59610e-01
8	3.36335e-01
9	2.66570e-01
10	2.80205e-01
11	4.34245e-01
12	3.10260e-01
13	2.94360e-01
14	3.13660e-01
15	1.13885e+00

(b) Computational Complexity for best results in 30D

Func.	$\hat{T}_1/T_0$
1	1.25175e+00
2	1.17270e+00
3	8.35025e+00
4	1.27570e+00
5	3.27215e+00
6	1.22645e+00
7	1.24210e+00
8	1.29370e+00
9	1.15715e+00
10	1.46540e+00
11	2.71990e+00
12	1.56960e+00
13	1.54560e+00
14	1.67210e+00
15	8.83170e+00

Nelder-Mead variants are shown in table 5.5. For Differential Evolution (Table 5.6), the best performing one is a classical current to best with reasonably standard parametrization. PSO results are shown in table 5.7. Unless stated otherwise, the initial velocity is 1 and the maximum velocity MV is 1.5.

5.6 Conclusion

Without surprise, when comparing basic algorithms on this testbed, CMA-ES was the best performing one. Perhaps more surprisingly, NM was able to get very good results on most problems and was able to compete against CMA-ES. DE, CMSA and PSO however suffered a lot from the short budget, and were never really able to get good results.

Despite the small budget, we can already see that portfolios (or restarts on individual algorithms) are competitive and able to quickly select the best performing optimizer on a given problem, without too much cost. In some cases, we even observed that a portfolio of three algorithms performing poorly on a problem was able to get better results.

On this testbed, we were able to validate the use of Quasi-Random mutations for CMA-ES, since usually - though not on all functions - DCMA outperformed CMA, with a clear overall improvement, the best overall variant being one with Quasi-Random mutation. When looking at each function in each dimension, we also noticed that in every case, the best performing variant was one with Quasi-Random mutation.

We did not include mirroring[Teytaud et al., 2006]. We did not investigate sophisticated memmetic algorithms on top of CMA. We did not experiments on dimensions other than those proposed in the Cec2015 expensive optimization testbed. Portfolios methods were tested, and validated for the total budget, but not for the mid-budget criterion.

CMA Variants	Mean	STD	Median	Rank
CMA+	5.78e8	1.60e8	5.34e8	20
CMA+QR	4.54e8	2.90e7	4.59e8	15
CMABI+	5.99e8	1.81e8	5.41e8	22
CMABIQR	4.52e8	2.81e7	4.46e8	14
CMABI	6.17e8	1.94e8	5.80e8	24
CMABISS+QR	4.54e8	2.90e7	4.59e8	15
CMABISS+	6.06e8	1.87e8	5.38e8	23
CMABISSQR	4.51e8	2.70e7	4.49e8	13
CMAMI+	5.79e8	1.73e8	5.98e8	21
CMAMI+QR	4.55e8	2.83e7	4.59e8	18
CMASPBI+	7.20e7	1.98e7	7.69e7	10
CMASPBI+QR	5.54e7	8.35e6	5.38e7	6
CMABPBI+	5.39e9	6.23e8	5.40e9	27
CMABPBI+QR	4.30e9	1.98e8	4.24e9	25
CMABPBI	5.72e9	1.20e9	5.62e9	28
CMABPBIQR	4.30e9	1.98e8	4.24e9	25
CMASPBI	8.08e7	2.52e7	7.66e7	12
CMASPBIQR	5.21e7	3.47e6	5.20e7	1
CMASPBISS+	7.39e7	2.66e7	6.42e7	11
CMASPBISS+QR	5.24e7	3.55e6	5.26e7	5
CMASPBISS	6.57e7	1.99e7	6.97e7	8
CMASPBISS+QR	5.21e7	3.47e6	5.20e7	1
CMASPMI+	6.96e7	1.56e7	6.76e7	9
CMASPMI+QR	5.21e7	3.47e6	5.20e7	1
CMASPHI+	6.53e7	1.80e7	6.34e7	7
CMASPHI+QR	5.21e7	3.47e6	5.20e7	1

Table 5.4 – Results of different variants of CMA-ES

NM Variants	Mean	STD	Median	Rank
$\alpha = 1, \gamma = 2, \rho = -.5, \sigma = .5$	1.11e10	5.16e9	1.00e10	2
$\alpha = .8, \gamma = 2, \rho = -.5, \sigma = .5$	3.27e10	1.16e10	2.98e10	5
$\alpha = 1, \gamma = 2, \rho = -.75, \sigma = .5$	1.13e10	6.51e9	9.91e9	3
$\alpha = 1, \gamma = 2, \rho = -.5, \sigma = .75$	1.06e10	5.41e9	9.72e9	1
$\alpha = 1.5, \gamma = 3, \rho = -.25, \sigma = .25$	3.55e10	1.28e10	3.26e10	6
$\alpha = .5, \gamma = 1.5, \rho = -.75, \sigma = .75$	9.84e10	1.02e10	1.00e11	7

Table 5.5 – Results of different variants of Nelder-Mead

DE Variants	Mean	STD	Median	Rank
$DE/curr-to-best/1$ $f_1 = .8, f_2 = .8, cr = .5$	7.17e10	8.69e9	7.14e10	8
$DE/curr-to-best/1$ $f_1 = 1, f_2 = 1, cr = .8$	1.34e11	2.52e10	1.38e11	9
$DE/curr-to-best/1$ $f_1 = 1, f_2 = 1, cr = .5$	6.72e10	1.29e10	6.96e10	1
$DE/curr-to-best/1$ $f_1 = .5, f_2 = .8, cr = .5$	7.02e10	1.24e10	7.01e10	5
$DE/rand-to-best/1$ $f_1 = 1, f_2 = 1, cr = .5$	7.14e10	1.12e10	6.99e10	6
$DE/best/1$ $f_1 = 1, f_2 = 1, cr = .5$	6.95e10	9.39e9	6.97e10	4
$DE/best/1$ $f_1 = 1, f_2 = 1, cr = .5$	6.94e10	9.55e9	6.97e10	3
$DE/best/1$ $f_1 = 1, f_2 = 1, cr = .5$	6.94e10	9.54e9	6.97e10	2

Table 5.6 – Results of different variants of Differential Evolution

PSO Variants	Mean	STD	Median	Rank
$\mu = 30, neighb = 10, \omega = 1/(2\log(2))$, $\phi_p = .5+\log(2), \phi_g = .5+\log(2)$	1.46e11	2.33e10	1.44e11	12
$\mu = 30, neighb = 5, \omega = 1/(2\log(2))$, $\phi_p = .5+\log(2), \phi_g = .5+\log(2)$	1.50e11	1.60e10	1.48e11	13
$\mu = 15, neighb = 5, \omega = 1/(2\log(2))$, $\phi_p = .5+\log(2), \phi_g = .5+\log(2)$ 1702	1.45e11	1.43e10	1.41e11	11
$MV = 2.5, \mu = 30, neighb = 15, \omega = 1/(2\log(2))$, $\phi_p = .5+\log(2), \phi_g = .5+\log(2)$	1.13e11	1.57e10	1.13e11	8
$MV = 5, \mu = 15, neighb = 5, \omega = 1/(2\log(2))$, $\phi_p = .5+\log(2), \phi_g = .5+\log(2)$	8.48e10	1.63e10	7.78e10	5
$MV = 2.5, \mu = 30, neighb = 10, \omega = 1/(2\log(2))$, $\phi_p = .5+\log(2), \phi_g = .5+\log(2)$ 1705	1.11e11	1.54e10	1.11e11	6
$MV = 2.5, \mu = 30, neighb = 10, \omega = 1/(2\log(2))$, $\phi_p = 1.5+\log(2), \phi_g = 1.5+\log(2)$	1.14e11	1.55e10	1.16e11	9
$MV = 1.5, \mu = 30, neighb = 10, \omega = 1/(2\log(2))$, $\phi_p = .5+\log(2), \phi_g = .5+\log(2)$	1.43e11	1.22e10	1.45e11	10
$MV = 2.5, \mu = 30, neighb = 10, \omega = 1/(2\log(2))$, $\phi_p = 1.5+\log(2), \phi_g = 1.5+\log(2)$ 1708	1.11e11	1.13e10	1.11e11	7
$MV = 10, \mu = 30, neighb = 5, \omega = 1/(2\log(2))$, $\phi_p = .5+\log(2), \phi_g = .5+\log(2)$	5.63e10	8.16e9	5.77e10	3
$MV = 20, \mu = 30, neighb = 5, \omega = 1/(2\log(2))$, $\phi_p = .5+\log(2), \phi_g = .5+\log(2)$	4.45e10	1.34e10	4.60e10	1
$MV = 10, \mu = 30, neighb = 5, \omega = 1/(2\log(2))$, $\phi_p = .5+\log(2), \phi_g = .5+\log(2)$ 1711	5.89e10	1.27e10	5.50e10	4
$MV = 20, \mu = 30, neighb = 5, \omega = 1/(2\log(2))$, $\phi_p = .5+\log(2), \phi_g = .5+\log(2)$	4.64e10	1.49e10	4.70e10	2

Table 5.7 – Results of different variants of PSO

Chapter 6

Sieves method in fuzzy control: logarithmically increase the number of rules

This chapter's content comes from the paper Berthier, V. and Teytaud, O. (2015b). Sieves method in fuzzy control: logarithmically increase the number of rules. In *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–9. IEEE Press. Its abstract was:

The Sieves method, in statistics, consists in extending a model progressively, as new data are made available. Typically, parameters are progressively added in a statistical estimation method while new samples are provided. We propose an adaptation of the Sieves method in optimization. Decision variables are progressively added while new fitness evaluations are received. We experiment the method on a simple set of noisy optimization problems, and then on a fuzzy control problem applied to unit commitment. The obtained algorithm is simple, applicable to various optimization algorithms (not only evolutionary optimization), and seemingly robust.

6.1 Introduction

The Sieves method consists in progressively adding variable in a computational intelligence problem. While it is classical in statistics, both supported by a range of experiments and a body of theoretical works, the Sieves method is unusual in optimization. In this chapter we (i) propose an adaptation of the Sieves method in optimization, (ii) perform artificial and (iii) perform real world experiments.

Section 6.1.1 describes the classical Sieves method in statistics. Section 6.1.2 proposes

a Sieves method in evolutionary optimization. Section 6.2 presents artificial experiments. Section 6.3 presents an application in fuzzy control.

6.1.1 Sieves method (SM)

Over-fitting is the poor behavior of parametric decision tools when the parameters are tuned on a too small dataset. Typically, over-fitting occurs when an optimization run is too short, compared to the number of parameters to be tuned. Over-fitting is widely discussed in supervised machine learning [Vapnik, 2013], but not that much in optimization and control. We will see (Fig. 6.3) that over-fitting does matter in Fuzzy control, and we propose a method for avoiding it.

There is a long tradition of considering the impact of dimension in optimization, statistics, machine learning. In statistics, this has given birth to the Sieves method: instead of considering a huge problem at once, we work on a small sub-problem (with far less variables), and then we progressively add other variables [Shen and Wong, 1994]. In optimization, a different approach has been preferred, based on decomposition: instead of working on a high-dimensional optimization problem, we work on several sub-problems, in a divide-and-conquer manner. For example, in mathematical programming [Nowak and Römisch, 2000] involves a master, coordinating the global optimization, and several sub-problems. In evolutionary computation, some main successes are based on the decomposition of a big problem into several sub-problems, with an evolutionary coordination, as in divide-and-evolve [Schoenauer et al., 2006].

A particular form of noisy optimization is simple regret bandits [Bubeck et al., 2011]. In such a setting, there is a counterpart of the Sieves method, namely Progressive Widening ([Coulom, 2006, Chaslot et al., 2007] for the experimental part, [Wang et al., 2009] for a mathematical analysis). However, these works on progressive widening are limited to discrete sets of actions with little or no structure.

There has been little effort on the application of Sieves methods to optimization. We here consider Sieves methods for noisy optimization.

6.1.2 Sieves method for evolutionary algorithms

Evolutionary algorithms are population-based meta-heuristics. At each iteration, offspring are generated. Each individual (offspring) is evaluated. The best individuals become the parents and generate the next offspring. As the optimization algorithm is not the core of this chapter, we refer to [Beyer and Schwefel, 2002, Beyer, 2001] for more details on SAES; the Sieves method that we propose can be implemented for other evolution strategies without changing the principles. We use a Sieves index $SievesIndex(i,d)$ which

decides, at iteration i and if the problem dimension is d, how many coordinates are considered. The pseudo-code of the Sieves method for (μ,λ)-SAES is presented in Alg.6.1.

Algorithm 6.1 Sieves method applied to (μ,λ)-SAES in dimension d. $\mathcal{N}$ denotes independent centered standard Gaussian (in dimension given by the context). The fitness function is obtained by averaging multiple reevaluations, as detailed in the experimental section.

Parameters: parent population size μ, step-size mutation rate τ, population size λ, initial parent population Pop, each parent is $Pop_i = (x_i, \sigma_i)_{i\in\{1,\ldots,\lambda\}}$ with σ_i initialized at some σ_0.
$i \leftarrow 0$
while Computation time not exhausted **do**
  $i \leftarrow i+1$
  for $j \in \{1,\ldots,\mu\}$ **do**
    $\sigma_j \leftarrow \sigma_j \times \exp(\tau\mathcal{N})$
  end for
  for $j \in \{1,\ldots,\lambda\}$ **do**
    $o_j = x$
    for $u = 1$ to $SievesIndex(i,d)$ **do**
      $(o_j)_u += \sigma_j\mathcal{N}$
    end for
    $f_j = fitness_{reevaluations}(o_j)$
  end for
  Sort individuals, so that $f_1 \leq f_2 \leq \cdots \leq f_\lambda$
  $\forall i \in \{1,\ldots,\mu\}, Pop_i \leftarrow (o_i, \sigma_i)$
end while

A $(\mu/\mu,\lambda)$-SAES version is also used. It is recalled in Alg. 6.2.

6.2 Artificial experiments

In a noisy optimization problem, the objective function f depends on the chosen search point θ and on a noise component ω. We consider the optimization of the following simple noisy sphere problem: $f(\theta,\omega) = ||\theta-\theta^*||^2 + \omega$, where ω is an independent centered standard Gaussian noise. The optimization is performed by a self-adaptive evolution strategy (SAES [Beyer and Schwefel, 2002]). The optimum is θ^*, which is randomly drawn as follows: $\forall j \in \{1,2,\ldots,100\}, \theta^*_j = \mathcal{N}/j^\gamma$ for various γ, which is the parameter controlling the averaged ill-condition of the problem: the higher γ is, the higher the ill-condition is (discussion later). The initial search point for SAES is 0. $\gamma > 0$ indicates that the first

Algorithm 6.2 Sieves method applied to $(\mu/\mu,\lambda)$-SAES. $\mathcal{N}$ denotes independent centered standard Gaussian (in dimension given by the context). The fitness function is obtained by averaging multiple reevaluations, as detailed in the experimental section.

Parameters: parent population size μ, step-size mutation rate τ, population size λ, initial parent x, initial step-size $\sigma = \sigma_0$
$i \leftarrow 0$
while Computation time not exhausted **do**
  $i \leftarrow i+1$
  for $j \in \{1,\ldots,\lambda\}$ **do**
    $\sigma_j = \sigma \exp(\tau\mathcal{N})$
    $o_j = x$
    for $u = 1$ to $SievesIndex(i,d)$ **do**
      $(o_j)_u += \sigma_j\mathcal{N}$
    end for
    $f_j = fitness_{reevaluations}(o_j)$
  end for
  Sort individuals, so that $f_1 \leq f_2 \leq \cdots \leq f_\lambda$
  $x \leftarrow \frac{1}{\mu}\sum_{j=1}^{\mu} o_j$
  $\sigma \leftarrow \exp(\frac{1}{\mu}\sum_{j=1}^{\mu} \ln(\sigma_j))$
end while

variable is (on average) more important than the second, which is more important than the third and so on.

The initial step-size σ_0 is the same for all coordinates, and we test several values. Each experiment is reproduced 57 times. Each function evaluation is repeated several times in order to mitigate the level of noise. More precisely, each evaluation at iteration i is repeated i^2 times and the obtained fitness values are averaged [Astete Morales et al., 2013].

We apply the Sieves method for optimization as follows. We compare several $sievesIndex(i,d)$ functions:

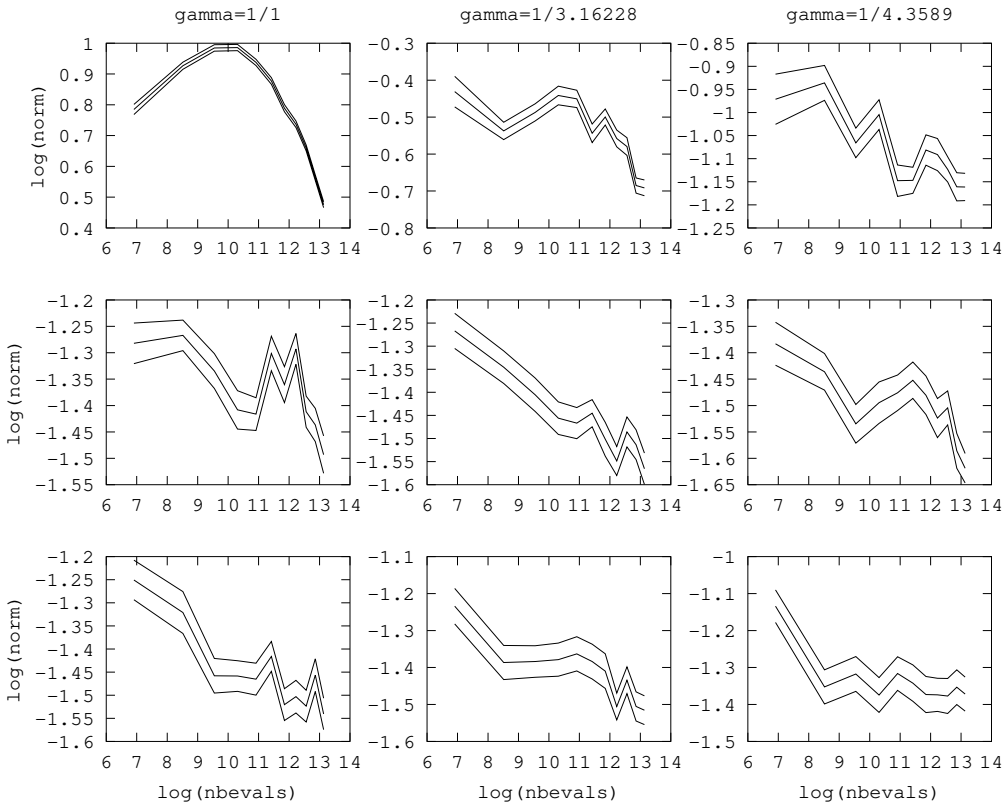
$$sievesIndex(i,d) = \lceil \frac{d}{1+9(k-1)} i^{(1+9(k-1))^{-\frac{1}{2}}} \rceil$$

for $k \in \{1,\ldots,9\}$. $k = 1$ means no Sieves method. k large means a strong Sieves method (*i.e.*additional variables are added very slowly).

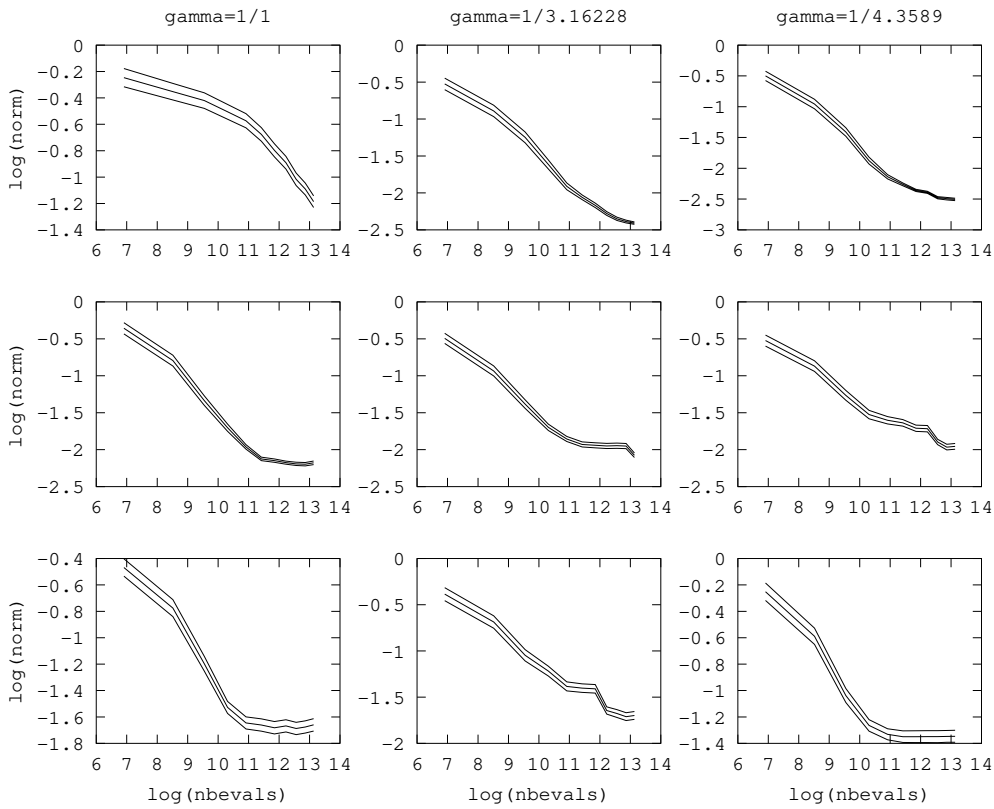
With $\gamma = 1.5$ (leading to an average condition of 1000) we get results as presented in Fig. 6.1a, Fig. 6.1b, Fig. 6.1c, depending on the initial step-size σ_0 (too small, correct, too large, respectively), with isotropic-SAES. Across all experiments performed here, $k = 6$ gave the best overall performances, leading to $sievesIndex(i,d) = \lceil \frac{20}{9} i^{1/\sqrt{46}} \rceil \simeq 2i^{.15}$.

Figure 6.1 – Impact of the k parameter on the Sieves method.

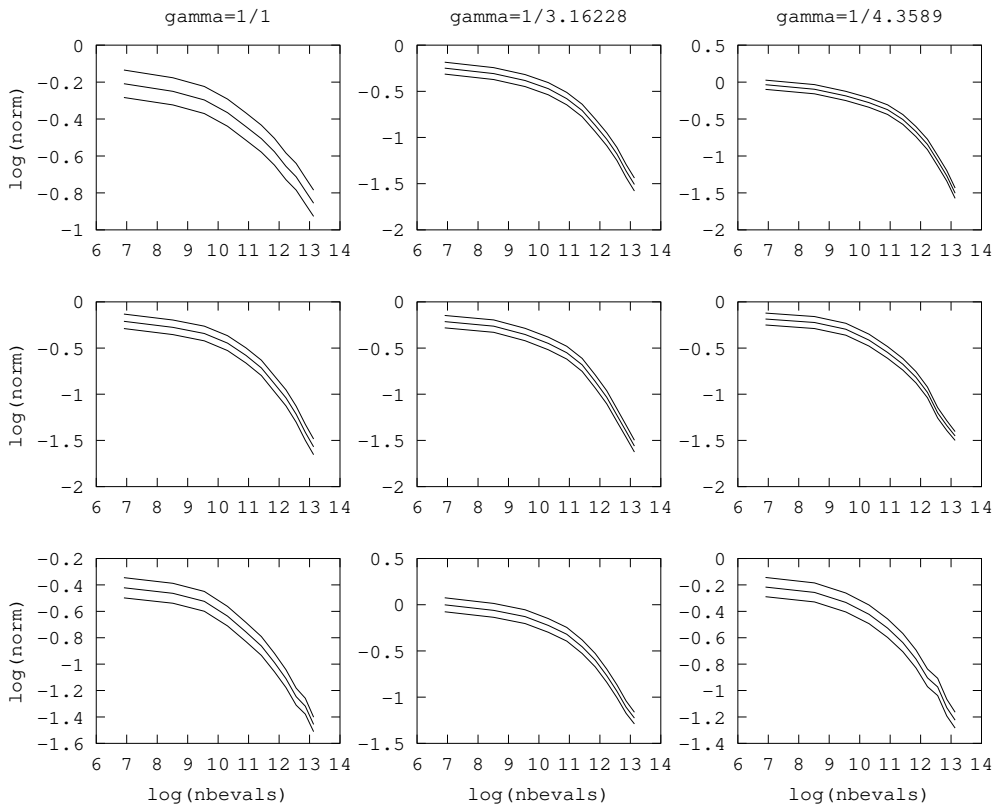
(a) Results for $k = 1, 2, \ldots, 9$, for an initial step-size $\sigma_0 = 2.5$ i.e. way too large. $k = 1$ means no Sieves method. $k = 6$ is one of the good values. The three curves are average, average + standard deviation, average - standard deviation, respectively.



(b) Results for $k = 1, 2, \ldots, 9$, for an initial step-size $\sigma_0 = 5/16$ i.e. approximately ok. $k = 1$ means no Sieves method. $k = 6$ is one of the good values. The three curves are average, average + standard deviation, average - standard deviation, respectively.



(c) Results for $k = 1, 2, \ldots, 9$, for an initial step-size $\sigma_0 = 5/54$ i.e. way too small. $k = 1$ means no Sieves method. $k = 6$ is one of the good values. The three curves are average, average + standard deviation, average - standard deviation, respectively.



We reproduced the experiments with various values of γ. Results were always positive with $\gamma > 1$ (i.e. the Sieves method worked better than the original method), but the optimal k was not always the same and $k = 6$ was not always better than no Sieves. With $\gamma < 1$, results were not good, whatever maybe k. We therefore conclude these preliminary experiments as follows:

- The Sieves method in optimization does not work when all coordinates are approximately equally important ($\gamma < 1$).
- The optimal parametrization of the *SievesIndex* function is problem-dependent, so some sub-optimal parametrizations seemingly cover a wide range of problems.

In the next section, we focus on fuzzy control, which is arguably a natural candidate for the Sieves method in optimization: there are many parameters, and the ranking of the

parameters is easy: a fuzzy controller is a combination of rules, the ordering of which does not matter, and the first rules are more important than the next ones. We got in the experiments above a polynomial Sieves function with a very small exponent, indeed close to a logarithmic function; we will see that a single logarithmic function will work fine on several of our fuzzy control problems.

6.3 Applications to fuzzy control

Section 6.3.1 introduces direct policy search. Section 6.3.2 briefly presents fuzzy control. Section 6.3.3 presents our optimization problem. Section 6.3.4 presents our optimization algorithm. Section 6.3.5 presents preliminary experiments. Section 6.3.6 presents our parametrization of the SM. Section 6.3.7 shows our results.

6.3.1 Direct Policy Search and noisy optimization

Direct Policy Search (DPS) consists in optimizing a parametric policy directly on simulations, i.e. the objective function simulates the whole system with the parametric decision policy. There are many such works [Bengio, 1998] with neural networks, and in fuzzy systems [Zadeh, 1990]. We here focus on fuzzy systems. Technically, DPS boils down to noisy optimization. Noisy optimization can be performed by various methods, including gradient-free gradient descent [Fabian, 1967] (the gradient is estimated by differences) or evolutionary algorithms [Heidrich-Meisner and Igel, 2009, Arnold and Beyer, 2006]. We here use evolutionary algorithms.

6.3.2 Fuzzy control

We consider fuzzy functions as follows:

$$\begin{aligned} action(s) &= \sum_{i=1}^{s} \frac{w_i(s)}{\sum_{j=1}^{s} w_j(s)} \theta_{i,1} \\ w_i(s) &= \\ I(\theta_{i,2}) &\times \prod_{j=1}^{d} \max\left(0, \frac{1-|s_j-\theta_{i,3,j}|}{\exp(d+\theta_{i,4,j})}\right) \end{aligned} \tag{6.1}$$

where $I : x \mapsto \frac{1}{2}(1+x/\sqrt{x^2+1})$ is a mapping from $\mathbb{R}$ to $[0,1]$, and where d is the dimension of the state space s and $action(s)$ is the chosen action for state s. As we consider methods adding progressively new rules, we also tested the replacement of Eq. 6.1 by Eq. 6.2, and

also by Eq. 6.3; there are aimed at reducing the weights of newly added rules. We mention them for the sake of completeness, but they did not bring any clear improvement:

$$\begin{aligned} w_i(s) &= \\ I(\theta_{i,2}-7) &\times \prod_{j=1}^{d} \max(0, 1-\frac{|s_j-\theta_{i,3,j}|}{\exp(d+\theta_{i,4,j})}) \end{aligned} \tag{6.2}$$

$$\begin{aligned} w_i(s) &= \\ I(\theta_{i,2}-2i) &\times \prod_{j=1}^{d} \max(0, 1-\frac{|s_j-\theta_{i,3,j}|}{\exp(d+\theta_{i,4,j})}) \end{aligned} \tag{6.3}$$

This means that the parameter is $\theta = (\theta_{i,1}, \theta_{i,2}, \theta_{i,3}, \theta_{i,4})_{i\in\{1,\ldots,NbRules\}}$ when the number of rules is $NbRules$. $\theta_{i,1}$ is the action chosen by the rule number i; $\theta_{i,2}$ is the a priori weight of the rule number i; $\theta_{i,3}$ is the typical state at which the rule applies; $\theta_{i,4}$ indicates the scope of the rule (the larger $\theta_{i,4,j}$, the wider the scope of the rule on axis j). We also tested a different membership function, i.e. $w_i(s)$ having no prior weight:

$$w_i(s) = \prod_{j=1}^{d} \max\left(0, \frac{1-|s_j-\theta_{i,3,j}|}{\exp(d+\theta_{i,4,j})}\right). \tag{6.4}$$

One important point here, is that when all parameters are set to zero, it's equivalent to the rule not existing at all: it's completely neutral, with no impact whatsoever on the fitness.

6.3.3 Objective function

The test case is a Unit Commitment problem with a number of hydroelectric production plants and a thermal unit. Given a demand in power, the goal is to produce as much electricity as possible from the dams (since it's very cheap) and as little as possible from the thermal plant (since it's expensive) on each successive time steps.

$$Cost = -\sum_{t=0}^{T}(max(0, Demand_t - \sum_{s=0}^{n_stocks} Production_{s,t}) \times ThermalCost) - Penalisation$$

The penalization term is here to ensure that we do not produce too much electricity from the dam, since it would waste water (at the very least, in a real network there would be other consequences).

Of course, to produce electricity from the dams, it is necessary to use water. This water may come from two sources: natural inflows (rain, snow melt, *etc.*), or from another dam situated higher in the chain. The main difficulty of the problem is that the natural inflows depend on the season and that there is a random component with more or less noise (so

Parameter	Value
Variant	DE/curr-to-best/1 i.e. $x'_i = x_i + F1(x_a - x_b) + F2(x_{best} - x_i)$ with x'_i the proposal, a and b random, x_{best} the best in the population.
Population size	30
Cr	0.5
F1	0.8
F2	0.8
Resampling	$10\sqrt{n}$ resamplings at iteration n

Table 6.1 – Parameters of our DE algorithm when optimizing N parameters, at iteration i.

more or less predictable), and that of course, water used at time t is not available anymore at time $t+1$ (at least by the same dam, since it might be used by a lower dam if they are linked).

6.3.4 Optimization algorithm

The optimization algorithm is a Differential Evolution [Storn and Price, 1997]. We use the same resampling policy as above, with coefficients optimized on the target problem, before including our Sieves method. The parameters used are presented in Table 6.1.

6.3.5 Preliminary experiments

There are many fuzzy membership functions available in the literature [Monicka et al., 2011, Zhao and Bose, 2002], so we conducted experiments for comparing some of them. Eq. 6.1 uses as a distance the product, over coordinates, of linearly decreasing member functions. We also tested Eq. 6.4, for checking the robustness of the method. We also tested two different problems, both however in the family of unit commitment.

6.3.6 Sieves parametrization

When using the Sieves Method, it is important to determine how the number of parameters should increase. In our experiments, we tested two different scheme, both depending on

Budget (s)	No PW	rate = 0.20	rate = 0.25	rate = 0.30
1	**-1.29e+04** (±2.66e+02)	-1.33e+04 (±1.92e+02)	-1.33e+04 (±2.16e+02)	-1.33e+04 (±1.53e+02)
2	**-1.26e+04** (±2.75e+02)	-1.30e+04 (±1.95e+02)	-1.31e+04 (±1.65e+02)	-1.32e+04 (±2.71e+02)
4	**-1.23e+04** (±3.53e+02)	-1.28e+04 (±3.31e+02)	-1.28e+04 (±2.55e+02)	-1.28e+04 (±2.23e+02)
8	**-1.23e+04** (±4.62e+02)	-1.27e+04 (±5.13e+02)	-1.26e+04 (±4.61e+02)	-1.27e+04 (±6.20e+02)
16	**-1.21e+04** (±2.63e+02)	-1.24e+04 (±3.77e+02)	-1.24e+04 (±3.43e+02)	-1.23e+04 (±2.64e+02)
32	**-1.20e+04** (±2.96e+02)	-1.21e+04 (±2.80e+02)	-1.21e+04 (±3.09e+02)	-1.21e+04 (±2.80e+02)
64	-1.20e+04 (±4.56e+02)	-1.18e+04 (±2.09e+02)	-1.19e+04 (±3.70e+02)	**-1.17e+04** (±3.33e+02)
128	-1.19e+04 (±2.53e+02)	-1.14e+04 (±2.48e+02)	-1.13e+04 (±3.39e+02)	**-1.11e+04** (±3.81e+02)
256	-1.17e+04 (±3.72e+02)	-1.10e+04 (±2.64e+02)	-1.06e+04 (±4.33e+02)	**-1.05e+04** (±3.68e+02)
512	-1.17e+04 (±3.97e+02)	-1.06e+04 (±4.14e+02)	-1.02e+04 (±3.23e+02)	**-1.02e+04** (±4.81e+02)

Table 6.2 – Sieves Method with Linear increase on the Big case, with the membership function 6.1

the optimization time and overall budget. One of them was a linear increase:

$$SievesIndex(t) = nbParamsPerRule \times \lceil t/rate \rceil$$

And the other one was logarithmic:

$$SievesIndex(t) = nbParamsPerRule \times \lceil \log_2(1+t)/rate \rceil$$

Where $SievesIndex(t)$ is the number of parameters optimized at time t, $nbParamsPerRule$ is the number of parameters of a rule in Fuzzy Control (number of parameters that depends on the number of stocks). Many different parameters were tried for $rate$, the only constraint being that by the end of the available budget, all rules were optimized.

Interestingly enough, if this constraint is respected, we were always able to improve the optimization result. Rules that are not yet considered have their parameters set to zero, which means they don't impact the solution found in any way.

6.3.7 Experimental results: Sieves method for fuzzy control

In our experiments we performed many tests, using both membership equations (6.1 and 6.4), using a small or large test case (the "Small" one had 5 plants and spanned 25 timesteps, while the "Big" one had 15 plants and 50 timesteps). On each of those test cases, we compared the two schemes of progression (linear and logarithmic) to the vanilla version, using different rates. Some typical results are visible in Figure 6.2 and 6.3: using the Sieves Method didn't always improve the results by a wide margin but it was always better. In some cases, the results were much better with the Sieves Method than without.

Tables 6.2 through 6.9 synthesise the results obtained. As can be seen, in every case using the Sieves Method improved the results. Something interesting to note is that in

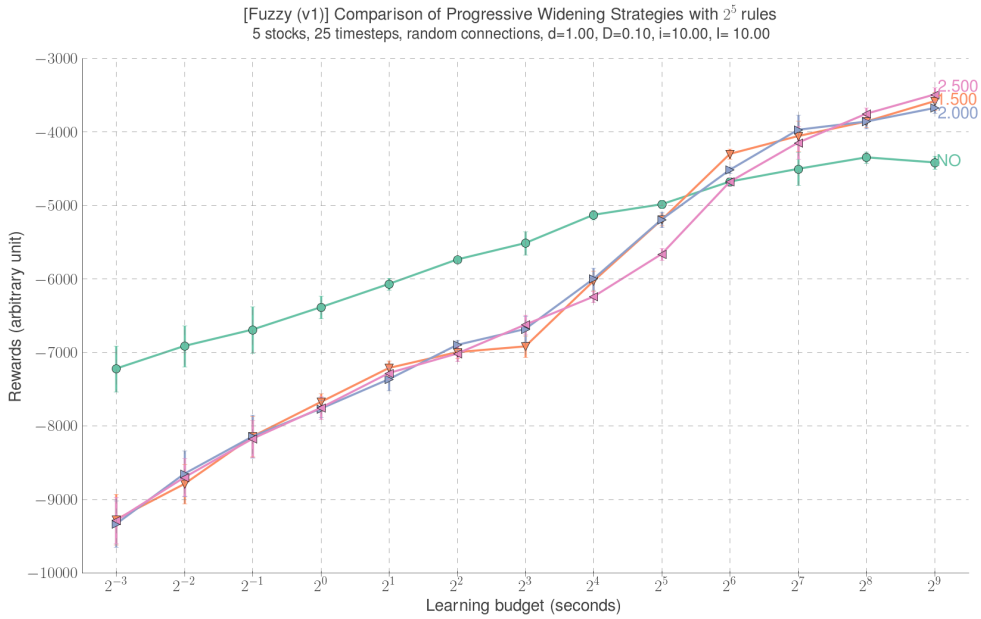


Figure 6.2 – Results with Eq. 6.1, smaller test case, logarithmic Sieves. "NO" indicates the results with no Sieves Method, while the numbers on the other curves indicates the rates used. We compare our Sieves method to the standard optimization with 32 optimized rules from the start. This case shows the typical behaviour of applying the Sieves Method to the optimization process: there's a noticeable cost at the start of the optimization process, but as the budget increases, the performances improve until they overtake the standard process.

Budget (s)	No PW	rate = 0.20	rate = 0.25	rate = 0.30
1	**-6.39e+03** (±4.85e+02)	-7.43e+03 (±4.48e+02)	-7.68e+03 (±3.96e+02)	-7.73e+03 (±2.64e+02)
2	**-6.07e+03** (±2.60e+02)	-7.17e+03 (±3.55e+02)	-7.01e+03 (±3.50e+02)	-6.93e+03 (±4.18e+02)
4	**-5.74e+03** (±1.65e+02)	-6.74e+03 (±2.78e+02)	-6.73e+03 (±2.59e+02)	-6.82e+03 (±3.39e+02)
8	**-5.51e+03** (±4.94e+02)	-6.16e+03 (±3.83e+02)	-6.44e+03 (±3.85e+02)	-6.53e+03 (±3.57e+02)
16	**-5.13e+03** (±1.98e+02)	-5.66e+03 (±3.74e+02)	-5.81e+03 (±3.34e+02)	-5.98e+03 (±4.25e+02)
32	**-4.99e+03** (±1.75e+02)	-5.06e+03 (±2.83e+02)	-5.26e+03 (±2.47e+02)	-5.08e+03 (±1.50e+02)
64	-4.68e+03 (±1.83e+02)	**-4.44e+03** (±2.01e+02)	-4.73e+03 (±2.21e+02)	-4.64e+03 (±2.31e+02)
128	-4.50e+03 (±7.00e+02)	**-4.04e+03** (±7.27e+02)	-4.25e+03 (±6.53e+02)	-4.25e+03 (±6.38e+02)
256	-4.35e+03 (±2.41e+02)	-3.95e+03 (±3.17e+02)	**-3.85e+03** (±3.14e+02)	-3.91e+03 (±2.49e+02)
512	-4.29e+03 (±3.31e+02)	-3.64e+03 (±2.49e+02)	-3.82e+03 (±1.97e+02)	**-3.63e+03** (±2.54e+02)

Table 6.3 – Sieves Method with Linear increase on the Small case, with the membership function 6.4

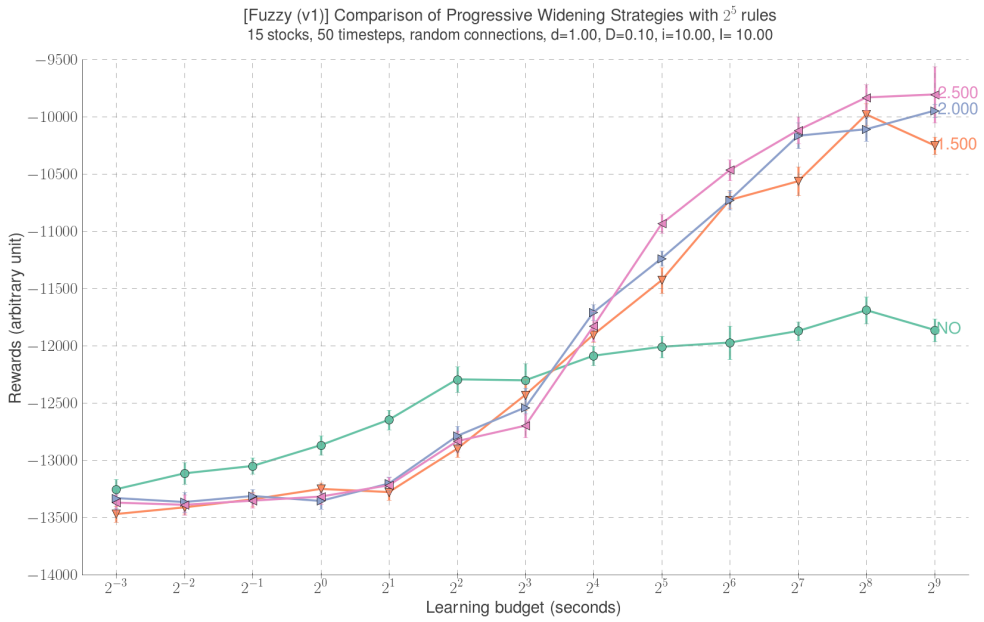


Figure 6.3 – Results with Eq. 6.1, large test case, logarithmic Sieves. "NO" indicates the results with no Sieves Method, while the numbers on the other curves indicates the rates used. We compare our Sieves method to the standard optimization with 32 optimized rules from the start. In this case, we can see a very noticeable increase of performances when using the Sieves Method compared to the vanilla process.

Budget (s)	No PW	rate = 0.20	rate = 0.25	rate = 0.30
1	**-1.29e+04** (±2.07e+02)	-1.32e+04 (±1.59e+02)	-1.32e+04 (±1.76e+02)	-1.32e+04 (±1.23e+02)
2	**-1.27e+04** (±1.77e+02)	-1.30e+04 (±2.39e+02)	-1.30e+04 (±2.12e+02)	-1.31e+04 (±1.47e+02)
4	**-1.24e+04** (±2.62e+02)	-1.26e+04 (±1.71e+02)	-1.27e+04 (±3.64e+02)	-1.28e+04 (±2.49e+02)
8	**-1.20e+04** (±3.09e+02)	-1.29e+04 (±2.42e+02)	-1.26e+04 (±6.13e+02)	-1.27e+04 (±3.80e+02)
16	-1.23e+04 (±3.49e+02)	-1.23e+04 (±2.88e+02)	**-1.21e+04** (±1.48e+02)	-1.23e+04 (±2.68e+02)
32	-1.21e+04 (±2.63e+02)	-1.22e+04 (±3.99e+02)	-1.22e+04 (±3.02e+02)	**-1.20e+04** (±2.52e+02)
64	-1.20e+04 (±2.85e+02)	-1.18e+04 (±1.31e+02)	-1.18e+04 (±3.20e+02)	**-1.17e+04** (±3.04e+02)
128	-1.20e+04 (±2.34e+02)	-1.13e+04 (±2.46e+02)	-1.14e+04 (±2.60e+02)	**-1.13e+04** (±2.89e+02)
256	-1.18e+04 (±2.95e+02)	-1.10e+04 (±3.06e+02)	-1.08e+04 (±4.16e+02)	**-1.06e+04** (±3.11e+02)
512	-1.17e+04 (±2.54e+02)	-1.04e+04 (±4.11e+02)	-1.03e+04 (±3.92e+02)	**-1.03e+04** (±2.96e+02)

Table 6.4 – Sieves Method with Linear increase on the Big case, with the membership function 6.4

Budget (s)	No PW	rate = 0.20	rate = 0.25	rate = 0.30
1	**-6.46e+03** (±8.46e+02)	-7.52e+03 (±4.99e+02)	-7.63e+03 (±6.55e+02)	-7.61e+03 (±5.66e+02)
2	**-6.09e+03** (±3.29e+02)	-6.94e+03 (±3.33e+02)	-7.06e+03 (±3.97e+02)	-7.04e+03 (±3.29e+02)
4	**-5.76e+03** (±2.59e+02)	-6.63e+03 (±4.14e+02)	-6.93e+03 (±4.02e+02)	-6.79e+03 (±3.56e+02)
8	**-5.55e+03** (±5.69e+02)	-6.50e+03 (±6.16e+02)	-6.35e+03 (±4.04e+02)	-6.55e+03 (±3.24e+02)
16	**-5.23e+03** (±3.03e+02)	-5.50e+03 (±3.12e+02)	-5.50e+03 (±2.59e+02)	-5.88e+03 (±5.07e+02)
32	**-4.91e+03** (±2.93e+02)	-4.94e+03 (±3.09e+02)	-5.20e+03 (±1.65e+02)	-5.12e+03 (±2.65e+02)
64	-4.86e+03 (±1.96e+02)	**-4.56e+03** (±1.97e+02)	-4.67e+03 (±2.28e+02)	-4.75e+03 (±1.48e+02)
128	-4.48e+03 (±6.82e+02)	**-4.24e+03** (±7.66e+02)	-4.38e+03 (±6.08e+02)	-4.29e+03 (±6.12e+02)
256	-4.35e+03 (±2.17e+02)	-3.90e+03 (±2.44e+02)	**-3.86e+03** (±3.19e+02)	-3.89e+03 (±3.52e+02)
512	-4.18e+03 (±3.28e+02)	-3.78e+03 (±2.86e+02)	-3.81e+03 (±3.25e+02)	**-3.61e+03** (±1.90e+02)

Table 6.5 – Sieves Method with Linear increase on the Small case, with the membership function 6.4

Budget (s)	No PW	rate = 1.5	rate = 2.0	rate = 2.5
1	**-6.46e+03** (±8.46e+02)	-7.52e+03 (±4.99e+02)	-7.63e+03 (±6.55e+02)	-7.61e+03 (±5.66e+02)
2	**-6.09e+03** (±3.29e+02)	-6.94e+03 (±3.33e+02)	-7.06e+03 (±3.97e+02)	-7.04e+03 (±3.29e+02)
4	**-5.76e+03** (±2.59e+02)	-6.63e+03 (±4.14e+02)	-6.93e+03 (±4.02e+02)	-6.79e+03 (±3.56e+02)
8	**-5.55e+03** (±5.69e+02)	-6.50e+03 (±6.16e+02)	-6.35e+03 (±4.04e+02)	-6.55e+03 (±3.24e+02)
16	**-5.23e+03** (±3.03e+02)	-5.50e+03 (±3.12e+02)	-5.50e+03 (±2.59e+02)	-5.88e+03 (±5.07e+02)
32	**-4.91e+03** (±2.93e+02)	-4.94e+03 (±3.09e+02)	-5.20e+03 (±1.65e+02)	-5.12e+03 (±2.65e+02)
64	-4.86e+03 (±1.96e+02)	**-4.56e+03** (±1.97e+02)	-4.67e+03 (±2.28e+02)	-4.75e+03 (±1.48e+02)
128	-4.48e+03 (±6.82e+02)	**-4.24e+03** (±7.66e+02)	-4.38e+03 (±6.08e+02)	-4.29e+03 (±6.12e+02)
256	-4.35e+03 (±2.17e+02)	-3.90e+03 (±2.44e+02)	**-3.86e+03** (±3.19e+02)	-3.89e+03 (±3.52e+02)
512	-4.18e+03 (±3.28e+02)	-3.78e+03 (±2.86e+02)	-3.81e+03 (±3.25e+02)	**-3.61e+03** (±1.90e+02)

Table 6.6 – Sieves Method with Linear increase on the Big case, with the membership function 6.1

Budget (s)	No PW	rate = 1.5	rate = 2.0	rate = 2.5
1	**-6.46e+03** (±8.46e+02)	-7.52e+03 (±4.99e+02)	-7.63e+03 (±6.55e+02)	-7.61e+03 (±5.66e+02)
2	**-6.09e+03** (±3.29e+02)	-6.94e+03 (±3.33e+02)	-7.06e+03 (±3.97e+02)	-7.04e+03 (±3.29e+02)
4	**-5.76e+03** (±2.59e+02)	-6.63e+03 (±4.14e+02)	-6.93e+03 (±4.02e+02)	-6.79e+03 (±3.56e+02)
8	**-5.55e+03** (±5.69e+02)	-6.50e+03 (±6.16e+02)	-6.35e+03 (±4.04e+02)	-6.55e+03 (±3.24e+02)
16	**-5.23e+03** (±3.03e+02)	-5.50e+03 (±3.12e+02)	-5.50e+03 (±2.59e+02)	-5.88e+03 (±5.07e+02)
32	**-4.91e+03** (±2.93e+02)	-4.94e+03 (±3.09e+02)	-5.20e+03 (±1.65e+02)	-5.12e+03 (±2.65e+02)
64	-4.86e+03 (±1.96e+02)	**-4.56e+03** (±1.97e+02)	-4.67e+03 (±2.28e+02)	-4.75e+03 (±1.48e+02)
128	-4.48e+03 (±6.82e+02)	**-4.24e+03** (±7.66e+02)	-4.38e+03 (±6.08e+02)	-4.29e+03 (±6.12e+02)
256	-4.35e+03 (±2.17e+02)	-3.90e+03 (±2.44e+02)	**-3.86e+03** (±3.19e+02)	-3.89e+03 (±3.52e+02)
512	-4.18e+03 (±3.28e+02)	-3.78e+03 (±2.86e+02)	-3.81e+03 (±3.25e+02)	**-3.61e+03** (±1.90e+02)

Table 6.7 – Sieves Method with Linear increase on the Small case, with the membership function 6.4

Budget (s)	No PW	rate = 1.5	rate = 2.0	rate = 2.5
1	**-6.46e+03** (±8.46e+02)	-7.52e+03 (±4.99e+02)	-7.63e+03 (±6.55e+02)	-7.61e+03 (±5.66e+02)
2	**-6.09e+03** (±3.29e+02)	-6.94e+03 (±3.33e+02)	-7.06e+03 (±3.97e+02)	-7.04e+03 (±3.29e+02)
4	**-5.76e+03** (±2.59e+02)	-6.63e+03 (±4.14e+02)	-6.93e+03 (±4.02e+02)	-6.79e+03 (±3.56e+02)
8	**-5.55e+03** (±5.69e+02)	-6.50e+03 (±6.16e+02)	-6.35e+03 (±4.04e+02)	-6.55e+03 (±3.24e+02)
16	**-5.23e+03** (±3.03e+02)	-5.50e+03 (±3.12e+02)	-5.50e+03 (±2.59e+02)	-5.88e+03 (±5.07e+02)
32	**-4.91e+03** (±2.93e+02)	-4.94e+03 (±3.09e+02)	-5.20e+03 (±1.65e+02)	-5.12e+03 (±2.65e+02)
64	-4.86e+03 (±1.96e+02)	**-4.56e+03** (±1.97e+02)	-4.67e+03 (±2.28e+02)	-4.75e+03 (±1.48e+02)
128	-4.48e+03 (±6.82e+02)	**-4.24e+03** (±7.66e+02)	-4.38e+03 (±6.08e+02)	-4.29e+03 (±6.12e+02)
256	-4.35e+03 (±2.17e+02)	-3.90e+03 (±2.44e+02)	**-3.86e+03** (±3.19e+02)	-3.89e+03 (±3.52e+02)
512	-4.18e+03 (±3.28e+02)	-3.78e+03 (±2.86e+02)	-3.81e+03 (±3.25e+02)	**-3.61e+03** (±1.90e+02)

Table 6.8 – Sieves Method with Linear increase on the Big case, with the membership function 6.4

Budget (s)	No PW	rate = 1.5	rate = 2.0	rate = 2.5
1	**-6.46e+03** (±8.46e+02)	-7.52e+03 (±4.99e+02)	-7.63e+03 (±6.55e+02)	-7.61e+03 (±5.66e+02)
2	**-6.09e+03** (±3.29e+02)	-6.94e+03 (±3.33e+02)	-7.06e+03 (±3.97e+02)	-7.04e+03 (±3.29e+02)
4	**-5.76e+03** (±2.59e+02)	-6.63e+03 (±4.14e+02)	-6.93e+03 (±4.02e+02)	-6.79e+03 (±3.56e+02)
8	**-5.55e+03** (±5.69e+02)	-6.50e+03 (±6.16e+02)	-6.35e+03 (±4.04e+02)	-6.55e+03 (±3.24e+02)
16	**-5.23e+03** (±3.03e+02)	-5.50e+03 (±3.12e+02)	-5.50e+03 (±2.59e+02)	-5.88e+03 (±5.07e+02)
32	**-4.91e+03** (±2.93e+02)	-4.94e+03 (±3.09e+02)	-5.20e+03 (±1.65e+02)	-5.12e+03 (±2.65e+02)
64	-4.86e+03 (±1.96e+02)	**-4.56e+03** (±1.97e+02)	-4.67e+03 (±2.28e+02)	-4.75e+03 (±1.48e+02)
128	-4.48e+03 (±6.82e+02)	**-4.24e+03** (±7.66e+02)	-4.38e+03 (±6.08e+02)	-4.29e+03 (±6.12e+02)
256	-4.35e+03 (±2.17e+02)	-3.90e+03 (±2.44e+02)	**-3.86e+03** (±3.19e+02)	-3.89e+03 (±3.52e+02)
512	-4.18e+03 (±3.28e+02)	-3.78e+03 (±2.86e+02)	-3.81e+03 (±3.25e+02)	**-3.61e+03** (±1.90e+02)

Table 6.9 – Sieves Method with Linear increase on the Small case, with the membership function 6.4

most cases, the higher the rate (so the slower the number of rules is added), the better the results are in the end but the longer it takes to achieve "good" results (something that can be seen in Table 6.3 for example: the rate 0.20 is the first to beat the vanilla version, but is then overcame by the rate 0.25, which in turn is beaten by rate = 0.30).

One might ask whether the improvement is because of the limited number of rules used during the optimization run, or if the SM of the number of rules provides an improvement. Additional experiments performed however show that even when there is a hard limit on the number of rules, there is still an improvement.

6.4 Conclusions and further work

Fuzzy control is a wide research area. The Sieves method is classical in statistics. We extended the Sieves method to noisy optimization, including experiments in fuzzy control. Results are clearly positive. The simple recommendation is that the number of rules should increase somewhere between logarithmically and linearly with the number of iterations. Results for early stages of the optimization are sometimes positive and sometimes not, but asymptotically in all cases the Sieves method provided better results. Seemingly, it avoids local minima.

The classical mathematical results for justifying the method of Sieves are based on VC-dimension arguments or other statistical complexity measure (see *e.g.* [Devroye et al., 1997, Shen and Wong, 1994]). They do not apply here, because it is hard to compute the VC-dimension or covering numbers of level sets of fitness functions involved in control problems [Vidyasagar, 1997]. Based on our experiments, we conjecture that a logarithmically increasing number of rules is a reasonable solution. The parameters had little impact on our results, provided the rule was logarithmic, starting at 1, and converging to a few dozen of rules within a reasonable time for the problem at hand.

Using this method, we can work without any hard constraint on the number of rules; just add them progressively during the optimization run, in an anytime manner.

Our work is limited to DE. Our choice of parameters for the Sieves index might have to be made adaptive - though we tested many rules and all of them were satisfactory under this "start with one rule and concavely add new rules, reaching a few dozen within time constants of the optimization problem". The artificial setting was tested with various initial step-sizes and several values of the problem parameter γ. The optimal parametrization is not always the same, but with a slow increase we almost always outperform the baseline if $\gamma > 1$ - which means that some parameters are more important than others and that we know which ones. We consider as a main goal the construction of "universal" rules for choosing the rate at which rules should be added; maybe validating our rule above on

more test-cases of refining it. Sieves method are classical in many fields; there is room for them in optimization.

Part II

Applications

Chapter 7

Comparing optimizers on a unit commitment problem

This chapter's content comes from the paper Berthier, V. (2015a). Comparing optimizers on a unit commitment problem. In *Artificial Evolution (EA2015)*. Springer Verlag. Its abstract was:

This paper compares several black-box optimization algorithms on a unit commitment problem. Compared to existing testbeds, this one provides several scales, is real-world, and none of the compared algorithms were created by the author of the testbed. Differential Evolution basically performs best overall, though not for all test cases.

7.1 Introduction

Several testbeds were provided by [Suganthan et al., 2005, Hansen et al., 2010a, Gould et al., 2003, Gallagher, 2016] for non-linear optimization. In the evolutionary computation community the most widely used might be [Suganthan et al., 2005]. We here propose an alternative set of experiments, on which we compare a set of optimizers.

7.2 Testbed

Our testbed has the following characteristics:

- It is a real problem, originally not designed for academic purpose. As a consequence, it has the same degree of partial separability as (at least some) real world problems.

- It includes several cases, with dimension ranging from 3 to several thousands. We should indeed include, later, bigger test-cases.
- It is restricted to direct policy search for power systems. This category of problems is definitely an important one; we do not claim that our results have some validity beyond this scope.
- We compared algorithms and implementations in a neutral manner. We have no special interest for one algorithm or another, we just try to find which algorithm we should recommend as default in our optimization platform.
- This family of problems has a huge economical (billions of dollars per year) and environmental impact.

Our testbed has the following parameters:

- Number of time steps.
- Number of stocks (number of state variables).
- Parameters for the inflows and demand and their variances, which are held constant over our experiments.

The objective function can be seen as:

$$Cost = -\sum_{t=0}^{T}(max(0, Demand_t - \sum_{s=0}^{n_stocks} Production_{s,t}) \times ThermalCost) - Penalisation$$

The penalization term is here to ensure that we do not produce too much electricity from the dam, since it would waste water (at the very least, in a real network there would be other consequences).

It would be impracticable to put a more detailed and in depth explanation on the test bed here, as a consequence we made it available on at `https://www.lri.fr/~teytaud/uctest/uctest.html`.

The number of decisions per time step is equal to the number of stocks (we decide how much water we use for each stock). The number of inputs for making each decision is one observation per stock (the level), plus the 4 calendar factors. Hence, the number of action variables is $nbActions = nbStocks$ and the number of input variables is $nbInputs = nbStocks + 4$.

The number of parameters for a given problem can be computed as follows (and the detailed policies can be seen at `https://www.lri.fr/~teytaud/uctest/uctest.html`):

- Handcrafted policy: the number of parameters is always $N = 3$.
- Conformant planning (a sequence of decisions, applied independently of observations): $N = T \times NbActions = T \times NbStocks$ (one parameter per time step and per stock).
- Neural network (feed-forward, one hidden layer), the number of parameters is $N = NbNeurons \times (NbInputs + NbActions + 1) + NbActions$:
  - $NbActions$ parameters for the biases for the output,
  - $NbInputs + 1$ for the input weights of each neuron,
  - and $NbActions$ connections between each hidden neuron and the output neurons).
- Fuzzy control: the number of parameters is $N = NbRules \times (2nbInputs + nbActions + 1)$ because each rule has $2NbInputs + 1$ parameters for the antecedent (one scale and one average value for each coordinate, plus one default rule weight) and $NbActions$ parameters for the succedent.

For example, with 25 stocks and 100 time-steps, the number of parameters are 3 for the handcrafted policy; 2500 for conformant planning; $2681 = 32(58+25)+25$ for fuzzy systems with 2^5 rules; $1785 = 32(29+25+1)+25$ for neural networks with 2^5 neurons.

The fuzzy rule used in the experiments uses a membership function product of coordinate-wise inverse distances. This was selected among various membership functions after preliminary experiments.

7.3 Summary of results

We summarize our results in tables below for 512 seconds of budget. Overall, DE performed best.

7.3.1 Overview and results per problem size

In Table 7.1 we give results averaged over all problems, and then for different sizes (< 10 parameters, 10 to 99, 100 to 999, and 1000+. Only the average performances of the algorithms are shown here to avoid an avalanche of tables, but the underlying study was performed based on average, worst, quartile and decile performances. We can see that in high dimension, CMA suffers due to its internal cost, as shown by the small number (relative to the other algorithms).

All problems average perf	
DE	0.79 +- 0.023
CMA	0.75 +- 0.035
SAiso	0.73 +- 0.023
PSO	0.66 +- 0.033
SA	0.66 +- 0.026
SAcov	0.66 +- 0.025
1+1	0.6 +- 0.027
NM	0.54 +- 0.036

Nb params $\leq$ 10 average perf	
PSO (1560k)	0.83 +- 0.092
DE (1545k)	0.82 +- 0.06
SAcov (1668k)	0.78 +- 0.064
CMA (1580k)	0.78 +- 0.054
SAiso (1580k)	0.78 +- 0.059
SA (1649k)	0.76 +- 0.077
NM (1548k)	0.67 +- 0.073
1+1 (1772k)	0.62 +- 0.071

10 $\leq$nb params $<$ 100 average perf	
DE (1622k)	0.87 +- 0.028
CMA (1660k)	0.82 +- 0.047
SAiso (1642k)	0.69 +- 0.04
PSO (1670k)	0.61 +- 0.04
1+1 (1629k)	0.59 +- 0.05
SAcov (1619k)	0.56 +- 0.056
SA (1618k)	0.55 +- 0.058
NM (1666k)	0.51 +- 0.046

100 $\leq$nbParams$<$ 1000 average perf	
DE (936k)	0.76 +- 0.035
CMA (641k)	0.75 +- 0.051
SAiso (932k)	0.74 +- 0.034
SA (926k)	0.68 +- 0.031
SAcov (875k)	0.68 +- 0.029
PSO (948k)	0.64 +- 0.053
1+1 (930k)	0.61 +- 0.041
NM (942k)	0.54 +- 0.056

1000 $\leq$nbParams$<$ 10000 average perf	
PSO (167k)	0.8 +- 0.088
SAiso (159k)	0.78 +- 0.064
SA (156k)	0.73 +- 0.056
DE (162k)	0.73 +- 0.075
SAcov (104k)	0.7 +- 0.054
CMA (5k)	0.59 +- 0.14
1+1 (159k)	0.58 +- 0.083
NM (152k)	0.54 +- 0.14

Table 7.1 – Each result is linearly normalized so that 1 is the maximum (best) result, and 0 is the minimum (worst) result over all runs for this controller and this unit commitment problem (so higher is better). The numbers between parenthesis are the number of fitness evaluations performed in the given budget of 512s.

7.3.2 Per family of controllers and per problem size

For each testbed, we specify with which frequency an algorithm (in row) outperforms another one (in column). These results in Tables 7.2 and 7.3 are the same as the results above, but broken down on the different test cases. Due to size constraints results on the large testbed are not shown, but they are essentially the same as the medium case, the only major difference is that PSO becomes the best algorithm on the Conformant Planning function.

7.4 Conclusions and further work

A short conclusion is that DE performs best overall, with also an excellent stability. This is consistent with the success of DE on several competitions - variants or combinations of DE have won the CEC 2006, CEC 2010 and CEC 2013 competitions [Das and Suganthan, 2011, LaTorre et al., 2013].

Still, there is no clear-cut conclusion; DE is a bit weaker with neural network controllers, and even algorithms which are usually not that stable (*e.g.* Nelder-Mead or $(1+1)$-ES) sometimes perform very well. In particular, the important special case of conformant planning is very well tackled by the simple $(1+1)$-ES.

PSO performed well in high dimensional problems. Nelder-Mead was surprisingly good in spite of long initialization (with a population linear in the dimension).

CMA performed very well in some cases, but was in general clearly outperformed by DE.Variants of CMA with limited covariance (*e.g.* diagonal) might be considered to alleviate the dimensional problem.

For sure, this work is not intended to be some kind of "final" comparison. This is one test case, with the advantage that it is a real world and (ecologically and economically) important test case. Besides the fact that our test cases can lead to different conclusions, we do not take into account the limit in terms of parallelization, whereas parallelization is one of the main body of work around PSO [McNabb et al., 2007, Mahdad et al., 2010, Schutte et al., 2003, Chang et al., 2005, Gardner et al., 2012].

The main further works are (i) including more algorithms (*e.g.* Newuoa [Powell, 2008], variants of DE and memetic algorithms) (ii) including noisy optimization (iii) parallel setting, *e.g.* constraining the population size to 1000 (iv) bigger test cases (we can without effort extend the test case to 100 stocks and 2000 time steps, which is consistent with some real world cases - unit commitment problems exist at various scales).

(a) DE outperforms everything for the specific policy

	SAiso	$(1+1)$	*SA*	*SACov*	*CMA*	*NM*	*DE*	*PSO*
SAiso		100.00	28.57	57.14	64.29	100.00	21.43	42.86
$(1+1)-ES$	0.00		0.00	0.00	7.14	14.29	7.14	0.00
$SA-ES$	71.43	100.00		57.14	71.43	92.86	14.29	50.00
$SA-ESCov$	42.86	100.00	42.86		78.57	92.86	21.43	50.00
$CMA-ES$	35.71	92.86	28.57	21.43		85.71	14.29	50.00
NM	0.00	85.71	7.14	7.14	14.29		0.00	0.00
DE	78.57	92.86	85.71	78.57	85.71	100.00		85.71
PSO	57.14	100.00	50.00	50.00	50.00	100.00	14.29	

(b) With the neural network, PSO is clearly the best algorithm

	SAiso	$(1+1)$	*SA*	*SACov*	*CMA*	*NM*	*DE*	*PSO*
SAiso		92.86	92.86	89.29	42.86	21.43	50.00	0.00
$(1+1)-ES$	7.14		46.43	50.00	35.71	10.71	3.57	0.00
$SA-ES$	7.14	53.57		75.00	28.57	10.71	10.71	0.00
$SA-ESCov$	10.71	50.00	25.00		28.57	10.71	10.71	0.00
$CMA-ES$	57.14	64.29	71.43	71.43		28.57	25.00	17.86
NM	78.57	89.29	89.29	89.29	71.43		78.57	3.57
DE	50.00	96.43	89.29	89.29	75.00	21.43		0.00
PSO	100.00	100.00	100.00	100.00	82.14	96.43	100.00	

(c) CMA is the best performing algorithm for Conformant Planning

	SAiso	$(1+1)$	*SA*	*SACov*	*CMA*	*NM*	*DE*	*PSO*
SAiso		100.00	71.43	71.43	0.00	42.86	7.14	100.00
$(1+1)-ES$	0.00		0.00	14.29	0.00	28.57	0.00	100.00
$SA-ES$	28.57	100.00		57.14	0.00	42.86	7.14	100.00
$SA-ESCov$	28.57	85.71	42.86		0.00	42.86	0.00	92.86
$CMA-ES$	100.00	100.00	100.00	100.00		100.00	64.29	100.00
NM	57.14	71.43	57.14	57.14	0.00		0.00	85.71
DE	92.86	100.00	92.86	100.00	35.71	100.00		100.00
PSO	0.00	0.00	0.00	7.14	0.00	14.29	0.00	

(d) For Fuzzy control, SA-iso is the best algorithm

	SAiso	$(1+1)$	*SA*	*SACov*	*CMA*	*NM*	*DE*	*PSO*
SAiso		100.00	100.00	100.00	100.00	100.00	92.86	100.00
$(1+1)-ES$	0.00		42.86	64.29	92.86	100.00	60.71	100.00
$SA-ES$	0.00	57.14		96.43	92.86	100.00	67.86	96.43
$SA-ESCov$	0.00	35.71	3.57		92.86	100.00	25.00	82.14
$CMA-ES$	0.00	7.14	7.14	7.14		71.43	0.00	28.57
NM	0.00	0.00	0.00	0.00	28.57		0.00	0.00
DE	7.14	39.29	32.14	75.00	100.00	100.00		96.43
PSO	0.00	0.00	3.57	17.86	71.43	100.00	3.57	

Table 7.2 – Frequency (in percentage) where an algorithm (in row) outperforms another one (in column) in the small case (5 stocks, 25 time-steps).

(a) DE outperforms everything for the specific policy

	SAiso	$(1+1)$	*SA*	*SACov*	*CMA*	*NM*	*DE*	*PSO*
SAiso		100.00	50.00	64.29	64.29	100.00	7.14	42.86
$(1+1)-ES$	0.00		0.00	0.00	0.00	42.86	0.00	0.00
$SA-ES$	50.00	100.00		64.29	42.86	92.86	0.00	28.57
$SA-ESCov$	35.71	100.00	35.71		28.57	92.86	0.00	14.29
$CMA-ES$	35.71	100.00	57.14	71.43		100.00	0.00	21.43
NM	0.00	57.14	7.14	7.14	0.00		0.00	7.14
DE	92.86	100.00	100.00	100.00	100.00	100.00		92.86
PSO	57.14	100.00	71.43	85.71	78.57	92.86	7.14	

(b) With the neural network, PSO is clearly the best algorithm

	SAiso	$(1+1)$	*SA*	*SACov*	*CMA*	*NM*	*DE*	*PSO*
SAiso		75.00	82.14	78.57	10.71	3.57	35.71	0.00
$(1+1)-ES$	25.00		46.43	60.71	10.71	3.57	46.43	0.00
$SA-ES$	17.86	53.57		75.00	7.14	0.00	17.86	0.00
$SA-ESCov$	21.43	39.29	25.00		7.14	0.00	25.00	0.00
$CMA-ES$	89.29	89.29	92.86	92.86		21.43	60.71	10.71
NM	96.43	96.43	100.00	100.00	78.57		100.00	25.00
DE	64.29	53.57	82.14	75.00	39.29	0.00		7.14
PSO	100.00	100.00	100.00	100.00	89.29	75.00	92.86	

(c) DE is the best performing algorithm for Conformant Planning

	SAiso	$(1+1)$	*SA*	*SACov*	*CMA*	*NM*	*DE*	*PSO*
SAiso		100.00	64.29	71.43	64.29	92.86	35.71	64.29
$(1+1)-ES$	0.00		0.00	21.43	21.43	57.14	0.00	50.00
$SA-ES$	35.71	100.00		64.29	64.29	92.86	35.71	57.14
$SA-ESCov$	28.57	78.57	35.71		42.86	78.57	7.14	64.29
$CMA-ES$	35.71	78.57	35.71	57.14		64.29	28.57	42.86
NM	7.14	42.86	7.14	21.43	35.71		7.14	21.43
DE	64.29	100.00	64.29	92.86	71.43	92.86		64.29
PSO	35.71	50.00	42.86	35.71	57.14	78.57	35.71	

(d) For Fuzzy control, SA-iso is the best algorithm

	SAiso	$(1+1)$	*SA*	*SACov*	*CMA*	*NM*	*DE*	*PSO*
SAiso		89.29	89.29	100.00	100.00	100.00	89.29	96.43
$(1+1)-ES$	10.71		21.43	64.29	100.00	100.00	53.57	60.71
$SA-ES$	10.71	78.57		82.14	100.00	100.00	85.71	78.57
$SA-ESCov$	0.00	35.71	17.86		100.00	100.00	46.43	42.86
$CMA-ES$	0.00	0.00	0.00	0.00		42.86	0.00	7.14
NM	0.00	0.00	0.00	0.00	57.14		0.00	0.00
DE	10.71	46.43	14.29	53.57	100.00	100.00		53.57
PSO	3.57	39.29	21.43	57.14	92.86	100.00	46.43	

Table 7.3 – Frequency (in percentage) where an algorithm (in row) outperforms another one (in column) in the medium case (15 stocks, 50 time-steps).

Chapter 8

Combining policies: the best of human expertise and neurocontrol

This chapter's content comes from the paper Berthier, V., Couëtoux, A., and Teytaud, O. (2015). Combining policies: the best of human expertise and neurocontrol. In *Artificial Evolution 2015*, pages To–appear. Its abstract was:

We consider sequential decision making in the case where a generative model and a parametric policy are available. Such a framework is naturally tackled with Direct Policy Search, i.e. parametric op-timisation over simulations. We propose a simple method that combines this parametric policy with a more generic neural network, where all parameters are trained simultaneously. As such, our approach doesn't require any computational overhead. We show that the resulting policy significantly outperforms both the domain specific policies and the neural network on a unit commitment test problem.

8.1 Introduction

In this chapter, we study planning under uncertainty, where only a generative model of the domain is available. We do not make any assumption on the inner dynamics of the problem. Instead, we assume that we have some prior knowledge, in the form of handcrafted parametric policies. These policies represent the existing methods to solve a problem. They can be optimal solutions of a simplified version of the problem, or simply human experience.

The constants in those parametric policies are replaced by parameters optimized on simulations. This is Direct Policy Search, also known as Simulation-Based optimization. More precisely, this is Direct Policy Search on top of expert policies; of course, Direct Policy Search can also be applied on top of generic policies such as neural networks or fuzzy

rules. As Direct Policy Search rarely provides a gradient and needs a lot of robustness, it is usually optimized by evolutionary algorithms.

This approach is stable and efficient. It is particularly convenient when an expert policy is available [Bengio, 1998]. However, in that case, it is limited by the structure of the policy.

To combine and exploit existing solvers, portfolios are now a widely established principle. They are used in combinatorial optimization [Nudelman et al., 2004, Hamadi, 2013] and noisy optimization [Baudis and Posik, 2014], including applications to control [Gagliolo, 2010].

In this work, we propose a simple method for combining parametric policies in a direct policy search framework. In contrast to portfolios as in [Gagliolo, 2010], our solution not only selects the best of several policies but also in some cases vastly outperforms each of them, without computational overhead. We perform experiments on a unit commitment problem, a kind of power system management problem where the goal is to optimize the cost of energy production.

The following sections briefly review discrete time controls methodologies, surveys methods aimed at combining policies, and presents the concept of orthogonality in portfolios, which will be central in our work.

8.2 Background and notations

With states noted $x \in \mathscr{X}$ and actions $u \in \mathscr{U}$, we assume a generative model is available, *i.e.* given (x,u), we can sample a resulting state $x' = f(x,u)$ and reward $r = \rho(x,u,x')$. f is the transition function and follows an unknown random distribution (e.g. $x' = f(x,u)$ depends on some random ω through $x' = f(x,y,\omega)$).

A policy π is an object that given a state x, returns an action u. It can be deterministic or stochastic, a parametric function or a qualitative heuristic.

Note that if the problem is non-Markovian, optimal policies might require to include the entire history of observation in the state variable x, making methods sensitive to the size of the state space highly impractical [Astrom, 1965].

The objective is to find a policy that maximizes the expected reward over a finite horizon T. Formally, given an initial state x_0, we try to find the solution π^* to

$$\underset{\pi}{\operatorname{argmax}}\, \mathbb{E}\left[\sum_t \rho(x_t, \pi(x_t), f(x_t, u_t))\right] \tag{8.1}$$

with $x_{t+1} = f(x_t, \pi(x_t))$ for $0 \leq t < T$.

8.2.1 Methodologies based on value functions

To find the optimal policy, the favourite methods in power systems applications (*e.g.* the management of long term hydroelectricity storage en production), come from Dynamic programming [Bellman, 1957] (DP) which is at the origin of a wide family of discrete-time control algorithms such as Stochastic Dual Dynamic Programming [Pereira and Pinto, 1991], Approximate Dynamic Programming [Powell, 2007], value iteration and a wide family of reinforcement learning algorithms.

Despite their solid theoretical basis, they are computationally expensive, they cannot directly handle large scale non-Markovian random processes, and they are usually not anytime algorithms (*i.e.* they return an incomplete answer if interrupted before termination).

Because of this, they are often less efficient than simpler deterministic approaches [Zambelli et al., 2011, Christophe et al., 2014].

8.2.2 Direct Policy Search

Another trend in control is Direct Policy Search (DPS), which consists in searching in the policy space directly, without any proxy.

This is often done by defining a set of parametric policies that depend on some parameter vector θ. One needs to find an optimal θ^*, so that π_{θ^*} is a solution to Eq. 8.1.

The search for a good parameter θ can be done in a noisy optimization framework, by relying on direct simulations of candidate policies π_θ on the test problem.

Various algorithms have been proposed, including evolutionary algorithms with re-sampling numbers chosen by Bernstein races [Heidrich-Meisner and Igel, 2009] or by simple re-sampling rules [Astete Morales et al., 2013]. They are often improved by the use of common random numbers [Strens and Moore, 2001, Strens et al., 2002, Kleinman et al., 1999].

The performance of parametric DPS heavily relies on the choice of the policy search space, i.e. the chosen class of policies that can be considered as candidates. Examples include neural networks [Bengio, 1998] and fuzzy systems [Zadeh, 1990], usually optimized by evolutionary algorithms [Stalph et al., 2008].

We use in this chapter a self-adaptive evolution strategy, with anisotropic step-size [Beyer, 2001]. The population size is set to $\lambda = 4N + 4$ where N is the number of parameters, and $\mu = \lambda/4$. The mutation rate is $\tau = 1/\sqrt{2N}$. Initial parameters are randomly drawn with a Gaussian distribution with step size 1 and step-sizes are independently randomly drawn as the exponential of standard Gaussian distributions.

Other algorithms were not considered here, since the goal was to assess the viability of the combination of two different strategies, not to compare the performances of each individual algorithms in that situation.

8.3 Meta-policy search

To find an optimum solution, it is of course possible to try each of the policies, and select the best one. This however implies to run the optimization process multiple times. Here, we propose a scheme to combine multiple policies: one is problem specific under the form of simple heuristics designed using prior knowledge on the domain, and the other one is a generic parametric policy (*eg.* Neural Network, Fuzzy rules).

8.3.1 Combining policies

Combining several policies has been done before, in different ways. A part of the literature combines policies in the sense that each policy, equipped with state prediction, handles a part of the state space [Doya and Samejima, 2002].

Some approach combine policies based on their Q-functions [Marivate and Littman, 2013] or by combining the policies themselves [van Hasselt, 2011]. Another method is to distribute the computational power over a family of algorithms (similarly to how multi-armed bandits distribute arm pulls) by combining DPS algorithms [Gagliolo, 2010].

As in [Gagliolo, 2010], we consider a DPS-based approach. More precisely, we consider several parametric policies, to be optimized by DPS. However, instead of optimizing each family separately, and then combining them, we consider a parametric combination $\alpha C_1 + (1-\alpha)C_2$ where, C_1 and C_2 are two policies. We then optimize the joint policy. With this, the decision resulting from the joint policy is the combination of each policy's output.

More formally, given a current state s, we select the decision:

$$C_{combination}(s) = \alpha C_1(s) + (1-\alpha)C_2(s). \tag{8.2}$$

This makes sense in the case of continuous actions. The number of parameters to optimize is $N_1 + N_2 + 1$, where N_1 and N_2 are the number of parameters of C_1 and C_2 respectively. We actually write α as a parametric function ranging from 0 to 1, with $\alpha = \frac{1}{2}(1+\beta/\sqrt{\beta^2+1})$; the parameter β is optimized in $\mathbb{R}$ and initialized at 0.

Our method has the following advantages:

- there is almost computational overhead in terms of computational cost per iteration, as all the parameters of the combined policy are trained at once, without specific training of each independent policy. Most of the computation time is spent in the simulations, not in the policies themselves. Therefore the computational overhead for a given number of iterations, compared to each of the families separately, is negligible.

- By searching in a wider set of functions and not only the functions encoded by method A or by method B, but instead all combinations of functions in A and functions in B, we can outperform all the individual policies, as the global family of functions contains weighted averages of the original policies and not only the union of both families of functions.

8.3.2 Orthogonal policies

[Samulowitz and Memisevic, 2007] pointed out the importance of using "orthogonal" algorithms in a portfolio. A portfolio containing too many optimizers tends to be unstable. It is then necessary to choose as few optimizers as possible, while covering as best as possible the set of all possible solvers. In order to increase the chances of finding a good solution, what matters is not (only) the number of optimizers in the portfolio, but how many orthogonal these optimizers are. Optimizers are said to be "orthogonal" if they are "very" different one to each other.

In the same way, combining many policies, or two policies of the same type (*eg.* neural networks) is not optimal. The best strategy would be to choose two policies as different from one another as possible.

This principle of orthogonality can be linked to the notion of diversity present in Ensemble Learning.

8.4 Experimental results: combining handcrafted functions and neural networks

To analyze our method, we designed many individual policies to later combine them. These include:

- A handcrafted function based on heuristics, designed by human experts.
- Several fuzzy control functions.
- Conformant planning: a sequence of actions, independent of state observations.
- A one layer feed-forward neural network with sigmoid activation functions, such that for a state at time t s_t, the action a_t is:

  $$a_t = W_0 + W_1 \times tanh(W_2 \times s_t + W_3)$$

  Where W_0 is a bias vector of size the number of actions at each time step, W_1 the activation weights of the neurons in the hidden layer of dimension $\|actions\| \times$

$\|neurons\|$, W_2 is the weight matrix from the states to the neurons of dimension $\|neurons\| \times \|states\|$, and W_3 is a bias vector the size of the number of neurons. The total number N of parameters to optimize is then

$$N = \|states\| \times \|neurons\| + \|neurons\| + \|actions\| \times \|neurons\| + \|actions\|$$

Fuzzy systems and conformant planning are intermediates between expert handcrafted functions and neural networks:

- They are less specialized than the expert function, which has only 3 parameters and works quite well.
- They are less parameter-free learners than the neural network.

Typically in our experiments the expert function is the best one for small learning time, and the neural network is the best function asymptotically. Interestingly however, we will see that our combination not only selects the best among the neural network and the expert function - it outperforms both.

8.4.1 Test problems: two types of unit commitment

Our test case is the one provided freely at `https://www.lri.fr/~teytaud/uctest/uctest.html`. In the unit commitment problem, the goal is to use available means of storage and production to satisfy a demand in energy over a given time horizon. We consider the case where energy can be produced from hydroelectric plants for free and from thermal plants at a cost.

Energy can be stored until a certain limit in hydroelectric plants. The goal being to minimize the costs, we want to use the thermal plants as little as possible, and to maximize the efficiency of the storage available, while still meeting the demand. Failures to produce the required demand are heavily penalized.

The objective function can be seen as:

$$Cost = -\sum_{t=0}^{T}(max(0, Demand_t - \sum_{s=0}^{n_stocks} Production_{s,t}) \times ThermalCost) - Penalisation$$

We study our method on two distinct versions of this problem: a hydroelectric valley (all dams are connected in series), and a random network of dams avoiding cycles.

In both cases, there are five dams, i.e. the state space contains 5 continuous variables.

There are 21 time steps. Thermal units complete the dispatch, *ie.* they produce the electricity needed to satisfy the demand. In short, the control problem has 5 input variables, 5 output variables, and 21 time steps.

The dams receive random inflows at each time step, simulating weather conditions. We study two cases: rainy seasons with large inflows, and dry seasons with small inflows.

The noise-free setting corresponds to a case in which we assume that all random processes can be predicted with high accuracy.The noisy setting represents a more difficult scenario. We need a noisy optimization algorithm instead of a classical optimization algorithm. In the noisy case, each fitness evaluation at iteration i of the evolutionary algorithm is averaged over $\lceil 10\sqrt{i} \rceil$ runs in order to mitigate the level of noise [Astete Morales et al., 2013].

8.4.2 Noise free setting

We first present experiments in a simplified noise-free case, *i.e.*the objective function is deterministic. This means that all random processes are replaced by a deterministic simplified counterpart. Results are presented in Fig. 8.1 (hydroelectric valley in the noise-free case; top: large inflows; bottom: small inflows) and Fig. 8.2 (hydroelectric network in the noise-free case, same two settings).

In each of these four noise-free cases, the combination is at least as efficient as each policy separately, and in two cases it outperforms them vastly. Each experiment is reproduced with various numbers of neurons; 2 or 4 neurons is usually optimal.

8.4.3 Noisy setting

We now perform experiments with random noise around the mean inflows and demands. Fig. 8.3 presents the results in the case of the hydroelectric valley and Fig. 8.4 presents the results in the case of the hydroelectric random network (in both cases, two settings, namely large inflows and small inflows).

In each of these four noisy cases, the combination is at least as efficient as each policy separately, and in two cases it outperforms them vastly. Each experiment is reproduced with various numbers of neurons; 2 or 4 neurons is usually optimal.

8.4.4 Experimental results

The results of the combination vastly outperform each individual policy in some cases: in the noise-free setting with large inflows (top plot of Figures 8.1 and 8.2) as well as in the noisy setting with, again, large inflows (top plots of Figures 8.3 and 8.4).

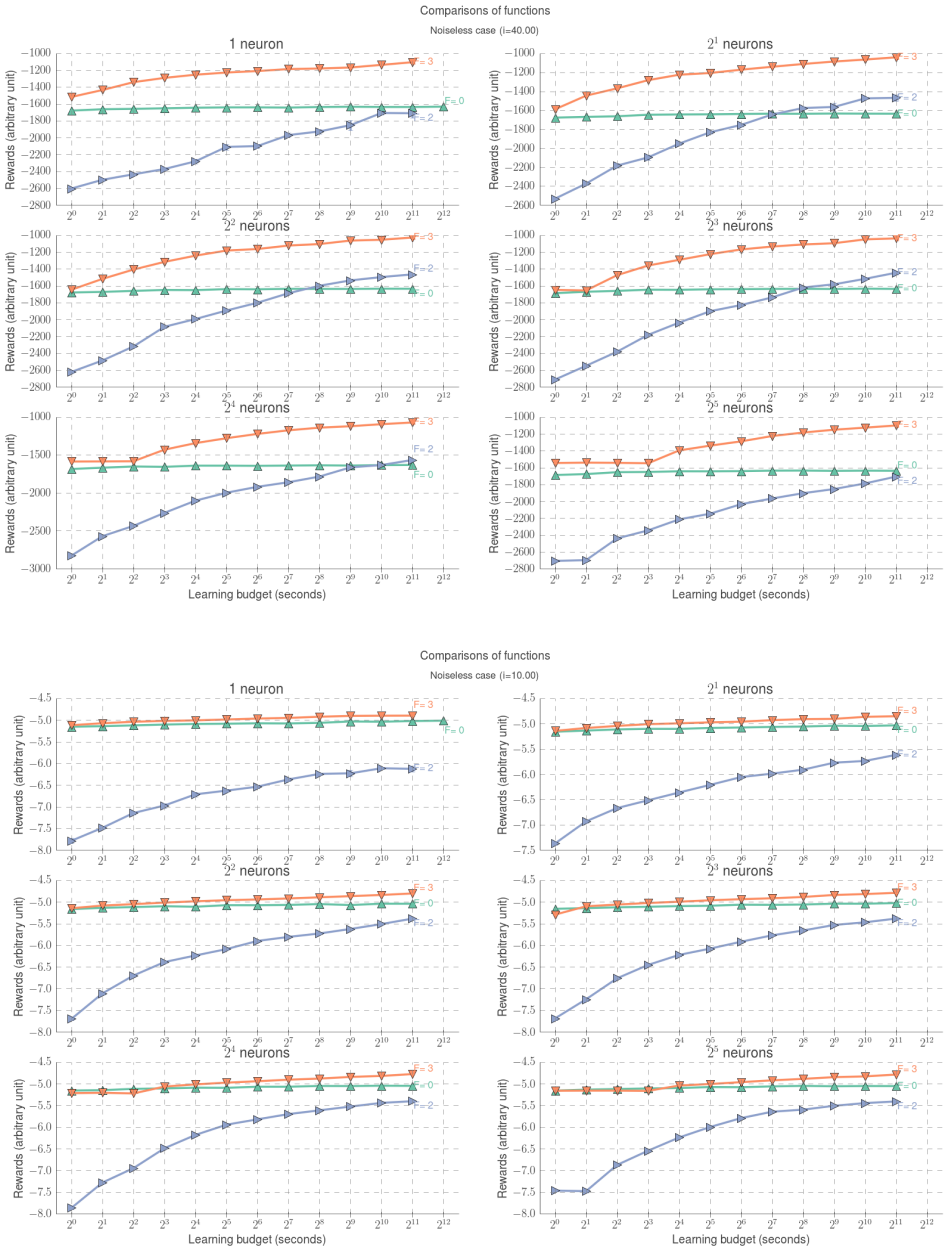


Figure 8.1 – Y-axis = reward. X-axis = learning budget. Hydroelectric valley. Noise-free setting (*i.e.*all random processes are simplified to their average values). Each subplot corresponds to a different number of neurons. $\triangle$: parametric expert function. $\triangleright$: neural network. $\bigtriangledown$: combination. The combination outperforms both separate functions. Top: large inflows. Bottom: small inflows.

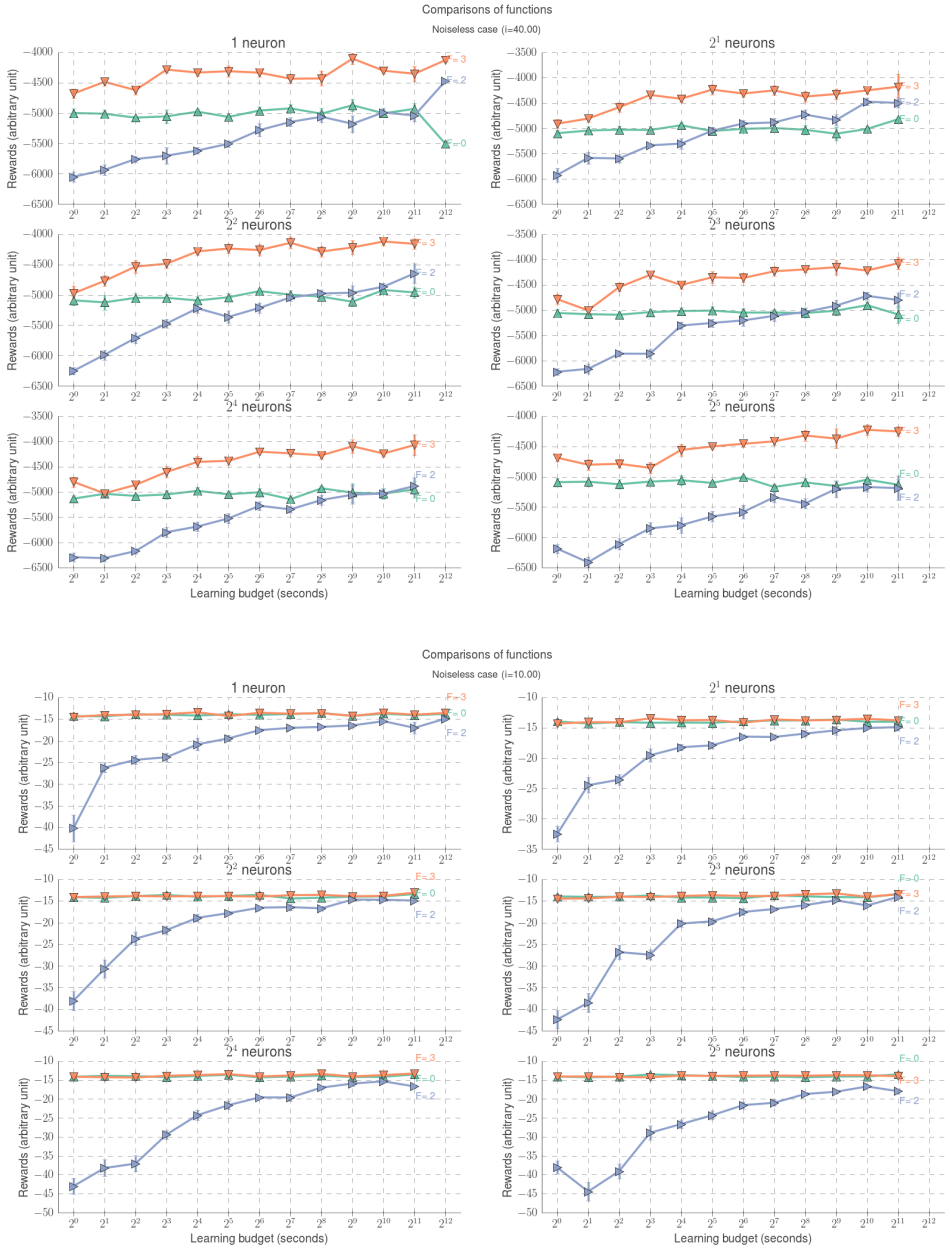


Figure 8.2 – Hydroelectric network, noise-free setting. Top: large inflows. Bottom: low inflows. The combination (▷) is a clear success in this case as well, though in the latter case the expert function also performs very well.

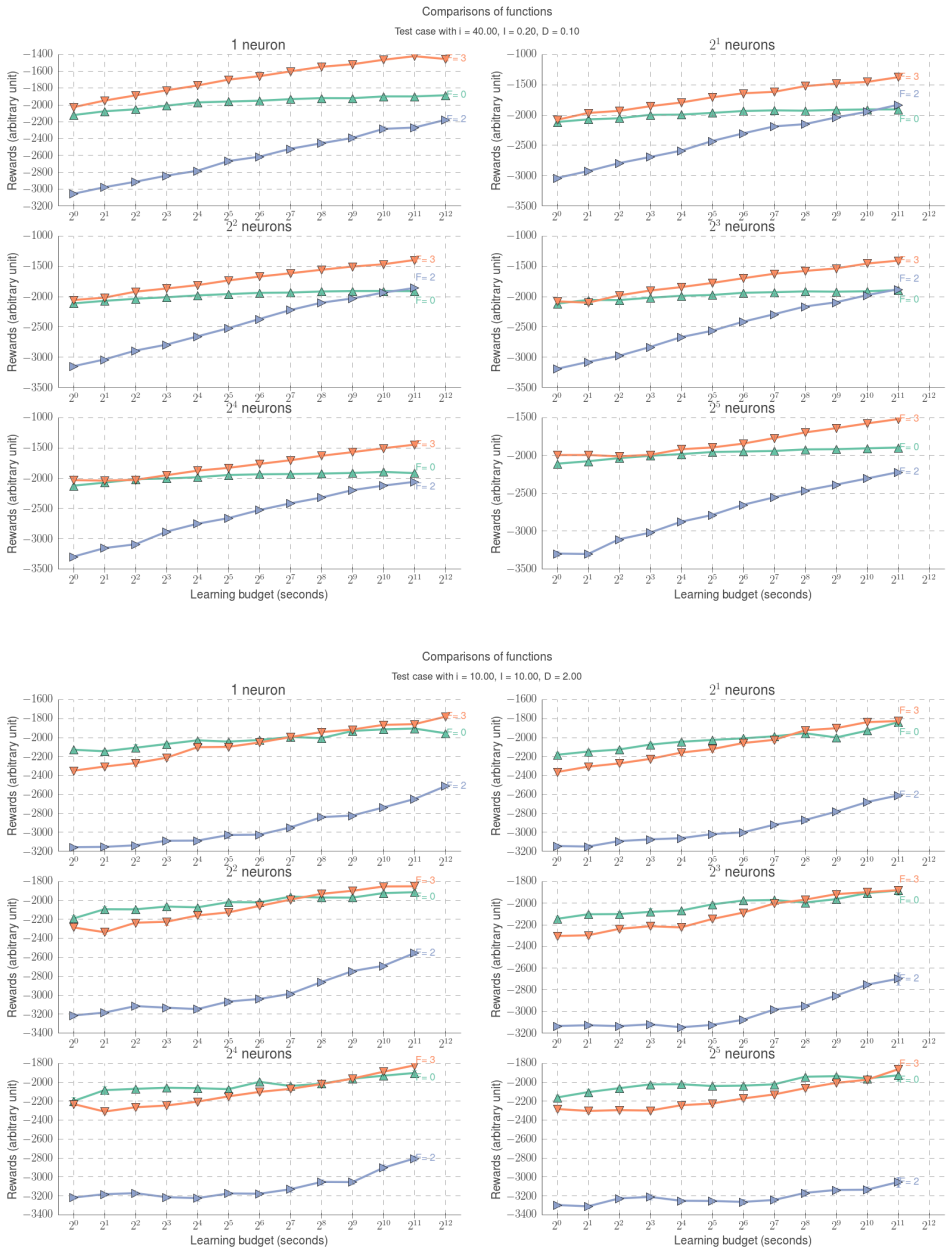


Figure 8.3 – Noisy setting, hydroelectric valley. Top: large inflow - the combination is excellent. Bottom: small inflows - the combination performs well; it does not always outperform the best of both solvers, but we point out that just selecting the best of two controllers takes more time than training them [Gagliolo, 2010].

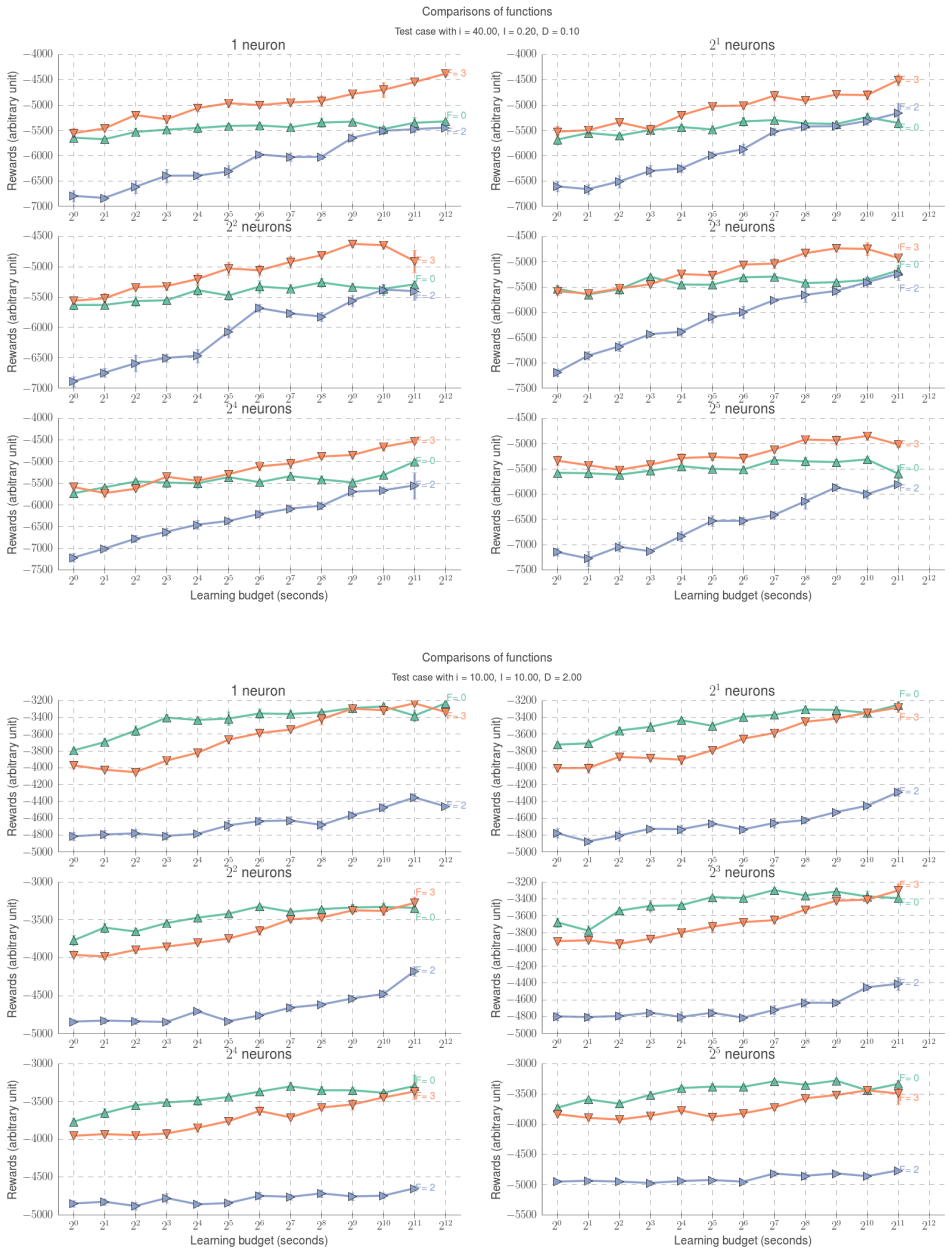


Figure 8.4 – Noisy setting, hydroelectric network. Top: large inflows. Bottom: small inflows. Results are qualitatively similar to Fig. 8.3.

In all the other cases, when there is little to no improvement though, it is important to note that the results aren't any worse than the best of the two other policies.

Considering the very low cost of optimizing the combination in opposition to one policy or the other, and the fact that in the worse case scenario the performances will be as good as a single policy, and that in the best case scenario the combination's results will be vastly better, designing and optimizing the combination policy seem like a good idea.

8.4.5 Experimental results: others

We also tried to replace the neural network policy by some other parametric policies such as fuzzy controllers, conformant planning, linear or quadratic controllers. However, none of them could be combined with the expert policy as efficiently as the neural network could.

Even more interestingly, when we combined two parametric policies, we could at best approximately get the best of the two (or four in cases of recursive combinations) but we never outperformed it. Furthermore, there was a clear delay to reach this selection, which is a result comparable to [Gagliolo, 2010].

8.5 Conclusions and further works

We proposed a simple tool for combining parametric DPS policies:

- Just one optimization pass for both policies (though we might consider more than two parametric policies);
- Usually quickly as good as the best of the considered policies;
- Sometimes much better.

Compared to separate learning, this makes the tool simpler (just one run) and faster (no separate learning). Compared to algorithm selection methods as [Gagliolo, 2010], we can outperform both approaches, whereas classical algorithm selection can only be equivalent to the best of the two methods.

We do not claim that we outperform portfolio methods, or at least not in all cases. Maybe for combining large numbers of policies our method would fail compared to portfolio methods. A limitation of our approach is that we can combine various parametric policies, but we can not combine DPS and completely different methods such as stochastic dual dynamic programming [Pereira and Pinto, 1991] or Monte Carlo Tree Search [Coulom, 2006, Kocsis and Szepesvari, 2006].

In addition, the success of our method was not reproduced with something else than the combination "expertise + neural networks"; we assume that this is related to the orthogonality (the expert policy is very different from the generic neural network). Still, the combination was very efficient in a stable manner, outperforming both methods without additional cost and without sophisticated developments. This was the case for $1, 2, 4, 8, 16, 32$ neurons, in all 8 sets of experiments (a deterministic and a stochastic case; a hydroelectric valley and a hydroelectric random network; and two levels of inflows). Therefore we consider that our simple combination (Eq. 8.2) should at least be considered when combining policies.

Last, we point out a specific property of evolution strategies. In the case where only one of the policies is relevant, then an optimization algorithm (evolutionary or not) might quickly find the optimal extreme value for α in Eq. 8.2. Then, the variables from the other policy have no impact on the objective function anymore, due to the weight zero of the corresponding policy. As a consequence, many variables become pointless, with no impact on the objective function. In contrast to many optimization algorithms, many evolutionary algorithms are not impacted by the presence of these pointless variables. Therefore, once α has been tuned, the evolutionary algorithm might just optimize the parameters of the relevant policy.

Combining four controllers was briefly considered in this work, without clear results. We considered combinations of controllers with less orthogonality (fuzzy systems, conformant planning, linear controllers) and results were far less convincing; whereas for neural networks and handcrafted policies the combination was already efficient. Extending the method in cases with less orthogonality might be interesting, as well as validating the fact that orthogonality is crucial.

Chapter 9

Progressive Differential Evolution on Clustering Real World Problems

This chapter's content comes from the paper Berthier, V. (2015c). Progressive differential evolution on clustering real world problems. In *International Conference on Artificial Evolution (Evolution Artificielle)*, pages 71–82. Springer. Its abstract was:

In this paper, we assess the performances of Differential Evolution on real-world clustering problems. To improve our results, we introduce Progressive Differential Evolution, a small modification of Differential Evolution which aims at optimizing a small number of parameters (eg. one cluster) at the beginning, and incrementally increase the number of optimized parameters.

9.1 Introduction

While many benchmarks used in the optimization community to evaluate algorithms are based on purely artificial functions such as [Suganthan et al., 2005] and [Hansen et al., 2010b], it can only be the first step in what ultimately is aimed at solving real world problems. Some recent initiatives went in that direction (see [Gould et al., 2003] for example), proposing new ways to assess the performances of optimization algorithms.

In this chapter, by comparing our results on one such benchmark, we (i) show that the Differential Evolution algorithm is very efficient on clustering problems and (ii) propose Progressive Differential Evolution, which starts with a low number of parameters to optimize and gradually increases it.

Section 9.2 describes the benchmark we used to compare our results to other algorithms and Section 9.3 validates our approach. Section 9.4 recalls the Differential Evolu-

tion algorithm and the "DE/curr-to-best/1" variant we used while Section 9.5 introduces Progressive Differential Evolution. In Section 9.6 we compare our results to the state of the art.

9.2 Continuous Real-World Representative benchmark

Most of the existing testbeds used to evaluate optimization algorithm compare their performances on artificial functions, such as the sphere, the ellipsoid or the Rosenbrock function to cite the most notable ones. With the improvements of the algorithms, more complex functions were introduced with some specific properties such as rotation, non separability, multimodality and so on, but ultimately, most testbeds are completely artificial.

While this is by no mean uninteresting, the ultimate goal in optimization is to solve real world problems. The gap between artificial functions - as complex as they are - to real world issues seems too large to directly apply what we know. As such, new testbeds, with some real world properties are advisable.

Knowing that using evolution strategy optimizers on clustering problems is perfectly viable, as shown in [Raul Hruschka et al., 2009], a new testbed revolving around clustering problems with interesting problems is proposed in [Gallagher, 2016]: those problems have interesting properties to evaluate optimization algorithms: challenging, scalable, easy to understand and implement, and most of all, their data can - and should - come from real world examples. Each cluster is used as a vector of coordinates in the parameters' space of data, which allows us to use optimization algorithms on those problems.

The three problems used here are the Iris [Fisher, 1936], the Ruspini [Ruspini, 1970] and the German Town [Spath, 1980] data-sets, all of them widely used in the clustering community to evaluate the performances of their own algorithms, and rooted in the real world. More importantly, [du Merle et al., 1999] computed the global optimum for those data-sets from two to ten clusters, which allows us to assess the performances of the algorithms. The German Town points are defined in 3D, the Iris ones in 4D and for Ruspini it is in 2D.

Along with a k-means clustering algorithm, [Gallagher, 2016] studied the performances of three black-box algorithms: CMA-ES [Hansen and Ostermeier, 2001] (one with standard population size, one with an increased population), Nelder-Mead [Nelder and Mead, 1965] and Random-Search. One of the conclusions is that even if the k-means algorithm converges very quickly, it is often beaten by CMA-ES (with increased population size) in term of quality of the solution found. Thus, complete black-box algorithms are able to outperform problem specific ones.

D	k	$f*$	CMA-ES(50,100)	CMA-ES(50,100) SP1	CMA-ES	CMA-ES SP1
G	02	6.02546e11	6.025472e11 (2.8e-04)	8.3400e03 (6.4e02)	1.172558e12 (7.6e11)	4.8798e04 (3.3e04)
	03	2.94506e11	4.486461e11 (1.5e11)	4.2083e04 (2.6e04)	8.196432e11 (1.2e12)	∞
	04	1.04474e11	3.362127e11 (1.4e11)	3.9970e05 (1.8e05)	7.629370e11 (4.7e11)	∞
	05	5.97615e10	2.049802e11 (1.4e11)	6.8410e05 (2.0e05)	7.488858e11 (1.2e12)	∞
	06	3.59085e10	1.585765e11 (1.5e11)	∞	8.818792e11 (6.3e11)	∞
	07	2.19832e10	1.051648e11 (1.1e11)	∞	6.463187e11 (7.5e11)	∞
	08	1.33854e10	1.068587e11 (9.3e10)	∞	7.005948e11 (7.1e11)	∞
	09	7.80442e09	2.780667e11 (3.1e11)	∞	1.003192e12 (9.7e11)	∞
	10	6.44647e09	5.869352e11 (5.2e11)	∞	7.677317e11 (6.5e11)	∞
I	02	1.52348e2	1.523480e02 (6.4e-14)	1.8344e04 (5.0e02)	1.523542e02 (3.0e-03)	∞
	03	7.88514e01	7.885144e01 (2.5e-14)	7.2048e04 (2.2e03)	1.279512e02 (1.2e02)	∞
	04	5.72285e01	5.836730e01 (3.9e00)	∞	9.728522e01 (3.5e01)	∞
	05	4.64462e01	4.766177e01 (1.7e00)	∞	1.330878e02 (1.3e02)	∞
	06	3.90400e01	4.149195e01 (2.9e00)	∞	1.292478e02 (1.3e02)	∞
	07	3.42982e01	4.037920e01 (3.5e00)	∞	7.892632e01 (4.5e01)	∞
	08	2.99889e01	3.739813e01 (4.2e00)	∞	7.750688e01 (5.4e01)	∞
	09	2.77861e01	3.831817e01 (5.3e00)	∞	8.018775e01 (7.6e01)	∞
	10	2.58341e01	5.653196e01 (6.9e01)	∞	9.553900e01 (1.0e02)	∞
R	02	8.93378e04	8.933783e04 (5.0e-12)	6.8260e03 (1.1e03)	8.933783e04 (3.1e-11)	3.5903e04 (5.0e03)
	03	5.10635e04	5.110393e04 (4.6e01)	2.0453e04 (5.3e03)	5.473043e04 (9.8e03)	∞
	04	1.28811e04	1.288105e04 (0.0e00)	∞	2.046652e04 (1.5e04)	∞
	05	1.01267e04	1.032449e04 (5.0e02)	∞	3.209521e04 (1.4e04)	∞
	06	8.57541e03	8.919118e03 (5.1e02)	2.5490e05 (1.7e04)	2.605724e04 (1.3e04)	∞
	07	7.12620e03	7.634386e03 (4.4e02)	7.7641e05 (4.9e04)	2.309534e04 (6.1e03)	∞
	08	6.14964e03	6.635902e03 (3.9e02)	∞	2.061007e04 (5.2e03)	∞
	09	5.18165e03	7.464273e03 (3.6e03)	∞	1.906988e04 (5.3e03)	∞
	10	4.44628e03	1.095691e04 (5.0e03)	∞	1.696298e04 (5.6e03)	∞

Table 9.1 – Average fitness results and SP1 measure (mean and standard deviation) for CMA-ES and CMA-ES(50,100). An SP1 measure of ∞ means that the optimum could not be reached for any of the 50 runs. Results are give for the German Town (G), Iris (I) and Ruspini (R) data-sets for all values of k.

9.3 Implementation validation

In order to compare results obtained on our platform using Evolving Objects (see [Keijzer et al., 2002]), we ran the benchmark on two CMA-ES with the same configuration as [Gallagher, 2016]: one has default parameters, one has a population size of $\mu = 50$ and $\lambda = 100$. In both cases, we stopped a run when $f_{best} \leq f^* + \frac{f^*}{1e15}$ (*ie.* the optimum is reached), when the best fitness stagnated for too long or when the allocated budget was consumed. This budget was set to $2e5$ function evaluations (all budgets in this chapter are expressed in terms of function evaluations).

As can be seen in Table 9.1, the mean fitnesses we were able to obtain are comparable to the ones reported in [Gallagher, 2016]: sometimes better, sometimes worse, but never

by far (except in high dimension where the results are degraded, probably due to different parameters). This allows us to validate our implementation, and serves as a baseline for the rest of our work.

In the original paper, the number of function evaluations was reported with the mean fitnesses. The given explanation is that the main focus of the exercise being the fitness - and not so much failures or successes - the required number of function evaluations to get a result is not that important: each algorithm should have the time - the budget - to reach the optimum or at least converge.

While this is perfectly valid, we don't feel comfortable to do so as it weakens the comparison between algorithms. Instead of reporting the mean number of function evaluations used, we will prefer the SP1 measure as defined in [Auger and Hansen, 2005] : $SP1 = \frac{\mathbb{E}(T_s)}{p_s}$, where $\mathbb{E}(T_s)$ is the expected number of function evaluations used in a successful run and p_s is the probability to get a success for a given run.

This measure has some disadvantages *e.g.* when the success probability is 0), but it allows a more accurate comparison between algorithms, in particular when using restarts. In such a way, two possible strategies (aiming for a 100% success rate no matter the cost or allowing restarts if the solution is not quickly found) are both possible and their performances can be compared without bias one way or another.

9.4 Differential Evolution

While the first work on this clustering benchmark obviously did not try to compare each and every possible optimization algorithm, we felt that given the specificities of the problem, Differential Evolution (DE) [Storn and Price, 1997] could perform quite well. This feeling is substantiated by [Das and Suganthan, 2011] in which DE is said to perform very well on a lot of testbeds.

Built around crossovers, the DE algorithm replaces part of a given individual with two or more others. Many different variants of DE exist, each one defining the crossovers rule. The one we chose was "DE/curr-to-best/1". For a given generation, we then have:

The only difference from "DE/best/1" is thus the update formula, which is $Y(i) \leftarrow Best(i) + f_1(A(i) - B(i))$.

In the spirit of [Gallagher, 2016], we didn't try to tune the algorithm's parameters. Instead, in the absence of a standard recommendation, we set $CR = 0.5$, $f_1 = f_2 = 0.8$ for a population size of 30. The initialization points were randomly drawn with a normal distribution of mean the average of the range of the variables, with a standard deviation of a third of that average. We used here the same stopping criteria as with CMA-ES in our previous experiment.

DE/curr-to-best/1: $U(0,1)$ is a random uniformly distributed number between 0 and 1, CR is the crossover rate parameter, f_1 and f_2 are two real numbers, *Best* is the best individual in the generation, and f is the evaluation function, n is the dimension of a point in the given dataset.

```
for each individual I do
    Y ← I
    Randomly choose A and B, two individuals distinct from I and Best
    Randomly select an index R ∈ {1,...,n}
    for all i ∈ {1,...,n} do
        if i = R or U(0,1) < CR then
            Y(i) ← I(i) + f1(A(i) − B(i)) + f2(Best(i) − I(i))
        end if
    end for
    if f(Y) < f(I) then
        Replace I by Y
    end if
end for
```

9.5 Progressive Differential Evolution

In some of our first trials, when studying the reasons for failures to reach the optimum, we reached the conclusion that in a third of the failed runs, this failure was due to falling in a local optimum. As can be seen on Figure 9.1 with a $3e4$ budget, in most cases the failures to reach the optimum are simply due to a lack of budget: the clusters found are not exactly at the optimum but centered around them. In fact, by increasing the budget, we saw that indeed, those points went to the optimum.

In the second case however, we can see that the points found are symmetrically opposed to the optimum solution, one cluster at the top, two at the bottom. This configuration on the Ruspini problem with $k = 3$ gives a fitness of ≈ 51155, which is only slightly worse than the optimum of ≈ 51063. As such, there is only a very small probability that any mutation would get to the real optimum close enough to improve the solution.

In order to avoid this, we introduced "Progressive Widening" we saw in Chapter 6 in Section 6.1.1, applied to Differential Evolution.

Here, we chose to use $R = 100$, which means that every one hundred generation, we increase the number of parameters to optimize until we reach $n \cdot k_{max}$.

Of course, the fact that we optimize k clusters doesn't mean that the others "disappear": they are still taken into account in the evaluation, but don't move from their initial position,

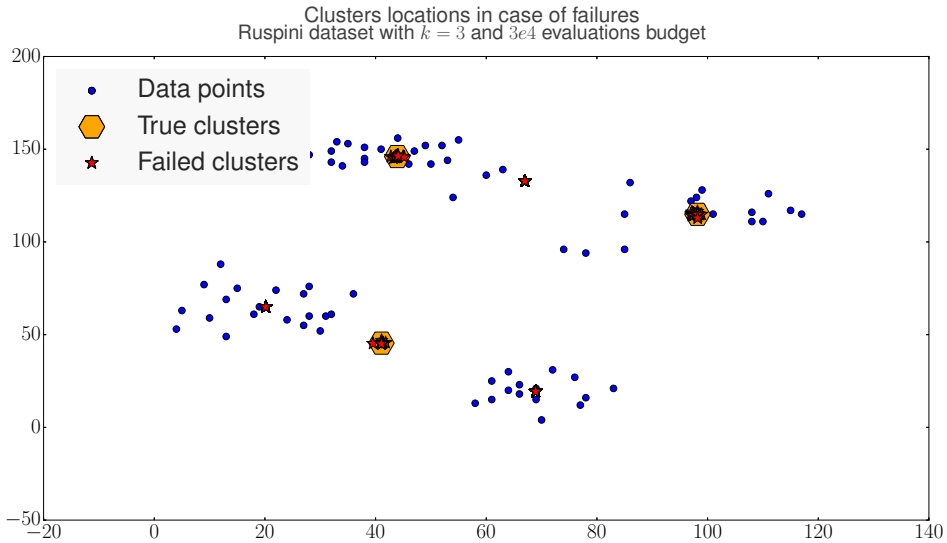


Figure 9.1 – Clusters position on failure cases, Ruspini dataset with $k = 3$

which is the center of the search space. This means that even when training k clusters, there is always one more that can be selected as the nearest from a given point. While we could have completely removed them from the evaluation, we felt that this would have reduced the black-box context of the problem.

In fact, one could argue that we are only able to use Progressive Widening by weakening the black-box setting of the problem. Indeed, since we know the dimension of the problem, we know that to add a cluster we have to add N parameters. We don't think this is an issue however, this knowledge being as much part of the specification of the problem

PDE: k_{max} is the desired number of clusters, N is the dimensionality of each point, R determines the number of generations to do with k clusters

```
Initialize population
k ← 1
while Not stop do
    for i = 0 to R do
        Run one generation of DE on the k · N first parameters
    end for
    k ← min(k + 1, k_max)
end while
```

as the definition of the search space.

9.6 Results

9.6.1 DE vs CMA-ES

D	k	$f*$	DE	DE SP1	PDE	PDE SP1
G	02	6.02546e11	6.025472e11 (5.0e-04)	5.6160e03 (6.6e02)	6.025472e11 (5.0e-04)	9.1242e03 (2.9e02)
	03	2.94506e11	3.006674e11 (4.3e10)	9.0893e03 (1.8e03)	2.945066e11 (0.0e00)	1.3496e04 (2.6e02)
	04	1.04474e11	1.500823e11 (8.1e10)	2.3898e04 (1.6e04)	1.044747e11 (0.0e00)	1.9849e04 (4.5e02)
	05	5.97615e10	7.423346e10 (3.6e10)	4.5462e04 (4.3e04)	6.065579e10 (6.3e09)	2.7124e04 (1.6e03)
	06	3.59085e10	4.776401e10 (3.8e10)	4.8107e04 (2.3e04)	3.611288e10 (1.4e09)	3.3399e04 (1.5e03)
	07	2.19832e10	3.176165e10 (1.6e10)	2.0357e05 (1.1e05)	2.423709e10 (5.1e09)	8.8896e04 (3.5e04)
	08	1.33854e10	2.182272e10 (8.3e09)	∞	1.639762e10 (4.1e09)	∞
	09	7.80442e09	1.562879e10 (6.3e09)	∞	1.127751e10 (2.9e09)	∞
	10	6.44647e09	1.281459e10 (6.2e09)	∞	8.793075e09 (4.3e09)	2.2032e06 (3.4e05)
I	02	1.52348e02	1.523480e02 (0.0e00)	8.9892e03 (2.7e03)	1.523480e02 (0.0e00)	1.3222e04 (3.5e02)
	03	7.88514e01	8.032188e01 (1.0e01)	2.0023e04 (1.3e04)	7.885212e01 (1.5e-03)	2.1965e04 (7.9e02)
	04	5.72285e01	5.867260e01 (5.2e00)	5.0221e04 (2.7e04)	5.722847e01 (4.8e-14)	2.6411e04 (5.6e03)
	05	4.64462e01	4.978281e01 (4.3e00)	1.3932e05 (8.1e04)	4.847058e01 (1.8e00)	1.7655e05 (5.8e04)
	06	3.90400e01	4.210588e01 (3.4e00)	1.9240e05 (7.7e04)	3.961203e01 (1.5e00)	1.2713e05 (4.7e04)
	07	3.42982e01	3.735682e01 (3.4e00)	1.3620e06 (3.7e05)	3.506627e01 (1.6e00)	4.1865e06 (0.0e00)
	08	2.99889e01	3.288639e01 (3.3e00)	1.0018e06 (1.2e05)	3.084912e01 (1.4e00)	9.5156e05 (1.3e05)
	09	2.77861e01	2.928749e01 (2.2e00)	1.6612e06 (1.1e05)	2.855921e01 (1.2e00)	1.3116e06 (2.8e05)
	10	2.58341e01	2.795759e01 (2.7e00)	3.5490e06 (0.0e00)	2.695760e01 (9.1e-01)	7.8765e06 (0.0e00)
R	02	8.93378e04	8.933783e04 (0.0e00)	5.9892e03 (2.3e03)	8.933783e04 (0.0e00)	1.0646e04 (4.0e02)
	03	5.10635e04	5.109841e04 (4.5e01)	2.0758e04 (9.1e03)	5.106348e04 (4.1e-11)	1.1740e04 (4.0e02)
	04	1.28811e04	1.288105e04 (0.0e00)	∞	1.288105e04 (0.0e00)	1.1175e06 (0.0e00)
	05	1.01267e04	1.015393e04 (1.9e02)	∞	1.013935e04 (1.1e01)	∞
	06	8.57541e03	8.664380e03 (2.5e02)	1.0505e05 (8.3e04)	8.660781e03 (1.1e02)	9.7329e04 (2.2e04)
	07	7.12620e03	7.179452e03 (1.4e02)	1.0829e05 (7.4e04)	7.193774e03 (1.0e02)	1.6640e05 (1.0e04)
	08	6.14964e03	6.246995e03 (3.6e02)	1.7645e05 (1.2e05)	6.168576e03 (3.4e01)	7.7184e04 (2.0e04)
	09	5.18165e03	5.441820e03 (4.2e02)	2.8236e05 (1.3e05)	5.314655e03 (1.9e02)	1.3664e05 (5.8e04)
	10	4.44628e03	4.694111e03 (4.3e02)	2.6013e05 (9.9e04)	4.622832e03 (8.4e01)	8.9633e05 (4.6e04)

Table 9.2 – Average fitness results and SP1 measure (mean and standard deviation) for DE and PDE. An SP1 measure of ∞ means that the optimum could not be reach for any of the 50 runs. Results are give for the German Town (G), Iris (I) and Ruspini (R) data-sets for all values of k.

The results we obtained with DE shown in Table 9.2 and Figure 9.3 were very good, often better - sometimes by far - than CMA-ES(50,100). The first striking result is that DE more consistently reaches the optimum solution: in only five cases (three on the German Town dataset, two on the Ruspini dataset) DE was not able to reach the optimum at least once in the 50 runs reported here.

As such, it comes as no surprise that the average fitness obtained by DE after 50 runs was improved in almost all cases (except on the Iris dataset when $k \leq 6$ and on the Ruspini dataset with $k = 3$). While this improvement is not necessarily ground breaking on the Ruspini dataset for example, it is much more important on the German Town problem (see Figure 9.3a).

9.6.2 DE vs PDE

The effects of the Progressive Widening on DE were twofold: first, it globally improved the average fitness across the board: in all but one trial (Ruspini with $k = 7$), the mean fitness and associated standard deviation were better with Progressive Widening than without. Once more, this is most notable on the German Town problem. Furthermore, in only one case now (Iris dataset with $k = 10$) is CMA-ES the best: on all other cases, PDE gets better results.

The second effect (shown in Table 9.3) was the one we expected: the success rate improved, we find the optimum more often. Most notably, with $k = 3$ on the Ruspini dataset, we went up from a 62% success rate to a full 100%: we no longer fall in the local optimum reported in Figure 9.1, which was our goal when adding Progressive Widening to DE.

In five cases though the rates went down but only in two cases was this decrease important: from 58% to 20% on the Ruspini dataset with $k = 7$ (which is also the only case where the mean fitness obtained by DE is better than PDE) and from 32% to 6% still on the Ruspini dataset but with $k = 10$. Interestingly here, while the success rate decreased by almost 30%, the mean fitness obtained by PDE is still better than the one from DE.

In fact thanks to this, we can see that while the Progressive Widening works very well in most instances in order to avoid a local minimum, in some rare cases it is exactly the opposite, as we can see on Ruspini with $k = 10$. While the solution found is often very good - there is not a huge difference between DE and PDE mean fitness there - by plotting the proposed solution we see that when PDE fails to reach the optimum and stagnates, it is because it fell in a local minimum.

9.6.3 The cost of PDE

Given the fact that the budget and stopping criteria are the same for DE and PDE, the SP1 measures reported in Table 9.2 mostly reflect the differences in success rate we saw previously. In the few cases were both algorithm have (almost) the same success rate, we can see that the SP1 measure is higher (or worse) for PDE than for DE : the introduction of Progressive Widening is not without cost.

(a) German Town dataset

k	CMA	DE	PDE
02	100	100	100
03	48	98	100
04	10	76	100
05	18	74	98
06	0	74	88
07	0	38	74
08	0	0	0
09	0	0	0
10	0	0	8

(b) Iris dataset

k	CMA	DE	PDE
02	100	100	100
03	100	86	84
04	0	56	100
05	0	28	28
06	0	32	50
07	0	4	2
08	0	6	8
09	0	4	8
10	0	2	2

(c) Ruspini dataset

k	CMA	DE	PDE
02	100	100	100
03	56	62	100
04	0	0	2
05	0	0	0
06	24	46	36
07	16	58	20
08	0	42	64
09	0	32	52
10	0	32	6

Table 9.3 – Success rate for CMA(50,100), DE and PDE

This is even more clearly illustrated in Figure 9.2, where some statistics on the fitnesses of 50 runs of DE and PDE are plotted. On the first few evaluations, PDE performs two orders of magnitude worse than DE, still one order of magnitude worse after $5e3$ evaluations, and it is not until at least $1.5e4$ evaluations that PDE performs at least as well as DE. While this is to be expected since until then not all clusters are optimized, it is still something to take into account.

9.7 Conclusion

DE performs very well on clustering problems, even when compared to clustering algorithms or CMA-ES, the current state of the art on this benchmark. This, by itself, is a very

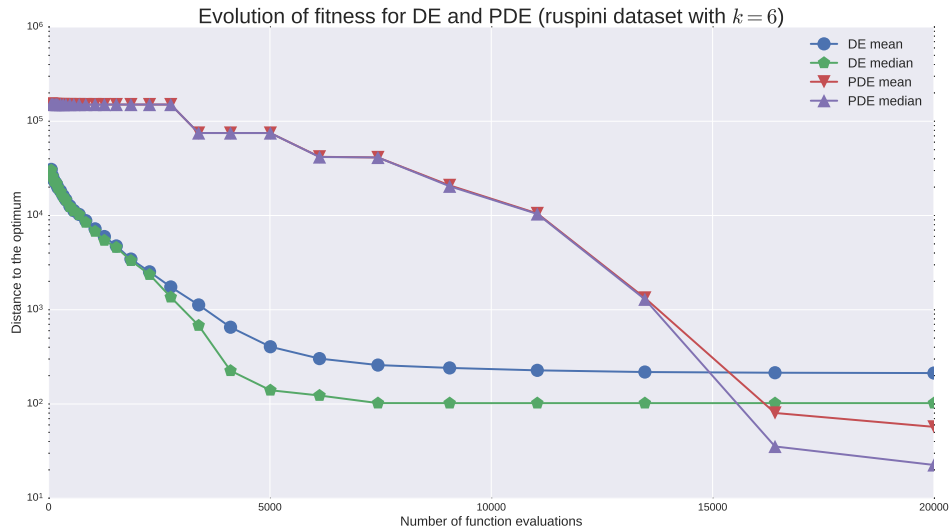


Figure 9.2 – Fitness statistics evolution on the Ruspini dataset with $k = 6$ with DE and PDE. The Progressive Widening has a clear cost at the beginning of the optimization process.

impressive result.

Our proposed variant of DE, PDE, gets even better results in most cases illustrating the good impact the concept of Progressive Widening can have on a black box algorithm.

In addition, we propose a baseline for the SP1 measure that will allow more robust comparisons of algorithms on this benchmark in the future.

9.8 Further work

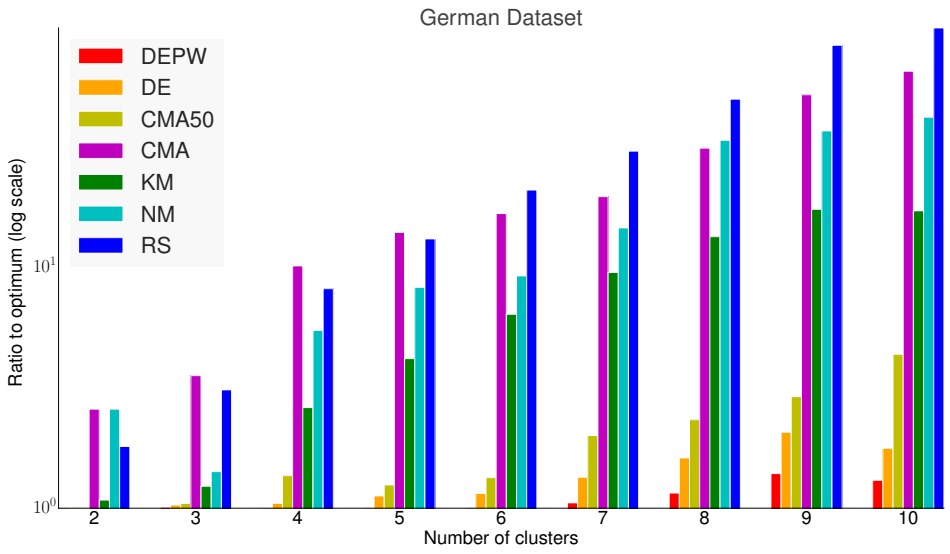
While still following the spirit of the original chapter by not tuning the algorithms parameters, there are still many possibilities to try and improve the results. Some ways to do so include other mutations rules for DE (DE/rand/1, DE/best/1, *etc.*), using Adaptive Differential Evolution, or other variants.

Of course, another way could be to use the progressive strategy on other algorithms when possible: for algorithms with covariance matrices such as CMA-ES, CMSA [Beyer and Sendhoff, 2008] or even the self-adaptive with covariance algorithm [Rechenberg, 1973b] such a change is not trivial. But for others like Particle Swarm Optimization [Eberhart and Kennedy, 1995, Shi and Eberhart, 1998a] or the other mem-

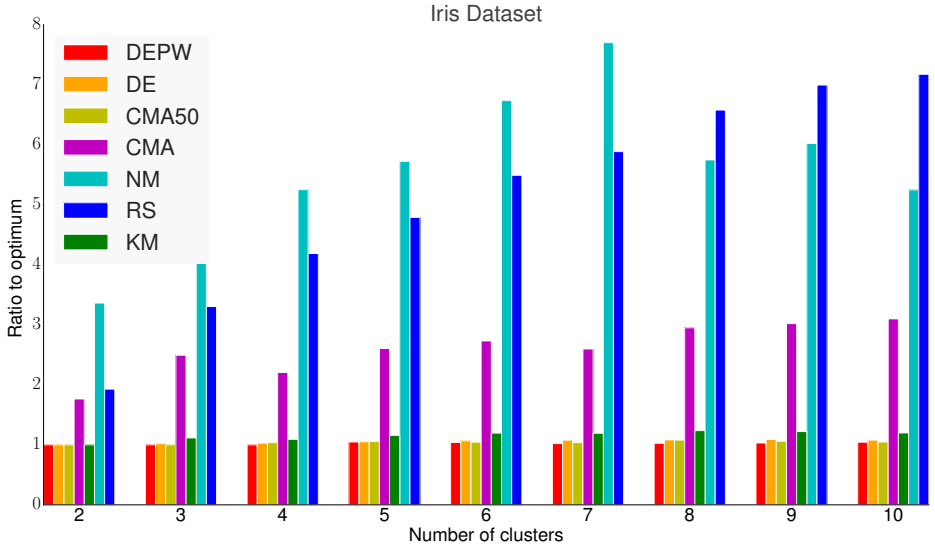
bers of the Self-Adaptive family [Beyer, 2001] (isotropic or anisotropic, 1+1, *etc.*) this is quite straightforward.

The most interesting improvements could be done on the Progressive Widening concept. For example, knowing why in some instances it is more prone to fall in a local minimum would be interesting.

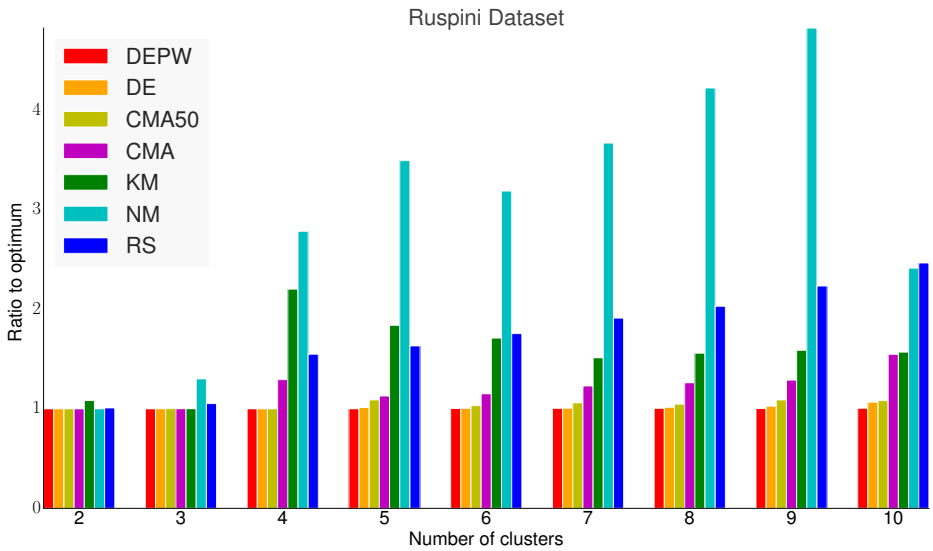
Furthermore, we have seen that the Progressive Widening is not without cost. To lessen that cost, instead of adding clusters (or parameters in the general case) at fixed time-steps we could design a rule that dynamically adds them when the fitness is reasonably stable. An intermediate step might be to add those parameters after an increasing number of time-steps (evaluations or generations) with a logarithmic rule for example, such that the more parameters are currently optimized, the more time is spent on them before adding more.



(a) German Towns dataset



(b) Iris dataset



(c) Ruspini dataset

Figure 9.3 – Performance as a ratio to the optimum ($\frac{\hat{f}}{f^*}$) of results reported in the original chapter compared to DE and PDE with a $2e5$ budget. From left to right are PDE, DE, CMA-ES(50,100), CMA-ES, KM, NM and RS.

Chapter 10

Understanding bio-physical structures with Genetic Algorithms

10.1 Introduction

There are many works done to test Optimization Algorithms, most of them on artificial benchmarks [Suganthan et al., 2005, Hansen et al., 2010b]. While there are fewer made on benchmarks rooted in real world problems, they do exist [Gould et al., 2003, Gallagher, 2016, Berthier, 2015d]. Looking further than simple tests of the algorithms, we find many applications in a wide range of domains: physics, economics, biology, imagery, *etc.*

One of the many applications of Stochastic Optimization in biology [Chatterjee et al., 1996, Balsa-Canto et al., 2012, Banga, 2008] is to identify and test models that are best able to fit the real world. This can also serve to validate - or invalidate - hypotheses about a given biological phenomenon.

Here, we will focus on one such phenomenon, regarding the properties exhibited by the shell of some insects or the wings of some butterflies to reflect light with a much better efficiency than most artificial dies can. In the 70s, with the development of electronic microscopes, it was found that the cuticles of some insects were assembled in a structure called Bragg Mirrors [Bragg and Bragg, 1913], something that we can manufacture reliably since the 60s (detailed in section 10.2).

Despite this knowledge and tools able to compute the optical response of a given structure, no one had been able to reproduce *in silico* a structure similar to those found in nature: the results of simulations were always much less efficient and a lot more complex.

Applying modern optimization algorithms (presented in section 10.3) was however able to achieve much better results (see section 10.4), almost identical to the structures

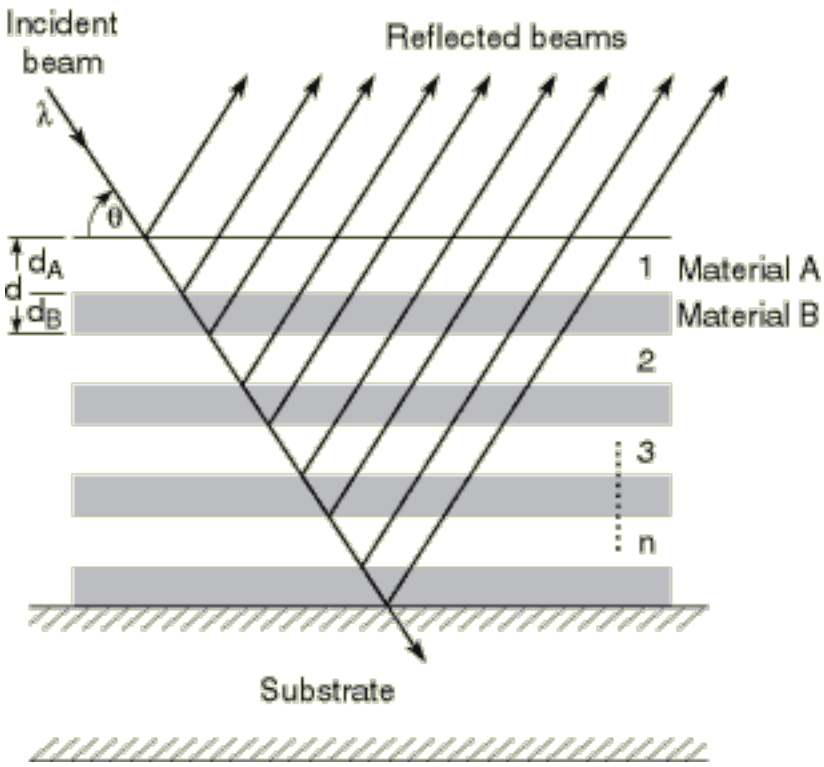


Figure 10.1 – Structure of a Dielectric Mirror: multiple layers of materials (usually two types) are stacked. The two materials have a different reflective index which produces the reflectivity of the structure. Image from [Underwood and Vaughan, 1986]

found in nature. Perhaps not so unsurprisingly, this work was able to exhibit the good results of algorithms conceptually closed to *in-vivo* evolution, when the goal is to mimic the results of millions of years of natural evolution.

10.2 Structural colours

A Bragg Mirror [Bragg and Bragg, 1913], also know as a dielectric mirror [Birge and Kärtner, 2006], is a structure composed of multiple alternate layers of different optical materials. Despite the fact that each layer is completely translucent, the overall structure is able to reflect a high percentage of the incoming light, whose wavelength is at or around a specific value. While the simplest dielectric mirrors are designed to reflect a small specific region of the spectrum, it is possible to design more complex mirrors such that they are able to reflect a wider spectrum, in which case we will

refer to them as Chirped Mirrors. Dielectric mirrors are mainly used for laser studies or applications [Keller et al., 1992, Hart et al., 2002].

The reflective properties of a dielectric mirror depend on the number of layers, their width and their reflection coefficient. Altering those parameters produces new reflective properties. The simplest mirrors, reflecting light at and around a specific wavelength consist of the same two type of layers regularly interleaved. The type and width of the layers are the parameters determining the exact wavelength they reflect. For mirrors reflecting multiple wavelength or even a wide spectrum, the nature and width of every layer can be different. The layers being completely translucent, they do not themselves reflect - nor absorb - light: it is the interfaces between two successive layers that allow a mirror to reflect light. The exact consequences can be computed thanks to the Fresnel Equations.

Dielectric mirrors, or Bragg mirrors are nothing new: while they may be used in modern applications, they take their name from William Lawrence Bragg and his father, Sir William Henry Bragg, who earned the Nobel Price of Physics in 1915 by using them. They didn't invent the mirrors though, and in fact, it is not clear who made their discovery first. Those structures can be found in nature, on the shell of some insects (for example the *Chrysochroa Fulgidissima*, a jewel beetle) or the wings of butterflies (of the *Morpho* type). Those structures are responsible for the iridescent (but mostly blue) sheen visible on those animals.

Despite being the focus of many studies for a long time now, dielectric mirrors have not yet revealed all their secrets: the exact properties of some natural examples are not yet understood. One of such properties was for example the exact impact of the first layer. There are of course some rules to design such structures, but more often than not, those same rules restricted what could be studied.

To understand, it was necessary to be able to simulate the impact of various constraints on the structure of the mirror, while staying as close as possible to the desired efficiency: a task Stochastic Optimizers are perfectly designed to handle.

10.3 Algorithms

For those problems in addition to the usual optimizers, a variant of CMA-ES was used, where instead of drawing the mutations on $\mathcal{N}(0,1)$, they were generated by a quasi-random process as proposed in [Teytaud and Gelly, 2007].

An interesting point to make, is to note the different origins of those algorithms: two of them - CMA-ES and NM - are the products of applied mathematics. The three others are inspired by nature: $(1+1)-ES$ can be seen as an asexual reproduction; DE is akin to evolution through reproduction (albeit with more than two partners); finally PSO mimics

the flight of birds, or shoals of fish.

All optimizers ran ten times for any given number of layers on each of the three problems. In every instance, the maximum budget was set at 10^4 evaluations. For the first two problems (unique wavelength and full visible spectrum), the dimension of the problem was the number of layers of the mirror. Each layer was constrained to have a thickness between 10 and 150 nanometers. The last problem, reproducing natural structures is more complex: each layer is defined by its thickness (ranging from 0 to 150), its shift from the origin (from 0 to 600), the third is its width (from 0 to 600) and the last is the space between this layer and the next (from 0 to 75). The overall dimension is then equal to four times the number of layers.

10.4 Results

The optimization process was done in three steps: the first step was to make sure that we were able to handle the problem, by designing the structure of a dielectric mirror able to reflect as well as possible a single wavelength (a Bragg Mirror). This achieved, we increased the complexity of the problem, and asked not the reflection of a single wavelength, but of all the visible spectrum (a Chirped Mirror). Finally, we introduced more complexity in order to approach the dielectric mirrors found in nature.

10.4.1 Bragg Mirrors

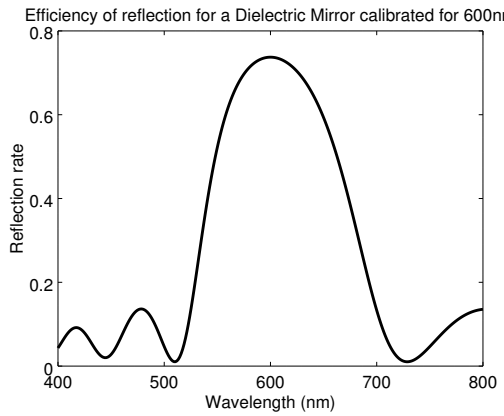
Here, the goal is to obtain a structure reflecting light with a wavelength of λ. The objective function is then $1 - r(\lambda)$, where $r(\lambda)$ is the reflection coefficient of the structure. This reflection is computed from the Fresnel equations.

As can be seen in Figure 10.2, ten layers was already quite good, since the mirror was able to reflect almost 80% of incoming light with a wavelength of 600nm. As the number of layers increases, the percentage of reflected light increases too, at more than 95% with 20 layers to almost 100% with 30: achieving a perfect 100% result is impossible, each added layer would only approach it a bit more.

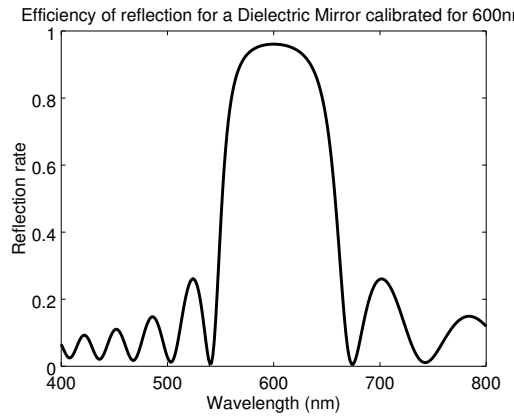
As such, since the bottom layers - added last - are less important in the overall reflection, it is interesting to note that this problem is ill conditioned.

Figure 10.3 shows the structure of the ten layers mirror obtained. The structure is almost perfectly regular, with alternative layers of width 86 and 106nm, which was the expected result.

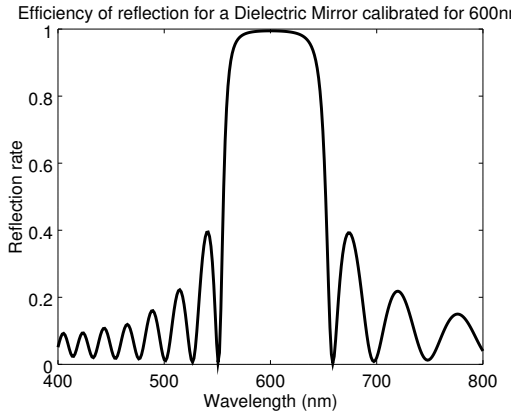
Looking at each optimizer individual results (shown in Table 10.1), we can notice the overall good performances of CMA-ES and DE. By comparing the medians to the best individual, we can also notice that up to twenty layers, the problem seems to be quite well



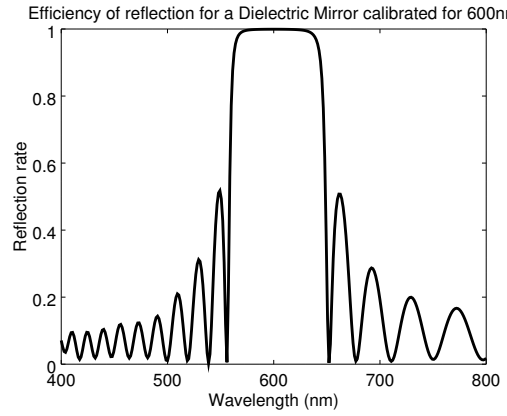
(a) Reflection profile for a Bragg Mirror with 10 layers



(b) Reflection profile for a Bragg Mirror with 20 layers



(c) Reflection profile for a Bragg Mirror with 30 layers



(d) Reflection profile for a Bragg Mirror with 40 layers

Figure 10.2 – Best results obtained by the optimization processes while designing a simple Bragg Mirror intended to reflect light of 600nm

Figure 10.3 – Interleaved layers of a Bragg Mirror reflecting almost 80% of incoming light with a wavelength of 600nm.

Layers	Stat	(1+1)-ES	CMA-ES Normal	CMA-ES QR	NM	DE	PSO
10	best	2.627e-01	**2.627e-01**	**2.627e-01**	2.627e-01	**2.627e-01**	2.628e-01
	mean	3.473e-01	2.729e-01	2.669e-01	3.082e-01	**2.627e-01**	2.645e-01
	median	3.676e-01	**2.627e-01**	**2.627e-01**	2.731e-01	**2.627e-01**	2.642e-01
	worst	6.535e-01	3.676e-01	3.676e-01	3.723e-01	**2.627e-01**	2.674e-01
20	best	3.928e-02	**3.926e-02**	**3.926e-02**	3.952e-02	3.926e-02	4.135e-02
	mean	8.166e-02	4.733e-02	4.524e-02	6.729e-02	**4.194e-02**	4.702e-02
	median	8.683e-02	**3.926e-02**	**3.926e-02**	6.093e-02	3.927e-02	4.659e-02
	worst	2.162e-01	5.841e-02	5.831e-02	1.285e-01	5.837e-02	**5.719e-02**
30	best	8.024e-03	**5.261e-03**	**5.261e-03**	8.272e-03	5.285e-03	8.308e-03
	mean	2.007e-02	2.600e-02	7.308e-03	1.438e-02	**6.168e-03**	1.160e-02
	median	1.874e-02	7.884e-03	7.881e-03	1.326e-02	**5.471e-03**	1.129e-02
	worst	6.323e-02	4.122e-01	1.183e-02	2.083e-02	**8.368e-03**	1.690e-02
40	best	1.310e-03	7.000e-04	**6.960e-04**	1.808e-03	7.260e-04	1.839e-03
	mean	5.068e-03	2.733e-02	**1.039e-03**	3.275e-03	1.274e-03	3.068e-03
	median	4.177e-03	2.364e-03	**1.044e-03**	3.032e-03	1.201e-03	2.819e-03
	worst	1.487e-02	3.791e-01	**2.345e-03**	6.475e-03	3.927e-03	5.607e-03

Table 10.1 – Optimization results for a simple Bragg Mirror with 10, 20, 30 and 40 layers. With ten layers the problem seems quite simple, but in higher dimensions there are visible performance differences between the optimizers. Overall, CMA-ES and DE perform best, with quasi-random noticeably improving the performances of CMA-ES.

handled by those two optimizers: at least half of the time they were able to reach the optimal solution. This observation is reinforced by the fact that while the other three optimizers are not competitive in higher dimension on this problem, they manage to approach the optimal solution for ten and twenty layers. After that however, their performances degrade to the point where for forty layers, there is a factor of two between their performances and those of DE/CMA-ES.

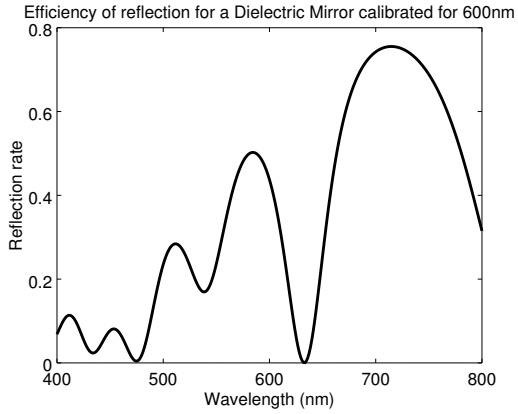
10.4.2 Chirped dielectric mirrors

Since we want here to reflect light at multiple wavelength, the objective function is complexified to become:

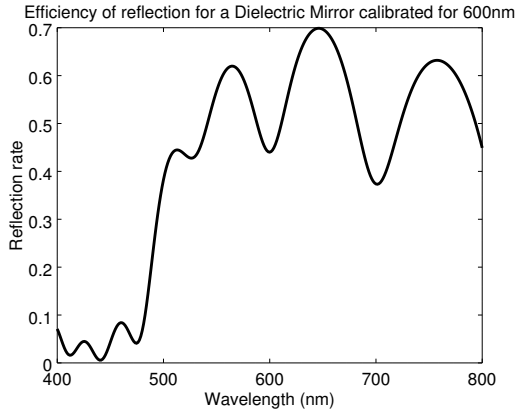
$$1 - \frac{1}{8}\sum_{n=0}^{7} r(500 + 50 \times n) \tag{10.1}$$

Again, $r(\lambda)$ is the reflection coefficient of the structure, with λ expressed in nanometers. λ goes from 500nm to 850nm, which corresponds to most of the visible spectrum, the blue color excluded (reflecting the blue color was done easily, every structure found was able to do so).

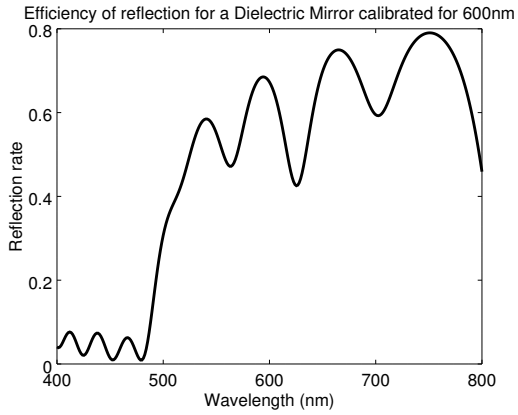
As we can see from figure 10.4, designing a Chirped Mirror to reflect the entirety of the visible spectrum seems to be a much more difficult problem. The appearance of waves can mostly be explained by the design of the objective function (See Eq. 10.1 that evaluates



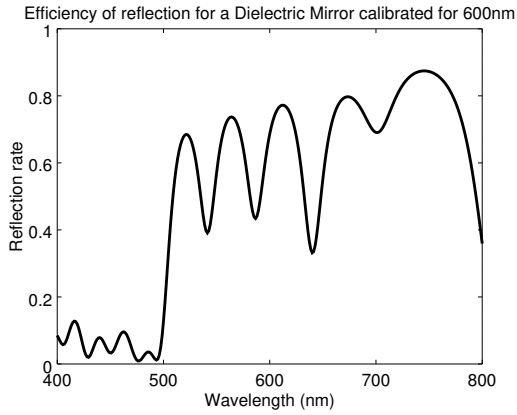
(a) Reflection profile for a Chirped Mirror with 12 layers



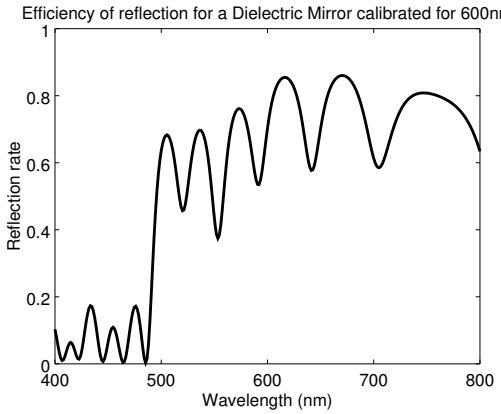
(b) Reflection profile for a Chirped Mirror with 16 layers



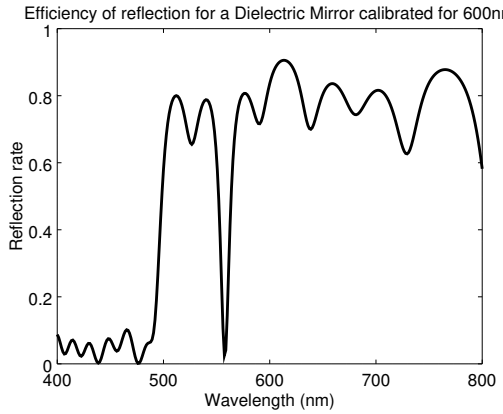
(c) Reflection profile for a Chirped Mirror with 20 layers



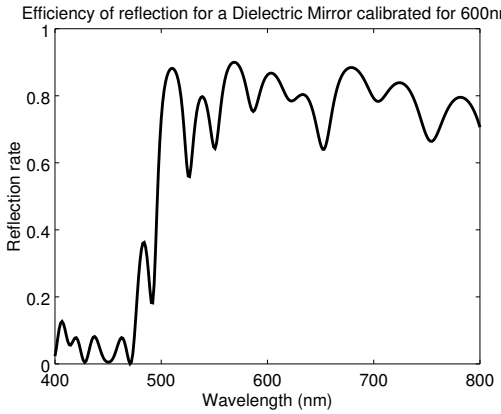
(d) Reflection profile for a Chirped Mirror with 24 layers



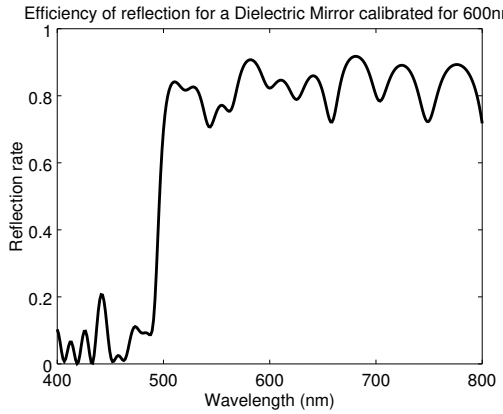
(e) Reflection profile for a Chirped Mirror with 28 layers



(f) Reflection profile for a Chirped Mirror with 32 layers



(g) Reflection profile for a Chirped Mirror with 36 layers



(h) Reflection profile for a Chirped Mirror with 40 layers

Figure 10.4 – Best results obtained by the optimization processes while designing a more complex Chirped Mirror intended to reflect light of wavelength 500nm to 800nm, which roughly correspond to the visible spectrum

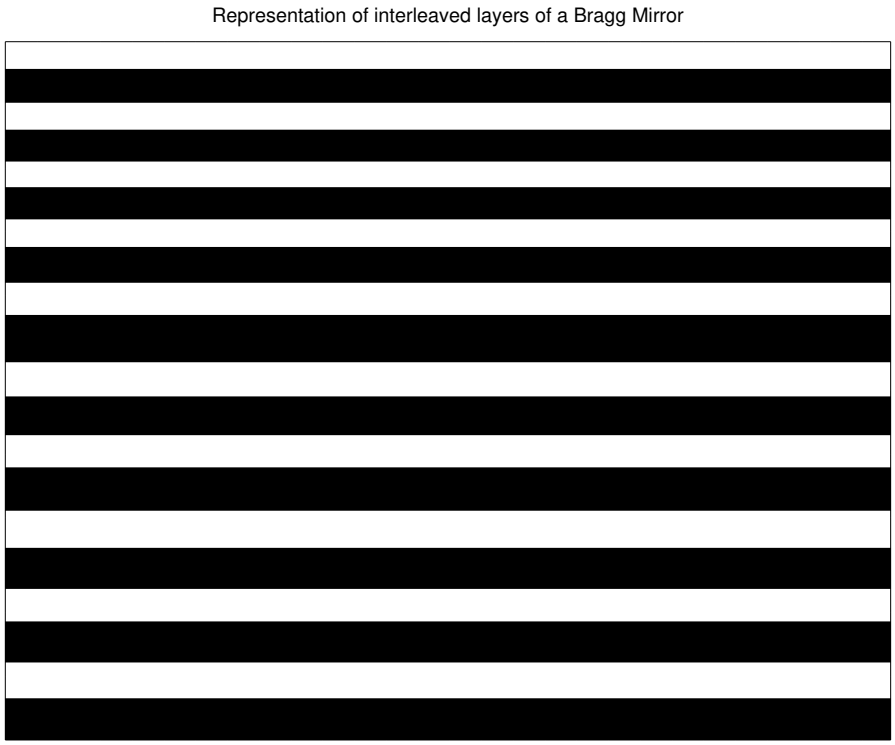


Figure 10.5 – Interleaved layers of a Chirped Mirror reflecting from 40% to 80% of incoming visible light.

the efficiency of the mirror on 40 points uniformly distributed between 500nm and 800nm (or more strictly, for $\lambda = 500 + 7.6923x$ with $x \in \{0, \cdots, 39\}$. To do otherwise would have taken too much time to allow the optimizer to converge in a reasonable time, especially when the number of layers increases.

As we can see with 20 layers, we are able to get some very nice reflection for a wavelength at around 775nm, but it is much lower at around 640nm, where not even half of the incoming light is reflected. Only with 36 and 40 layers are we able to consistently reflect around 80% of the incoming light on the full visible spectrum, which is a very good result when designing a structure like that.

As can be seen on Figure 10.5, the optimal Chirped Mirror for the full visible spectrum is not regular anymore as was the case in the simplest case. However, as is shown in table 10.2, there is some regularity, with a thin layer followed by a thicker one.

Layers	1 & 2	3 & 4	5 & 6	7 & 8	9 & 10	11 & 12	13 & 14	15 & 16	17 & 18	19 & 20
Odd, thin	85.148	82.220	78.148	84.214	98.724	104.391	98.556	113.842	99.507	109.016
Even, thick	103.159	97.388	98.108	109.286	144.311	118.996	132.302	125.882	125.991	127.987

Table 10.2 – Thickness of layers in a 20 layers Chirped Mirror reflecting around 50% of the visible spectrum. Table is read top to bottom, left to right. In this mirror, a thin layer is always followed by a thicker one

Looking at each optimizer's results (found on Table 10.3), it is interesting to note that while the problem is a difficult one in the physics sense, it doesn't seem to be for the optimizers: up to 18 layers, all of them except PSO were able to get to the optimal solution in at least one of the ten runs. In fact, DE performed so well that it was able to find the optimum solution on almost each of its runs. While PSO wasn't able to do so, it still performed very well: on each problem, it was just a little worse than DE. In fact, looking at the other statistics (mean, median and worst), we can see that PSO beats all the other optimizers except DE in almost all instances. Still, in this problem, it is not so good: the point of the exercise is to get *the* optimal solution, not to be consistent.

Increasing the number of layers, we can at first glance notice that $(1+1)-ES$, CMA-ES and NM begin to be noticeably less consistent: $(1+1)-ES$ never gets the optimal solution (or at least, the best solution found by all the optimisers since there is no theoretical way to know what is the optimum on the problem) for 30+ layers, the QR version of CMA-ES only gets it once for 32 layers, and for 20+ layers, NM only gets it for 36 and 40 layers. In the meantime, DE is once again the most consistent: reaching the optimal solution in all but 3 cases (32, 36 and 40 layers), but with better means, medians and worst results. On those indicators, PSO, NM and $(1+1)-ES$ are just behind, while both versions of CMA-ES are noticeably farther.

10.4.3 Reproducing the structures found on the *Morpho* family

When we speak about light reflection, we are thinking about a specific type of reflection: the specular reflection. Specular reflection is the reflection we get from a simple mirror: if light hit the mirror at an angle θ_i, then it is reflected with an angle θ_r with respect to the normal of the mirror, such that $\theta_i = \theta_r$. The reflection of clouds in a lake is an example of specular reflection. The specular reflection is the one we get with standard Bragg or Chirped mirrors.

On the shell of insects or the wings of butterflies, those structures are more complex: indeed, were they standard mirrors, those insects would only be visible at some very specific angles. This is of course not the case: insects "want" to be seen. In order to do so, the specular reflection must be minimized, while other angles of reflection (or orders of

Layers	Stat	(1+1)-ES	CMA-ES Normal	CMA-ES QR	NM	DE	PSO
08	best	**6.329e-01**	**6.329e-01**	**6.329e-01**	**6.329e-01**	**6.329e-01**	6.329e-01
	mean	6.614e-01	6.693e-01	6.355e-01	6.343e-01	**6.329e-01**	6.332e-01
	median	**6.329e-01**	**6.329e-01**	**6.329e-01**	**6.329e-01**	**6.329e-01**	6.331e-01
	worst	7.527e-01	7.874e-01	7.333e-01	6.910e-01	**6.329e-01**	6.358e-01
12	best	**5.570e-01**	**5.570e-01**	**5.570e-01**	**5.570e-01**	**5.570e-01**	5.617e-01
	mean	5.867e-01	7.130e-01	5.690e-01	5.643e-01	**5.595e-01**	5.660e-01
	median	5.745e-01	7.241e-01	**5.570e-01**	5.614e-01	**5.570e-01**	5.643e-01
	worst	6.559e-01	7.920e-01	7.714e-01	5.940e-01	**5.745e-01**	5.766e-01
16	best	**4.616e-01**	**4.616e-01**	**4.616e-01**	**4.616e-01**	**4.616e-01**	4.645e-01
	mean	4.999e-01	7.220e-01	4.836e-01	4.681e-01	**4.662e-01**	4.743e-01
	median	5.036e-01	7.332e-01	4.680e-01	**4.680e-01**	4.680e-01	4.720e-01
	worst	6.270e-01	7.671e-01	7.539e-01	5.178e-01	**5.103e-01**	5.200e-01
20	best	**3.882e-01**	5.768e-01	**3.882e-01**	3.883e-01	3.882e-01	3.958e-01
	mean	4.301e-01	7.073e-01	4.242e-01	4.055e-01	**3.933e-01**	4.207e-01
	median	4.237e-01	7.119e-01	3.950e-01	3.965e-01	**3.950e-01**	4.163e-01
	worst	5.302e-01	7.551e-01	7.122e-01	4.473e-01	**3.950e-01**	4.535e-01
24	best	**3.432e-01**	6.529e-01	**3.431e-01**	3.442e-01	3.432e-01	3.536e-01
	mean	3.930e-01	6.946e-01	4.798e-01	3.652e-01	**3.462e-01**	3.869e-01
	median	3.805e-01	6.965e-01	3.528e-01	3.606e-01	**3.464e-01**	3.839e-01
	worst	5.597e-01	7.250e-01	7.325e-01	4.281e-01	**3.507e-01**	4.311e-01
28	best	**2.955e-01**	6.153e-01	**2.954e-01**	3.008e-01	2.965e-01	3.188e-01
	mean	3.522e-01	6.794e-01	5.551e-01	3.183e-01	**3.080e-01**	3.572e-01
	median	3.425e-01	6.844e-01	6.213e-01	3.133e-01	**3.078e-01**	3.568e-01
	worst	4.838e-01	7.165e-01	6.946e-01	3.489e-01	**3.261e-01**	3.947e-01
32	best	2.542e-01	6.129e-01	**2.491e-01**	2.520e-01	2.501e-01	2.963e-01
	mean	3.203e-01	6.659e-01	5.208e-01	2.786e-01	**2.651e-01**	3.314e-01
	median	3.165e-01	6.678e-01	5.870e-01	2.772e-01	**2.613e-01**	3.321e-01
	worst	4.033e-01	7.030e-01	6.811e-01	3.219e-01	**2.911e-01**	3.592e-01
36	best	2.366e-01	5.897e-01	2.183e-01	**2.148e-01**	2.163e-01	2.613e-01
	mean	2.763e-01	6.523e-01	5.533e-01	2.453e-01	**2.319e-01**	3.079e-01
	median	2.699e-01	6.550e-01	5.952e-01	2.449e-01	**2.309e-01**	3.069e-01
	worst	3.331e-01	6.822e-01	6.619e-01	2.863e-01	**2.526e-01**	3.508e-01
40	best	1.923e-01	5.789e-01	4.287e-01	**1.757e-01**	1.761e-01	2.466e-01
	mean	2.478e-01	6.404e-01	6.112e-01	2.172e-01	**2.017e-01**	2.931e-01
	median	2.381e-01	6.438e-01	6.236e-01	2.167e-01	**2.034e-01**	2.945e-01
	worst	3.402e-01	6.705e-01	6.544e-01	2.631e-01	**2.350e-01**	3.285e-01

Table 10.3 – Optimization results for a simple Bragg Mirror with 8 to 40 layers, reflecting the entire visible spectrum. In low dimension, no optimizer seems really better than any other. As the dimension increases however, DE really starts to outperform all the other optimizers.

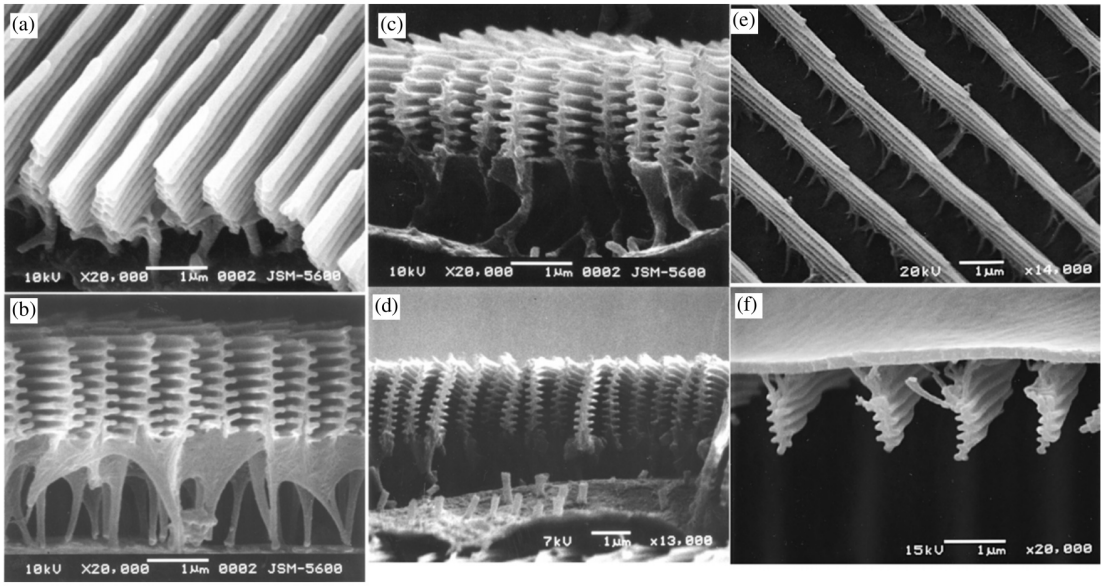


Figure 10.6 – Examples of scales found on the wings of butterflies from the *Morpho* family, from [Kinoshita et al., 2008].

reflection) are maximized.

Figure 10.6 shows the natural structures found on the wings of butterflies from the *Morpho* families, some of the most complex found in nature. In Figure 10.6b, we can see that the mirrors are "interdigitated", there is a shift between two successive mirrors. It is this phenomenon that is supposed to be responsible from the minimization of the specular reflection, and the maximization of other angles of reflection. Preliminary studies (see Figure 10.7) showed that indeed, this interdigitation should be responsible for the specific reflection properties of the *Morpho* family.

Despite the much more complex structure of the Mirror (not only is the vertical aspect of the layers important, the way they fit to one another horizontally is now important too), we didn't add any hard constraints, we just allowed the Mirror to be more than a succession of layers, but a structure in blocks, each one defined by its dimensions, shift to the origin and distance to the next block. The objective function had of course to be made more complex, to take into account this change in the definition of the Mirror, but also to force the specular reflection (r_0) to be as close to zero, and other angles of reflection (r_1 and r_{-1}) to be maximized. The hope was to "naturally" converge to a structure resembling the one found in nature. Note that since we only focused on two reflection angled, natural structures, we would get a beam effect of course absent in nature: as a consequence,

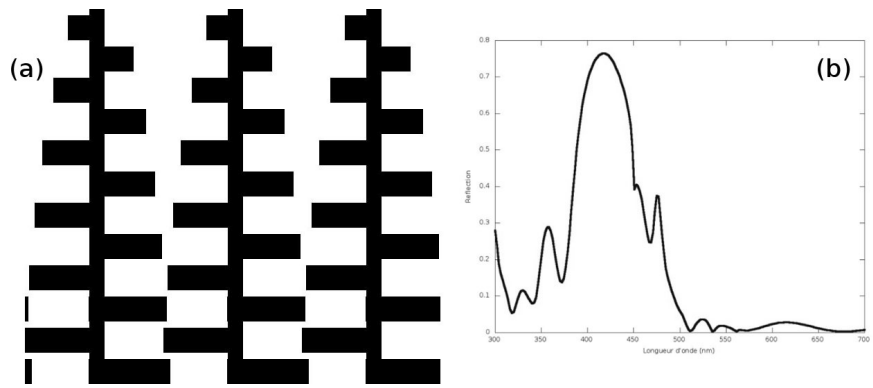


Figure 10.7 – a) Pattern of the scales found on the wings of butterflies from the *Morpho* family. b) Computed reflection of the scales. We can see a peak at around 425nm, which corresponds to the blue colour, where around 75% of the light is refracted.

natural structures are less ordered than the ones obtained here.

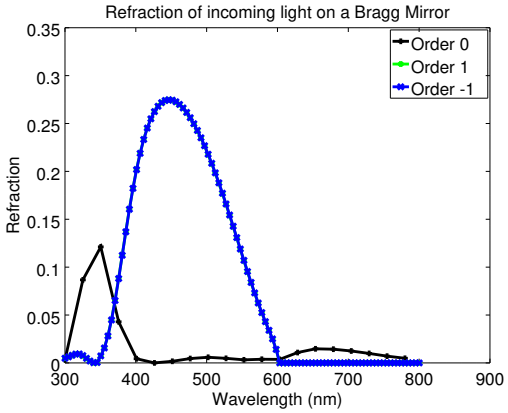
The objective function became:

$$1 - \frac{1}{2}(r_{+1}(450) + r_{-1}(450) - r_0(450)) + \frac{1}{N}\sum_{i=1}^{N} r_0(\lambda_i) \tag{10.2}$$

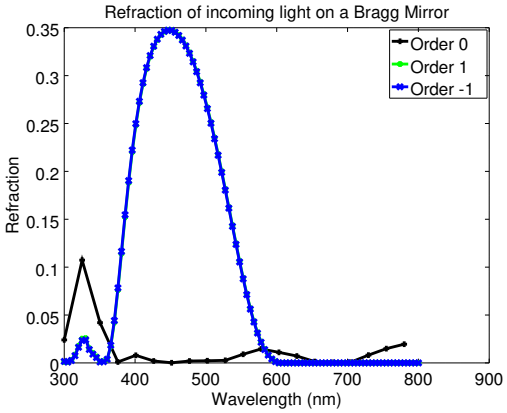
with $\lambda_i = \{300, 400, 500, 600, 700, 800\}$, ensuring the maximization of the diffraction orders at 450 nm and the minimization of the specular reflection for a wide range of wavelength.

As we can see from Figure 10.8, the goal was reached: the Order 0 (the specular reflection) is almost null for all the visible spectrum, and the Orders 1 and -1 (two other angles of reflection) are maximized for a wavelength of around 450nm. With only five layers, only about 30% of the blue light is reflected along each angle, but this percentage increases as we add more layers, reaching almost 50% with 12 layers. Another important goal was to have the two orders 1 and -1 to each have the same profile. While for 7, 8 and 12 layers there was some difference, it was minimal and acceptable. For the other number of layers however, the two orders reflected blue light in exactly the same way, which is a very good result.

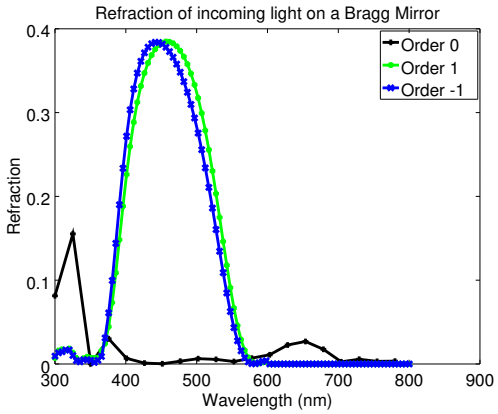
It is important to note that while on those examples we were able to get close to 100% percent of light reflected (50% along each reflection angle), and thus obtained close to perfect results, better than the structures found on the wings of the butterflies, it is because these tests didn't take some constraints important to the butterflies into account: for example, if we add a constraint on the weight of the structure, we get less reflection, but a structure even more close to the natural ones than we do here.



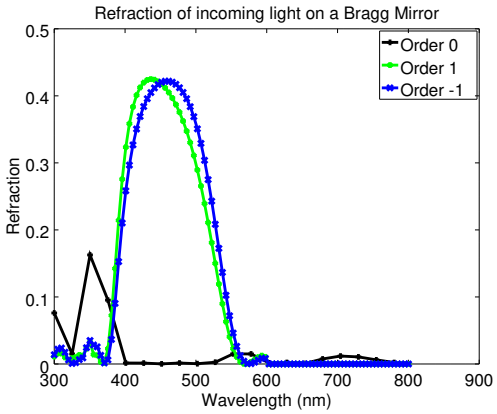
(a) Reflection profile for a *Morpho*-like structure with 5 layers



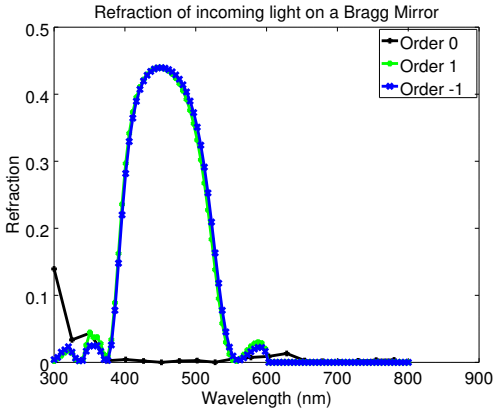
(b) Reflection profile for a *Morpho*-like structure with 6 layers



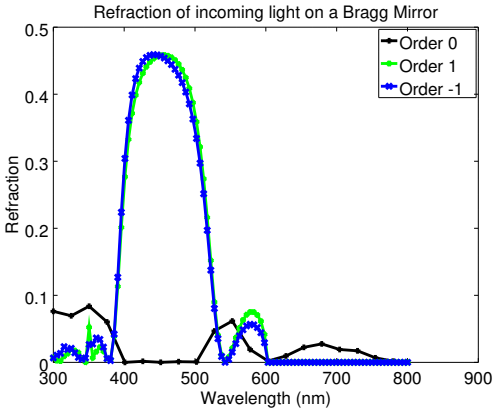
(c) Reflection profile for a *Morpho*-like structure with 7 layers



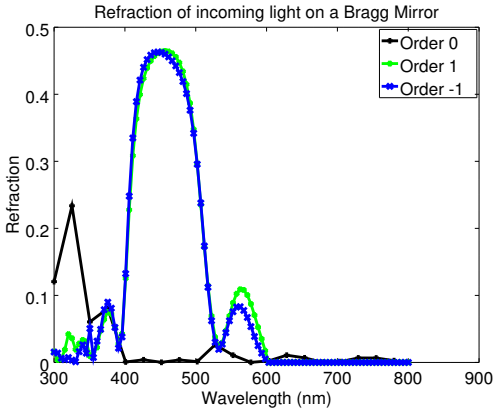
(d) Reflection profile for a *Morpho*-like structure with 8 layers



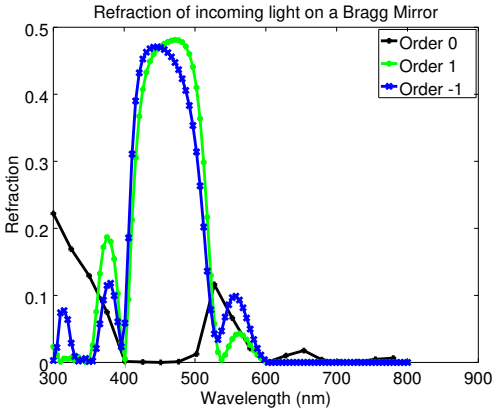
(e) Reflection profile for a *Morpho*-like structure with 9 layers



(f) Reflection profile for a *Morpho*-like structure with 10 layers



(g) Reflection profile for a *Morpho*-like structure with 11 layers



(h) Reflection profile for a *Morpho*-like structure with 12 layers

Figure 10.8 – Refraction spectrum obtained for a *Morpho*-like structure.

Layers	Stat	(1+1)-ES	CMA-ES Normal	CMA-ES QR	NM	DE	PSO
05	best	7.583e-01	7.450e-01	**7.334e-01**	7.376e-01	7.335e-01	7.678e-01
	mean	8.951e-01	8.366e-01	8.931e-01	7.637e-01	**7.360e-01**	8.175e-01
	median	8.944e-01	8.268e-01	9.416e-01	7.551e-01	**7.338e-01**	8.115e-01
	worst	9.907e-01	9.325e-01	9.869e-01	8.243e-01	**7.841e-01**	8.985e-01
06	best	7.400e-01	6.886e-01	**6.614e-01**	6.694e-01	6.616e-01	7.176e-01
	mean	8.874e-01	8.532e-01	9.138e-01	6.969e-01	**6.654e-01**	7.888e-01
	median	8.929e-01	8.528e-01	9.484e-01	6.827e-01	**6.624e-01**	7.859e-01
	worst	9.793e-01	9.703e-01	9.846e-01	7.981e-01	**7.285e-01**	8.688e-01
07	best	7.593e-01	7.840e-01	**5.965e-01**	6.314e-01	6.252e-01	7.106e-01
	mean	8.858e-01	9.404e-01	8.708e-01	6.675e-01	**6.334e-01**	7.636e-01
	median	8.985e-01	9.612e-01	9.388e-01	6.658e-01	**6.279e-01**	7.611e-01
	worst	1.035e+00	9.870e-01	9.728e-01	7.496e-01	**6.730e-01**	8.387e-01
08	best	7.830e-01	9.335e-01	**5.848e-01**	5.952e-01	5.850e-01	6.869e-01
	mean	9.023e-01	9.605e-01	8.017e-01	6.257e-01	**6.022e-01**	7.571e-01
	median	9.041e-01	9.609e-01	9.053e-01	6.204e-01	**5.920e-01**	7.517e-01
	worst	1.021e+00	9.897e-01	9.662e-01	6.841e-01	**6.490e-01**	8.651e-01
09	best	6.661e-01	8.856e-01	**5.631e-01**	5.692e-01	5.644e-01	6.424e-01
	mean	8.656e-01	9.496e-01	6.859e-01	6.021e-01	**5.941e-01**	7.405e-01
	median	8.707e-01	9.536e-01	6.177e-01	5.979e-01	**5.868e-01**	7.517e-01
	worst	1.020e+00	9.992e-01	9.808e-01	**6.675e-01**	7.089e-01	8.073e-01
10	best	7.224e-01	8.964e-01	**5.431e-01**	5.625e-01	5.520e-01	6.937e-01
	mean	8.607e-01	9.496e-01	7.678e-01	5.842e-01	**5.762e-01**	7.458e-01
	median	8.501e-01	9.485e-01	8.442e-01	5.813e-01	**5.728e-01**	7.392e-01
	worst	1.013e+00	9.911e-01	9.656e-01	**6.281e-01**	**6.281e-01**	8.322e-01
11	best	7.486e-01	9.119e-01	**5.384e-01**	5.464e-01	5.443e-01	6.736e-01
	mean	8.758e-01	9.503e-01	6.952e-01	**5.735e-01**	5.839e-01	7.499e-01
	median	8.758e-01	9.504e-01	5.867e-01	**5.678e-01**	5.705e-01	7.571e-01
	worst	1.043e+00	9.950e-01	9.575e-01	**6.340e-01**	7.605e-01	8.095e-01
12	best	7.167e-01	9.075e-01	5.404e-01	**5.336e-01**	5.376e-01	6.725e-01
	mean	8.704e-01	9.565e-01	6.365e-01	**5.563e-01**	5.976e-01	7.561e-01
	median	8.808e-01	9.553e-01	5.683e-01	**5.587e-01**	5.888e-01	7.467e-01
	worst	1.001e+00	9.968e-01	9.272e-01	**5.779e-01**	7.270e-01	8.067e-01

Table 10.4 – Optimization results for a a structure mimicking the scales of butterflies of the *Morpho* family.

Individual results shown in Table 10.4 shows a different picture from the previous results. DE still gets excellent results, but with 11 and 12 layers is beaten by NM, which gets very good results across the board. $(1+1)-ES$ and PSO are slightly worse, and don't perform as well on this problem than in the previous two. CMA-ES gets poor performances: it manages to approach the optimal with 4 and 5 layers but not quite. After that, it is quite badly outperformed by all of the others.

10.4.4 QR on CMA-ES

CMA-ES performed well on the simplest problem, not too badly on the more complex one, but got poor results when the goal was to mimic natural structures. The performances of the Quasi-Random variant however were in each and every case better than the vanilla

version on the first and second problems. On the last one, even if it was not the case for each computed statistic, it was always the case for the best run. In fact, where the vanilla version didn't reach the optimal a single time, the QR version got it in all but one case (the last one, with 12 layers), and even then, it wasn't far.

10.5 Conclusion & further work

This work managed to reach several important objective. Once more, it showed that Quasi-Random mutations really improved the results of CMA-ES in almost all cases for absolutely no additional cost. In addition to that, it is interesting to note that the optimizer that worked best, by a comfortable margin, is Differential Evolution, an optimizer inspired by sexual reproduction and natural evolution. While purely applied mathematics optimizers like CMA-ES or Nelder-Mead performed well at times, they were never able to really challenge DE. $(1+1)-ES$ and PSO are a bit in the middle: they were not able to perform as well as DE, but still got good results for the most part.

On this work, we used five optimizers, but it could be interesting to see what would be the results obtained by optimizers such as SA-ES (isotropic, anisotropic [Beyer, 2001, Beyer and Schwefel, 2002] or with a covariance matrix [Schwefel, 1981b]), CMSA-ES [Beyer and Sendhoff, 2008] or even Newuoa [Powell, 2008].

More importantly however, at least from a physicist or biologist point of view, this work shows that modern stochastic optimizers are perfectly able to tackle problems where the tools used in those fields fail. Considering that it even allowed to understand some aspects of natural structures, it is a very important advance. Even more importantly, this is the first time such complex structures naturally emerge without guiding the process along some "known" rules. Should it be needed, it is proof that simple evolutionary rules are able to create some very complex structures.

Part III

Summary, discussions and perspectives

Chapter 11

Discussions

This Ph.D. thesis deals with continuous optimization with comparison based optimizers. Part I studied new properties or an existing property under a different light and presented some improvements to existing algorithms. Part II was focused on the application of existing optimizers to real world problems, as well as the assessment of the improvements proposed in Part I in such setting.

Every chapter includes its own conclusions and perspectives, so here will only be found more general conclusions. Before we do so however, there is one important point to address:

11.1 Are those results unfair?

The results presented throughout this thesis can be seen as somewhat unfair at times, mainly for three reasons:

1. The diagonal variant of CMA-ES which was not considered
2. Time budgets instead of evaluations
3. "Default" parameters on the optimizers, and no tweaking or optimization.

There are multiple answers to give to each of those points:

11.1.1 Time budgets

It is well known that some optimizers, such as CMA-ES, have widely different performances depending on if we are looking at time or evaluations budgets. In most benchmarks, only evaluations budgets are considered, with time budgets completely ignored.

As such, it can seem surprising that we often used time budgets here. However, this can be easily understood on some of the problems we tackled, such as the Unit Commitment one. A power provider will not care how many times the simulations are run, his only requirement is to get the best possible solution before that solution has to go into effect.

Of course, it would be possible to allocate more resources to CMA-ES to reach a higher number of evaluations, and thus allow for an evaluations based comparison. But in such case, other optimizers could also benefit from the increase of resources, be it directly by running more evaluations for example to reduce the impact of noise through reevaluations, or indirectly, by increasing the complexity of the model, thus reaching a higher quality of the proposed solution.

In fact, that question could very well be turned the other way around: isn't it unfair to always present results based only on evaluation budgets?

11.1.2 Default parameters

It seems obvious that any optimizer here would be able to get better results had we tweaked their parameters, to say nothing of a meta-optimization of those parameters. The choice to do nothing of the sort was in line with the "real world" considerations of this thesis: while it would be pretty easy for us to do, a biologist, an industrialist, may not even know it is a possibility. Furthermore, tweaking parameters can be really hard to do given that they often have an impact on one another, to say nothing of a meta-optimization.

Even if it were a possibility in some cases (not considering budget issues), optimizers here were all considered with default parameters: that means that all of them could benefit from a meta-optimization, which could very well lead us to the exact same conclusions as those done here. Of course, the results could also be slightly different (for example, if CMA-ES benefited greatly from an optimization of its parameters, much more than DE would), but that would require a whole new thesis to work on.

11.1.3 CMA's diagonal variant

In chapters 3 and 4, we considered high dimension problems, where it came as no surprise that CMA-ES struggled to even get results, a problem compounded with the use of time budgets instead of evaluations ones. A possibility would have been to use the diagonal variant of CMA-ES. At the very least, it would have allowed the optimizer to get results on problems where it was simply impossible to even store the covariance matrix.

That is of course perfectly true, and it would be interesting to make such a study to know how competitive the diagonal variant of CMA-ES can be on those problems. It can not be considered as a default parametrization of CMA-ES however, and as such it was

ignored.

11.2 Methods: Summary

11.2.1 New invariances: beyond the dichotomy "full separability" and "invariance per rotation"

Invariance per rotation and its opposition, full separability has been extensively studied. There is however a whole world between - or even outside of - those two extremes.

While the condition of a problem - and its impact on the performances of a given algorithm - is well studied, and an important criterion to assess the quality of an optimizer, Chapter 3 went a step ahead, and looked at the consequences of having a large number of completely useless variables, with no impact whatsoever on the evaluation of an individual. In doing so, we introduced a new invariance, invariance with regard to useless variables.

In our experiments, we show that despite the fact that all optimizers should have this invariance, such was not the case in practice. However, while some optimizers were completely unable to handle a large number of useless variables, others were able to succeed (*e.g.* Differential Evolution and $(1+1)$-ES), reaching the same optima as when there are no useless variables.

Chapter 4, also discussed invariance, but this time with regard to rotations, mainly in the context of ill-conditioned problems. However, contrary to most works in the literature, it was this time in a context of very high dimension, with different levels of separability. In small scale, unsurprisingly, CMA-ES and CMSA obtained very good results, even with high level of non-separability and ill-conditioning. However, like in the previous chapter, those algorithms were unable to handle large scale problems. In such cases, Particle Swarm Optimization, Differential Evolution or $(1+1)$-ES are the go-to algorithms, even if they exhibited different properties: Particle Swarm Optimization was excellent with a high level of separability, whereas Differential Evolution was the best algorithm in cases of high non-separability. Interestingly enough, despite its simplicity, $(1+1)$-ES, being very fast, often obtained good results when nothing else worked.

11.2.2 Portfolio methods in continuous optimization

Portfolios are well known and studied in combinatorics and were recently introduced in continuous optimization [Baudis and Posik, 2014]. Here, one more step is done in that direction.

If unable to run each algorithm separately, designing a portfolio of each of those optimizers is probably the smartest way to proceed. In that case, no matter what are the levels of separability or the condition, the portfolio will find the optimizer best suited to the considered problem.

In fact, as discussed in Chapter 5 in the case of "restart portfolios", while there is of course some loss of budget while the portfolio determines which optimizer is the best of the current problem, this loss often ends up being more than compensated in the end.

11.2.3 Quasi Random: a generic improvement of randomized search heuristics

In addition, this chapter also discussed the improvement using quasi-random mutations can make on CMA-ES. With less redundancy, and more exploration, quasi-random mutations were able to noticeably improve the performances of every variants of CMA-ES that were tested: low or high population, mutation step-size, *etc.* In each case, the average of the QR variant was better than the average of the vanilla version.

11.2.4 Sieves methods in optimization: a generic improvement of high-dimensional noisy optimization

In addition to this improvement to CMA-ES, a variant of Self-Adaptive Evolution Strategy and Differential Evolution was proposed in Chapter 6, by introducing a technique known to statisticians as the "Sieves Method" to optimization. Here, called Progressive Widening, despite a clear cost at the start of the optimization process it showed excellent results both on artificial experiments as well as problems grounded in the real world, especially when the number of optimized variables increased logarithmically with the generation. In addition, this process has the advantage over other techniques to be anytime: it can be stopped in the middle of the optimization process, and still give a reasonable - if incomplete - solution.

11.3 Applications: modularity, real world, and beyond rotationally invariant artificial testbeds

In chapters 7 through 10, we have confirmed most of what theory and artificial experiments from previous chapters have claimed, in clustering, unit commitment, artificial and

biological photonic structures.

Testing and comparing optimizers on existing benchmarks gives some nice ideas about their respective proprieties and performances but testing them on real world applications is a must have. In Chapter 7, optimizers are tested and compared on a problem inspired by the real world. There, while CMA-ES is the king of the hill in most artificial benchmarks, it is clearly outperformed by Differential Evolution or even in some cases by $(1+1)$-ES and Particle Swarm Optimization. This might be explained by the fact that the real world is not fully invariant by rotations and that modularity (broken by introducing random rotations of the domain) makes sense in the real world and the dimension of the problem, with few cases having less than a hundred variables to optimize.

Perhaps the most important thing to keep from this chapter is that while considering the exact same problem, optimizers can have widely different performances depending on the control function used. As such, it would be perfectly appropriate to use a portfolio of optimisers on problems such as those.

Chapter 8 continues on the same unit commitment testbed but this time with the goal to design a control meta-policy. This is inspired by portfolios, since the goal is to take advantage of the individual strengths of two or more policies, but with the added advantage that the choice is not binary: the two policies can work in tandem to reach the best possible solution.

In addition to being able to train two policies at the same time for no additional cost, this technique was able to show very good results: in most cases, it performed at worse as well as the best of its constituent policies. But in some cases, it vastly outperformed any of them.

In Chapter 9, a new benchmark based on clustering with real-world data was used to compare the performances of Differential Evolution with CMA-ES. While the problems included in the benchmark seemed well suited to CMA-ES at first glance - with a dimension that doesn't go further than 40 - it was systematically beaten by DE in all but two instances.

The main lesson in this chapter however is found in the performances of Progressive Differential Evolution, a variant of Differential Evolution using Progressive Widening introduced in Chapter 6. While DE already exhibited very good results on those problems, it was itself beaten by PDE in all but one instance as far as fitnesses were concerned. In addition to that, PDE was able to reach the optimum more consistently than DE in most instances.

This improvement was not however without a cost: at the beginning of the optimization process, when PDE hasn't yet optimized every single variables, the fitness obtained

by PDE is much worse than DE. Should the budget be limited or the optimization stopped before then, PDE will be outperformed by DE.

Finally, Chapter 10 presents another application domain, by trying to reproduce a structure found in the natural world, consequence of millions of years of evolution. By doing so, it shows that once again Quasi-Random mutations are able to improve the performances of CMA-ES by a very comfortable margin here, to no additional cost whatsoever. In addition, this work shows here too excellent performances for Differential Evolution, which seems to be really well suited for real world applications. Here, it may be helped by its conceptual similarity to the natural evolution process resulting in the goal structure.

More importantly, while the goal was not perfectly reached since there is a gap between the performances of the natural structure and the best one obtained by the optimizers, another interesting result was achieved by managing to identify the role of some aspects of the natural structure, something that was, until now, not understood.

Chapter 12

Perspectives

Each chapter already includes its respective perspectives and further work ideas, so this Chapter will only focus on two points.

12.0.1 Alleviate CMA-ES dimensionality problem

In Chapters 3, 4 and throughout Part II, the results of CMA-ES were disappointing, particularly compared to its performances on artificial benchmarks, even considering that the algorithm was never tuned specifically for each problem.

While in some instances it is difficult to pinpoint the exact reason, such as in Chapters 9 and 10, in most other cases the poor performances are a consequence of the dimensionality: in Chapter 7 where the budget is given in time, CMA-ES performs poorly in high dimension due to the time it takes to compute the Covariance Matrix. In Chapters 3 and 4 in addition to this computation time problem there is the issue of the storage size for the covariance matrix in the RAM (more than 16 terabytes in dimension one million!).

There are ways to, if not suppress this issue, at least mitigate it, by only using the diagonal of the covariance matrix or the partial Hessian to update the covariance matrix. Problem is, by only using incomplete information, the performance of CMA-ES would be degraded on an evaluation for evaluation basis. This however is not an issue where CMA-ES would not be able to work at all in its complete version.

The main question would instead be to know if, while it would enable CMA-ES to work on high dimension problems, would it still be competitive with other algorithms that suffer far less from high-dimension, like Differential Evolution or even $(1+1)-ES$.

12.0.2 Progressive Widening

As Chapter 6 and 9 showed, Progressive Widening seems a very promising tool, able to improve, sometimes by a lot, the performances of algorithms like Differential Evolution or Self-Adaptive Evolution Strategy. There is however a lot of possible works to be done there.

While implementing Progressive Widening on other algorithms such as Particle Swarm Optimization is quite straightforward, for other algorithms is would at least be much more complicated, if not harmful. For example, while it would be quite easy to implement a Progressive Widening variant of $(1+1)$-ES, it would probably not improve the results of the vanilla variant due to the unique step-size for all variables.

For algorithms with covariance matrix, while it would most likely be possible to implement a Progressive Widening variant, it would require a lot more work in order to decide how to refresh the covariance matrix for example. Limited to its diagonal it would probably become easier, but it remains to be seen if it can improve anything. Finally, in the case of Nelder-Mead, a Progressive Widening variant doesn't make any sense.

Another important work would be to formalize Progressive Widening for optimization. As shown in Chapter 6, logarithmically increasing the number of optimized variables seemed to give the best results, with a formula looking like $N = log(generation \times rate)$. However, how should $rate$ be chosen remains to be seen. In Chapter 6 and 9, there didn't seem to be any "bad" option, as long as all variables were optimized before the end of the optimization process, but it should be possible to formalize this.

Finally, experiments showed a very clear cost for implementing Progressive Widening at the beginning of the optimization process. Instead of using a fixed formula, it would probably be better to increase the number of optimized variables when the optimization is stagnating: this would help reduce the cost, limiting the plateau effect clearly visible in Figure 9.2.

Quick experiments in that direction showed however that it may well be more complicated than that. In fact, it seems that the convergence of the population to the intermediary solution, while harmful in the short term, is necessary for long term gains. However, even if that is the case, it should still be possible to determine when said convergence isn't useful anymore.

Bibliography

[Arnold and Beyer, 2006] Arnold, D. V. and Beyer, H.-G. (2006). A general noise model and its effects on evolution strategy performance. *IEEE Transactions on Evolutionary Computation*, 10(4):380–391.

[Astete Morales et al., 2013] Astete Morales, S., Liu, J., and Teytaud, O. (2013). Noisy optimization convergence rates. In *Proceeding of the Fifteenth Annual Conference Companion on Genetic and Evolutionary Computation Conference Companion*, GECCO '13 Companion, pages 223–224, New York, NY, USA. ACM.

[Astrom, 1965] Astrom, K. (1965). Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10:174–205.

[Auger, 2005] Auger, A. (2005). Convergence results for (1,λ)-SA-ES using the theory of φ-irreducible markov chains. *Theoretical Computer Science*, 334:35–69.

[Auger and Hansen, 2005] Auger, A. and Hansen, N. (2005). 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 et al., 2009] Auger, A., Hansen, N., Zerpa, J. M. P., Ros, R., and Schoenauer, M. (2009). Experimental comparisons of derivative free optimization algorithms. In *8th International Symposium on Experimental Algorithms*, volume 5526, pages 3–15. Springer.

[Balsa-Canto et al., 2012] Balsa-Canto, E., Banga, J., Egea, J., Fernandez-Villaverde, A., and de Hijas-Liste, G. (2012). *Global optimization in systems biology: stochastic methods and their applications*, pages 409–424. Springer.

[Banga, 2008] Banga, J. R. (2008). Optimization in computational systems biology. *BMC systems biology*, 2(1):47.

[Baudis and Posik, 2014] Baudis, P. and Posik, P. (2014). Online black-box algorithm portfolios for continuous optimization. In *Proceedings of PPSN*, pages 40–49.

[Bellman, 1957] Bellman, R. (1957). *Dynamic Programming*. Princeton Univ. Press.

[Bengio, 1998] Bengio, Y. (1998). Using a financial training criterion rather than a prediction criterion. CIRANO Working Papers 98s-21, CIRANO.

[Berthier, 2015a] Berthier, V. (2015a). Comparing optimizers on a unit commitment problem. In *Artificial Evolution (EA2015)*. Springer Verlag.

[Berthier, 2015b] Berthier, V. (2015b). Experiments on the cec 2015 expensive optimization testbed. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 1059–1066. IEEE.

[Berthier, 2015c] Berthier, V. (2015c). Progressive differential evolution on clustering real world problems. In *International Conference on Artificial Evolution (Evolution Artificielle)*, pages 71–82. Springer.

[Berthier, 2015d] Berthier, V. (2015d). Progressive differential evolution on clustering real world problems. In *International Conference on Artificial Evolution (Evolution Artificielle)*, pages 71–82. Springer.

[Berthier et al., 2015] Berthier, V., Couëtoux, A., and Teytaud, O. (2015). Combining policies: the best of human expertise and neurocontrol. In *Artificial Evolution 2015*, pages To–appear.

[Berthier and Teytaud, 2015a] Berthier, V. and Teytaud, O. (2015a). 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.

[Berthier and Teytaud, 2015b] Berthier, V. and Teytaud, O. (2015b). Sieves method in fuzzy control: logarithmically increase the number of rules. In *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–9. IEEE Press.

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

[Beyer and Schwefel, 2002] Beyer, H.-G. and Schwefel, H.-P. (2002). Evolution strategies: a comprehensive introduction. *Natural Computing*, 1(1):3–52.

[Beyer and Sendhoff, 2008] Beyer, H.-G. and Sendhoff, B. (2008). Covariance matrix adaptation revisited - the CMSA evolution strategy. In Rudolph, G., Jansen, T., Lucas, S. M., Poloni, C., and Beume, N., editors, *Proceedings of PPSN*, pages 123–132.

[Birge and Kärtner, 2006] Birge, J. R. and Kärtner, F. X. (2006). Efficient analytic computation of dispersion from multilayer structures. *Applied optics*, 45(7):1478–1483.

[Booker et al., 1999] Booker, A., Jr., J. D., Frank, P., Serafini, D., Torczon, V., and Trosset, M. (1999). A rigorous framework for optimization of expensive functions by surrogates. *Structural Optimization*, 17(1):1–13.

[Bragg and Bragg, 1913] Bragg, W. H. and Bragg, W. L. (1913). The reflection of x-rays by crystals. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 88(605):428–438.

[Brest et al., 2006] Brest, J., Greiner, S., Boskovic, B., Mernik, M., and Zumer, V. (2006). Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *Evolutionary Computation, IEEE Transactions on*, 10(6):646–657.

[Broyden, 1970] Broyden, C. G. (1970). The convergence of a class of double-rank minimization algorithms 2. *The New Algorithm. J. of the Inst. for Math. and Applications*, 6:222–231.

[Bubeck et al., 2011] Bubeck, S., Munos, R., and Stoltz, G. (2011). Pure exploration in finitely-armed and continuous-armed bandits. *Theor. Comput. Sci.*, 412(19):1832–1852.

[Cauwet et al., 2014] Cauwet, M.-L., Liu, J., and Teytaud, O. (2014). Algorithm portfolios for noisy optimization: Compare solvers early. In *International Conference on Learning and Intelligent Optimization*, pages 1–15. Springer.

[Chang et al., 2005] Chang, J.-F., Chu, S.-C., Roddick, J. F., and Pan, J.-S. (2005). A parallel particle swarm optimization algorithm with communication strategies. *J. Inf. Sci. Eng.*, 21(4):809–818.

[Chaslot et al., 2007] Chaslot, G., Winands, M., Uiterwijk, J., van den Herik, H., and Bouzy, B. (2007). Progressive Strategies for Monte-Carlo Tree Search. In Wang, P. et al., editors, *Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)*, pages 655–661. World Scientific Publishing Co. Pte. Ltd.

[Chatterjee et al., 1996] Chatterjee, S., Laudato, M., and Lynch, L. A. (1996). Genetic algorithms and their statistical applications: an introduction. *Computational Statistics & Data Analysis*, 22(6):633–651.

[Chen et al., 2014] Chen, Q., Liu, B., Zhang, Q., Liang, J. J., Suganthan, P. N., and Qu, B. Y. (2014). Problem definition and evaluation criteria for cec 2015 special session and competition on bound constrained single-objective computationally expensive numerical optimization. Technical report, Computational Intelligence Laboratory, Zhengzhou University, China and Nanyang Technological University, Singapore.

[Christophe et al., 2014] Christophe, J.-J., Decock, J., and Teytaud, O. (2014). Direct model predictive control. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, Bruges, Belgique.

[Conn et al., 1991] Conn, A. R., Gould, N. I., and Toint, P. L. (1991). Convergence of quasi-newton matrices generated by the symmetric rank one update. *Mathematical Programming*, 50(1-3):177–195.

[Coulom, 2006] Coulom, R. (2006). Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. *In P. Ciancarini and H. J. van den Herik, editors, Proceedings of the 5th International Conference on Computers and Games, Turin, Italy*, pages 72–83.

[Das and Suganthan, 2011] Das, S. and Suganthan, P. N. (2011). Differential evolution: A survey of the state-of-the-art. *IEEE Trans. on Evolutionary Computation*, 15(1):4–31.

[Davidon, 1959] Davidon, W. (1959). Variable metric algorithm for minimization. *Report, Argonne National Laboratory*.

[Devroye et al., 1997] Devroye, L., Györfi, L., and Lugosi, G. (1997). *A probabilistic Theory of Pattern Recognition*. Springer.

[Doya and Samejima, 2002] Doya, K. and Samejima, K. (2002). Multiple model-based reinforcement learning. *Neural Computation*, 14:1347–1369.

[du Merle et al., 1999] du Merle, O., Hansen, P., Jaumard, B., and Mladenovic, N. (1999). An Interior Point Algorithm for Minimum Sum-of-Squares Clustering. *SIAM Journal on Scientific Computing*, 21(4):1485–1505.

[Eberhart and Kennedy, 1995] Eberhart, R. and Kennedy, J. (1995). A new optimizer using particle swarm theory. In *, Proceedings of the Sixth International Symposium on Micro Machine and Human Science, 1995. MHS '95*, pages 39–43.

[El-Beltagy et al., 1999] El-Beltagy, M. A., Nair, P. B., and Keane, A. J. (1999). Metamodeling techniques for evolutionary optimization of computationally expensive problems: Promises and limitations. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1*, pages 196–203. Morgan Kaufmann Publishers Inc.

[Fabian, 1967] Fabian, V. (1967). Stochastic approximation of minima with improved asymptotic speed. *Ann. Math. Statist.*, 38(1):191–200.

[Fisher, 1936] Fisher, R. A. (1936). The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*, 7(2):179–188.

[Fletcher, 1970] Fletcher, R. (1970). A new approach to variable-metric algorithms. *Computer Journal*, 13:317–322.

[Fournier and Teytaud, 2011] Fournier, H. and Teytaud, O. (2011). Lower bounds for comparison based evolution strategies using vc-dimension and sign patterns. *Algorithmica*, 59(3):387–408.

[Gagliolo, 2010] Gagliolo, M. (2010). *Online Dynamic Algorithm Portfolios*. PhD thesis, IDSIA/University of Lugano, Lugano, Switzerland.

[Gagliolo and Schmidhuber, 2005] Gagliolo, M. and Schmidhuber, J. (2005). A neural network model for inter-problem adaptive online time allocation. *Artificial Neural Networks: Formal Models and Their Applications–ICANN 2005*, pages 752–752.

[Gagliolo and Schmidhuber, 2006] Gagliolo, M. and Schmidhuber, J. (2006). Learning dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence*, 47(3-4):295–328.

[Gallagher, 2016] Gallagher, M. (2016). Towards improved benchmarking of black-box optimization algorithms using clustering problems. *Soft Computing*, 20(10):3835–3849.

[Gardner et al., 2012] Gardner, M., McNabb, A. W., and Seppi, K. D. (2012). A speculative approach to parallelization in particle swarm optimization. *Swarm Intelligence*, 6(2):77–116.

[Gelly et al., 2007] Gelly, S., Ruette, S., and Teytaud, O. (2007). Comparison-based algorithms are robust and randomized algorithms are anytime. *Evolutionary Computation*, 15(4):411–434.

[Girosi, 1998] Girosi, F. (1998). An equivalence between sparse approximation and support vector machines. In *Proc. NIPS 10*, pages 1455–1480. mk.

[Goldfarb, 1970] Goldfarb, D. (1970). A family of variable-metric algorithms derived by variational means. *Mathematics of Computation*, 24:23–26.

[Gould et al., 2003] Gould, N. I. M., Orban, D., and Toint, P. L. (2003). Cuter and sifdec: A constrained and unconstrained testing environment, revisited. *ACM Trans. Math. Softw.*, 29(4):373–394.

[Hamadi, 2013] Hamadi, Y. (2013). *Search: from Algorithms to Systems (HDR)*. Habilitation à diriger des recherches, Université Paris-Sud.

[Hansen, 2008] Hansen, N. (2008). Adaptive Encoding for Optimization. Research Report RR-6518, INRIA.

[Hansen et al., 2010a] Hansen, N., Auger, A., Ros, R., Finck, S., and Pošík, P. (2010a). 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., 2010b] Hansen, N., Auger, A., Ros, R., Finck, S., and Posik, P. (2010b). Comparing Results of 31 Algorithms from the Black-Box Optimization Benchmarking BBOB-2009. In *ACM-GECCO Genetic and Evolutionary Computation Conference*, Portland, United States. pp. 1689-1696.

[Hansen and Ostermeier, 2001] Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195.

[Hansen and Ostermeier, 2003] Hansen, N. and Ostermeier, A. (2003). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 11(1).

[Hansen et al., 2008] Hansen, N., Ros, R., Mauny, N., Schoenauer, M., and Auger, A. (2008). PSO Facing Non-Separable and Ill-Conditioned Problems. Research Report RR-6447, INRIA.

[Hart et al., 2002] Hart, S. D., Maskaly, G. R., Temelkuran, B., Prideaux, P. H., Joannopoulos, J. D., and Fink, Y. (2002). External reflection from omnidirectional dielectric mirror fibers. *Science*, 296(5567):510–513.

[Heidrich-Meisner and Igel, 2009] Heidrich-Meisner, V. and Igel, C. (2009). Hoeffding and bernstein races for selecting policies in evolutionary direct policy search. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 401–408, New York, NY, USA. ACM.

[Jägersküpper, 2006] Jägersküpper, J. (2006). In between progress rate and stochastic convergence. *Dagstuhl's seminar*.

[Jägersküpper and Witt, 2005] Jägersküpper, J. and Witt, C. (2005). Rigorous runtime analysis of a (μ+ 1) es for the sphere function. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 849–856. ACM.

[Jamieson et al., 2012] Jamieson, K. G., Nowak, R. D., and Recht, B. (2012). Query complexity of derivative-free optimization. In *NIPS*, pages 2681–2689.

[Jones et al., 1998] Jones, D. R., Schonlau, M., and Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *J. of Global Optimization*, 13(4):455–492.

[Kearns et al., 1999] Kearns, M., Mansour, Y., and Ng, A. (1999). A sparse sampling algorithm for near-optimal planning in large markov decision processes. In *IJCAI*, pages 1324–1231.

[Keijzer et al., 2002] Keijzer, M., Merelo, J. J., Romero, G., and Schoenauer, M. (2002). Evolving Objects: A General Purpose Evolutionary Computation Library. In Collet, P., Fonlupt, C., Hao, J.-K., Lutton, E., and Schoenauer, M., editors, *Artificial Evolution*, number 2310 in Lecture Notes in Computer Science, pages 231–242. Springer Berlin Heidelberg.

[Keller et al., 1992] Keller, U., Miller, D., Boyd, G., Chiu, T., Ferguson, J., and Asom, M. (1992). Solid-state low-loss intracavity saturable absorber for nd: Ylf lasers: an antiresonant semiconductor fabry–perot saturable absorber. *Optics letters*, 17(7):505–507.

[Kennedy and Eberhart, 1995] Kennedy, J. and Eberhart, R. C. (1995). Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948.

[Kinoshita et al., 2008] Kinoshita, S., Yoshioka, S., and Miyazaki, J. (2008). Physics of structural colors. *Reports on Progress in Physics*, 71(7):076401.

[Kleinman et al., 1999] Kleinman, N. L., Spall, J. C., and Naiman, D. Q. (1999). Simulation-based optimization with stochastic approximation using common random numbers. *Management Science*, 45(11):1570–1578.

[Kocis and Whiten, 1997] Kocis, L. and Whiten, W. J. (1997). Computational investigations of low-discrepancy sequences. *ACM Transactions on Mathematical Software (TOMS)*, 23(2):266–294.

[Kocsis and Szepesvari, 2006] Kocsis, L. and Szepesvari, C. (2006). Bandit based Monte-Carlo planning. In *15th European Conference on Machine Learning (ECML)*, pages 282–293.

[LaTorre et al., 2013] LaTorre, A., Muelas, S., and Pena, J.-M. (2013). Large scale global optimization: Experimental results with mos-based hybrid algorithms. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 2742–2749.

[Liu and Lampinen, 2005] Liu, J. and Lampinen, J. (2005). A fuzzy adaptive differential evolution algorithm. *Soft Computing*, 9(6):448–462.

[Mahdad et al., 2010] Mahdad, B., Srairi, K., Bouktir, T., and Benbouzid, M. (2010). Fuzzy Controlled Parallel PSO to Solving Large Practical Economic Dispatch. In IEEE, editor, *Proceedings of the 2010 IEEE International Conference of the IEEE Industrial Electronics Society*, pages 2695–2701, Phoenix, United States. IEEE.

[Marivate and Littman, 2013] Marivate, V. N. and Littman, M. L. (2013). An ensemble of linearly combined reinforcement-learning agents. In *AAAI (Late-Breaking Developments)*.

[Matoušek, 1998] Matoušek, J. (1998). On the l2-discrepancy for anchored boxes. *Journal of Complexity*, 14(4):527–556.

[McNabb et al., 2007] McNabb, A., Monson, C., and Seppi, K. (2007). Parallel PSO using MapReduce. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 7–14.

[Monicka et al., 2011] Monicka, J. G., Sekhar, N. G., and Kumar, K. R. (2011). Performance evaluation of membership functions on fuzzy logic controlled ac voltage controller for speed control of induction motor drive. *International Journal of Computer Applications*, 13(5):8–12.

[Nelder and Mead, 1965] Nelder, J. A. and Mead, R. (1965). A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313.

[Nowak and Römisch, 2000] Nowak, M. P. and Römisch, W. (2000). Stochastic lagrangian relaxation applied to power scheduling in a hydro-thermal system under uncertainty. *Annals of Operations Research*, 100(1-4):251–272.

[Nudelman et al., 2004] Nudelman, E., Leyton-Brown, K., Hoos, H. H., Devkar, A., and Shoham, Y. (2004). Understanding random sat: beyond the clauses-to-variables ratio. In Wallace, M., editor, *Principles and Practice of Constraint Programming CP 2004, LLNCS 3258*, volume 3258 of Lecture Notes in Computer Science, pages 438–452. Springer Berlin / Heidelberg.

[Ökten, 2002] Ökten, G. (2002). Random sampling from low-discrepancy sequences: applications to option pricing. *Mathematical and computer modelling*, 35(11-12):1221–1234.

[Pereira and Pinto, 1991] Pereira, M. V. F. and Pinto, L. M. V. G. (1991). Multi-stage stochastic optimization applied to energy planning. *Math. Program.*, 52(2):359–375.

[Poaík and Klema, 2012] Poaík, P. and Klema, V. (2012). Jade, an adaptive differential evolution algorithm, benchmarked on the bbob noiseless testbed. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, pages 197–204. ACM.

[Poloni and Pediroda, 1997] Poloni, C. and Pediroda, V. (1997). GA coupled with computationaly expensive simulations: tools to improve efficiency. In *Genetic Algorithms and Evolution Strategies in Engineering and Computer Sciences*, pages 267–288. John Wiley.

[Powell, 2008] Powell, M. J. (2008). Developments of newuoa for minimization without derivatives. *IMA journal of numerical analysis*, 28(4):649–664.

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

[Price, 1999] Price, K. V. (1999). An introduction to differential evolution. In *New ideas in optimization*, pages 79–108. McGraw-Hill Ltd., UK.

[Pulina and Tacchella, 2009] Pulina, L. and Tacchella, A. (2009). A self-adaptive multi-engine solver for quantified boolean formulas. *Constraints*, 14(1):80–116.

[Ratitch and Precup, 2004] Ratitch, B. and Precup, D. (2004). Sparse distributed memories for on-line value-based reinforcement learning. In *ECML 2004: 347-358*.

[Raul Hruschka et al., 2009] Raul Hruschka, E., Campello, R. J., Freitas, A. A., and Ponce Leon F de Carcalho, A. C. (2009). A survey of evolutionary algorithms for clustering. *IEEE transactions on systems, man and cybernetics. Part C, Applications and reviews*, 39(2):133–155.

[Rechenberg, 1973a] Rechenberg, I. (1973a). Evolution strategy: Optimization of technical systems by means of biological evolution. *Fromman-Holzboog, Stuttgart*, 104.

[Rechenberg, 1973b] Rechenberg, I. (1973b). *Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fromman-Holzboog Verlag, Stuttgart.

[Ruspini, 1970] Ruspini, E. H. (1970). Numerical methods for fuzzy clustering. *Information Sciences*, 2(3):319–350.

[Samulowitz and Memisevic, 2007] Samulowitz, H. and Memisevic, R. (2007). Learning to solve qbf. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, pages 255–260. AAAI.

[Schoenauer et al., 2006] Schoenauer, M., Savéant, P., and Vidal, V. (2006). Divide-and-evolve : une nouvelle méta-heuristique pour la planification temporelle indépendante du domaine. In et Gérard Verfaillie, F. G., editor, *Journées Francophones Planification, Décision, Apprentissage*, Toulouse. GDR I3 groupe PDMIA.

[Schutte et al., 2003] Schutte, J. F., Reinbolt, J. A., Fregly, B. J., Haftka, R. T., and George, A. D. (2003). Parallel global optimization with the particle swarm algorithm. *Journal of numerical methods in engineering*, 61:2296–2315.

[Schwefel, 1977] Schwefel, H.-P. (1977). *Numerische optimierung von computermodellen mittels der evolutionsstrategie*, volume 1. Birkhäuser, Basel Switzerland.

[Schwefel, 1981a] Schwefel, H.-P. (1981a). *Numerical optimization of computer models*. John Wiley & Sons, Inc.

[Schwefel, 1981b] Schwefel, H.-P. (1981b). *Numerical Optimization of Computer Models*. John Wiley & Sons, New-York. 1995 – 2nd edition.

[Shanno, 1970] Shanno, D. F. (1970). Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24:647–656.

[Shen and Wong, 1994] Shen, X. and Wong, W.-H. (1994). Convergence rate of sieve estimates. *The Annals of Statistics*, 22(2):580–615.

[Shi and Eberhart, 1998a] Shi, Y. and Eberhart, R. (1998a). A modified particle swarm optimizer. In *, The 1998 IEEE International Conference on Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence*, pages 69–73.

[Shi and Eberhart, 1998b] Shi, Y. and Eberhart, R. C. (1998b). A Modified Particle Swarm Optimizer. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 69–73, Washington, DC, USA. IEEE Computer Society.

[Spath, 1980] Spath, H. (1980). Cluster analysis algorithms for data reduction and classification of objects.

[St-Pierre et al., 2011] St-Pierre, D., Louveaux, Q., and Teytaud, O. (2011). Online sparse bandit for card game. In *Proceedings of Advanced in Computer Games 2011 (ACG 2011)*, pages 295–305.

[Stalph et al., 2008] Stalph, P. O., Ebner, M., Michel, M., Pfaff, B., and Benz, R. (2008). Genetic and evolutionary computation conference, gecco 2008, proceedings, atlanta, ga, usa, july 12-16, 2008. In Ryan, C. and Keijzer, M., editors, *GECCO*, pages 535–536. ACM.

[Storn and Price, 1997] Storn, R. and Price, K. (1997). Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11(4):341–359.

[Strens and Moore, 2001] Strens, M. and Moore, A. (2001). Direct policy search using paired statistical tests. In *Proceedings of the 18th International Conference on Machine Learning*, pages 545–552. Morgan Kaufmann, San Francisco, CA.

[Strens et al., 2002] Strens, M., Moore, A., Brodley, C., and Danyluk, A. (2002). Policy search using paired comparisons. In *Journal of Machine Learning Research*, pages 921–950.

[Suganthan et al., 2005] Suganthan, P. N., Hansen, N., Liang, J. J., Deb, K., Chen, Y.-P., Auger, A., and Tiwari, S. (2005). Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. *KanGAL report*, 2005005:2005.

[Sutton, 1996] Sutton, R. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1038–1044. The MIT Press.

[Teytaud and Gelly, 2007] Teytaud, O. and Gelly, S. (2007). Dcma: yet another derandomization in covariance-matrix-adaptation. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 955–963, New York, NY, USA. ACM.

[Teytaud et al., 2006] Teytaud, O., Gelly, S., and Mary, J. (2006). On the ultimate convergence rates for isotropic algorithms and the best choices among various forms of isotropy. In *PPSN*, pages 32–41. Springer.

[Tuffin, 2004] Tuffin, B. (2004). Randomization of quasi-monte carlo methods for error estimation: Survey and normal approximation. *Monte Carlo Methods and Applications mcma*, 10(3-4):617–628.

[Underwood and Vaughan, 1986] Underwood, J. and Vaughan, D. (1986). X-ray data booklet. *Center for X-ray Optics, Lawrence Berkeley Laboratory, Berkeley, CA*, pages 4–1.

[van Hasselt, 2011] van Hasselt, H. P. (2011). *Insights in Reinforcement Learning: formal analysis and empirical evaluation of temporal-difference learning algorithms*. PhD thesis, Universiteit Utrecht.

[Vapnik, 2013] Vapnik, V. (2013). *The nature of statistical learning theory*. Springer science & business media.

[Vidyasagar, 1997] Vidyasagar, M. (1997). *A Theory of Learning and Generalization*. Springer-Verlag, New York, New York.

[Villemonteix et al., 2009] Villemonteix, J., Vazquez, E., and Walter, E. (2009). An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*.

[Wang et al., 2009] Wang, Y., Audibert, J.-Y., and Munos, R. (2009). Algorithms for infinitely many-armed bandits. In *Advances in Neural Information Processing Systems*, pages 1729–1736.

[Yu and Zhang, 2011] Yu, W.-J. and Zhang, J. (2011). Multi-population differential evolution with adaptive parameter control for global optimization. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 1093–1098, New York, NY, USA. ACM.

[Zadeh, 1990] Zadeh, L. A. (1990). The birth and evolution of fuzzy logic. *International Journal Of General System*, 17(2-3):95–105.

[Zambelli et al., 2011] Zambelli, M., Soares Filho, S., Toscano, A. E., Santos, E. d., and Silva Filho, D. d. (2011). Newave versus odin: comparison of stochastic and deterministic models for the long term hydropower scheduling of the interconnected brazilian system. *Sba: Controle & Automação Sociedade Brasileira de Automatica*, 22(6):598–609.

[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.

[Zhao and Bose, 2002] Zhao, J. and Bose, B. K. (2002). Evaluation of membership functions for fuzzy logic controlled induction motor drive. In *IECON 02 [Industrial Electronics Society, IEEE 2002 28th Annual Conference of the]*, volume 1, pages 229–234. IEEE.

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Contributions à l'optimisation stochastique et applications au monde réel

Mots clefs : Optimisation, Stochastique, Boîte Noire, Monde Réel, Quasi-Aléatoire, Élargissement Progressif

Résumé : Un grand nombre d'études ont été faites dans le domaine de l'Optimisation Stochastique en général et les Algorithmes Génétiques en particulier. L'essentiel des nouveaux développements ou des améliorations faites sont alors testés sur des jeux de tests très connus tels que BBOB, CEC, *etc.* conçus de telle manière que soient présents les principaux défis que les optimiseurs doivent relever : non séparabilité, multimodalité, des vallées où le gradient est quasi-nul, et ainsi de suite.
La plupart des études ainsi faites se déroulent *via* une application directe sur le jeu de test, optimisant un nombre donné de variables pour atteindre un critère précis. La première contribution de ce travail consiste à étudier l'impact de la remise en cause de ce fonctionnement par deux moyens : le premier repose sur l'introduction d'un grand nombre de variables qui n'ont pas d'impact sur la valeur de la fonction optimisée ; le second quant à lui relève de l'étude des conséquences du mauvais conditionnement d'une fonction en grande dimension sur les performances des algorithmes d'optimisation stochastique.
Une deuxième contribution se situe dans l'étude de l'impact de la modification des mutations de l'algorithme CMA-ES, où, au lieu d'utiliser des mutations purement aléatoires, nous allons utiliser des mutations quasi-aléatoires. Ce travail introduit également la "Sieves Method", bien connue des statisticiens. Avec cette méthode, nous commençons par optimiser un faible nombre de variables, nombre qui est ensuite graduellement incrémenté au fil de l'optimisation.
Bien que les jeux de tests existants sont bien sûr très utiles, ils ne peuvent constituer que la première étape : dans la plupart des cas, les jeux de tests sont constitués d'un ensemble de fonctions purement mathématiques, des plus simples comme la sphère, aux plus complexes. Le but de la conception d'un nouvel optimiseur, ou l'amélioration d'un optimiseur existant, doit pourtant *in fine* être de répondre à des problèmes du monde réel. Ce peut-être par exemple la conception d'un moteur plus efficace, d'identifier les bons paramètres d'un modèle physique ou encore d'organiser des données en groupes.
Les optimiseurs stochastiques sont bien évidemment utilisés sur de tels problèmes, mais dans la plupart des cas, un optimiseur est choisi arbitrairement puis appliqué au problème considéré. Nous savons comment les optimiseurs se comparent les uns par rapport aux autres sur des fonctions artificielles, mais peu de travaux portent sur leur efficacité sur des problèmes réels. L'un des principaux aspects des travaux présentés ici consiste à étudier le comportement des optimiseurs les plus utilisés dans la littérature sur des problèmes inspirés du monde réel, voire des problèmes qui en viennent directement. Sur ces problèmes, les effets des mutations quasi-aléatoires de CMA-ES et de la "Sieves Method" sont en outre étudiés.

Title : Studies on stochastic optimisation and applications to the real world

Keywords : Stochastic, Optimisation, Black Box, Real World, Quasi-Random, Progressive Widening

Abstract : A lot of research is being done on Stochastic Optimisation in general and Genetic Algorithms in particular. Most of the new developments are then tested on well know testbeds like BBOB, CEC, *etc.* conceived to exhibit as many pitfalls as possible such as non-separability, multi-modality, valleys with an almost null gradient and so on.
Most studies done on such testbeds are pretty straightforward, optimising a given number of variables for the recognized criterion on the testbed. The first contribution made here is to study the impact of some changes in those assumptions, namely the effect of supernumerary variables that don't change anything to a function evaluation on the one hand, and the effect of a change of the studied criterion on the other hand.
A second contribution is in the modification of the mutation design for the algorithm CMA-ES, where we will use Quasi-Random mutations instead of purely random ones. This will almost always result in a very clear improvement of the observed results. This research also introduces the Sieves Method well known in statistics, to stochastic optimisers: by first optimising a small subset of the variables and gradually increasing the number of variables during the optimization process, we observe on some problems a very clear improvement.
While artificial testbeds are of course really useful, they can only be the first step: in almost every case, the testbeds are a collection of purely mathematical functions, from the simplest one like the sphere, to some really complex functions. The goal of the design of new optimisers or the improvement of an existing one is however, *in fine*, to answer some real world question. It can be the design of a more efficient engine, finding the correct parameters of a physical model or even to organize data in clusters.
Stochastic optimisers are used on those problems, in research or industry, but in most instances, an *optimiser is chosen almost arbitrarily. We know how optimisers compare on artificial functions, but almost nothing is known about their performances on real world problems. One of the main aspect of the research exposed here will be to compare some of the most used optimisers in the literature on problems inspired or directly coming from the real-world. On those problems, we will additionally test the efficiency of quasi-random mutations in CMA-ES and the Sieves-Method.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France