Titre: Title:	Répartition computationnelle efficace entre boîte noire et solveur
Auteur: Author:	Samuel Mendoza
Date:	2024
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Mendoza, S. (2024). Répartition computationnelle efficace entre boîte noire et solveur [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/61588/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Répartition computationnelle efficace entre boîte noire et solveur

SAMUEL MENDOZA
Département de génie électrique

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie électrique

Novembre 2024

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Répartition computationnelle efficace entre boîte noire et solveur

présenté par **Samuel MENDOZA**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Roland MALHAMÉ, président
Antoine LESAGE-LANDRY, membre et directeur de recherche
Sébastien LE DIGABEL, membre et codirecteur de recherche
Stéphane ALARIE, membre et codirecteur de recherche
Miguel DIAGO-MARTÍNEZ, membre

DÉDICACE

À ma mère

REMERCIEMENTS

De prime abord, je tiens à exprimer ma reconnaissance à mes superviseurs Antoine, Sébastien et Stéphane de m'avoir offert ce projet et de leur temps investi. Également de leur constante énergie positive et de leur encouragement.

Au jury, M. Roland Malhamé et M. Miguel Diago-Martínez pour la lecture de mon mémoire.

Je souhaite adresser une mention particulière à Charles Audet, qui dès notre première rencontre en calcul scientifique pour l'ingénieur, a toujours eu sa porte ouverte pour répondre à mes questions avec le sourire et la bonne humeur. Je lui suis extrêmement reconnaissant pour toutes les connaissances qu'il m'a partagées.

Je remercie également Christophe Tribes et Ludovic Salomon pour leur **GRANDE** disponibilité et leurs conseils avisés, tant sur l'utilisation du logiciel que pour orienter ma recherche.

À mes collègues et amis du GERAD et LORER, soit Alexis, Pierre-Yves, Xavier, Mathieu (le petit monsieur), Christina, Théo, Ulrich, Oussama, Meisseme, Mohamed, Maxence, Valentin, Sacha, Olivier, Xavier, Julien, Matthias, Loreley, Jean-Luc, Abraham et Victor pour les bons moments passés ensemble. Je garderai un bon souvenir de vous.

À Geneviève Paquin et Caroline-Emmanuelle Petit-Jetée, mes enseignantes en mathématique au cégep qui m'ont transmis la fibre pour les mathématiques. Sans elles, je n'aurais probablement pas envisagé d'entreprendre des études en ingénierie.

Enfin, je tiens à exprimer toute ma gratitude à ma mère, la personne qui m'a soutenu inlassablement au fil des années. Ses encouragements constants m'ont poussé à toujours viser plus haut et à me dépasser. Je lui suis particulièrement reconnaissant pour son soutien financier depuis mes débuts à Polytechnique. Merci pour tout, je me compte très chanceux.

RÉSUMÉ

Le calcul haute performance joue un rôle essentiel dans la recherche scientifique en permettant de résoudre des problèmes de plus en plus complexes, mais aussi d'accélérer leur résolution. Ces avantages ne sont toutefois possibles qu'avec une utilisation optimale des ressources de calcul. L'optimisation de boîtes noires, où la fonction objectif et les contraintes sont issues de simulations numériques ou expérimentales, est un domaine spécifique de l'optimisation. Lorsque des simulations numériques interviennent dans le processus d'optimisation, elles nécessitent généralement des temps de calcul importants et des ressources conséquentes, constituant ainsi un facteur limitant pour la résolution de problèmes complexes.

Ce travail combine calcul parallèle et optimisation de boîtes noires à travers le solveur NOMAD (*Nonlinear Optimization by Mesh Adaptive Direct Search*), une implémentation logicielle de l'algorithme `MADS` (*Mesh Adaptive Direct Search*). En premier lieu, une étude numérique des différentes variantes parallèles de l'algorithme `MADS` est menée pour évaluer leurs performances. Ensuite, une modélisation du parallélisme de l'algorithme `MADS` est réalisée dans l'optique de mieux répartir les ressources de calcul entre le solveur d'optimisation et la boîte noire. Cette démarche est motivée par le fait que les boîtes noires exploitent souvent le calcul parallèle, ce qui nécessite une répartition optimale des ressources entre la boîte noire et le solveur pour réduire le temps de calcul tout en obtenant une solution de qualité. Des tests numériques sont alors réalisés sur des problèmes analytiques afin d'étudier l'impact de différentes répartitions des ressources de calcul.

Enfin, un test numérique sur une boîte noire réaliste est effectué avec un code `C++` simulant une centrale solaire (solar), parallélisée via la programmation par mémoire distribuée.

Les résultats obtenus montrent que l'algorithme `MADS` présente une certaine résilience vis-à-vis de la répartition des ressources de calcul en ce qui concerne la qualité de la solution.

ABSTRACT

High-performance computing plays an essential role in scientific research by enabling increasingly complex problems to be solved and accelerating their resolution. These advantages, however, are only achievable with an optimal use of computational resources. Black-box optimization, where the objective function and constraints are derived from numerical or experimental simulations, is a specific branch of optimization. When numerical simulations are involved in the optimization process, they generally require significant computation time and resources, often becoming a limiting factor in solving complex problems.

This work combines parallel computing and black-box optimization through the NOMAD solver (*Nonlinear Optimization by Mesh Adaptive Direct Search*), a software implementation of the MADS algorithm (*Mesh Adaptive Direct Search*). First, a numerical study of the different parallel versions of the MADS algorithm is conducted to evaluate their performance. Next, a parallelism model for the MADS algorithm is developed to improve the allocation of computational resources between the optimization solver and the black box. This approach is motivated by the fact that black boxes often exploit parallel computing, necessitating an optimal allocation of resources between the black box and the solver to reduce computation time while obtaining a high-quality solution. Numerical tests are then conducted on analytical problems to examine the impact of various resource allocation strategies.

Finally, a realistic numerical test is performed using a C++ code that simulates a solar power plant (solar), parallelized via distributed memory programming.

The results show that the MADS algorithm demonstrates resilience with respect to resource allocation, maintaining solution quality.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

ALU	Unité arithmétique et logique (*Arithmetic Logic Unit*)
BBO	Optimisation de boîtes noires (*Blackbox Optimization*)
BE	Barrière extrême
BP	Barrière progressive
CPU	Unité centrale de calcul (*Central Processing Unit*)
CASIR	Calcul scientique de l'Institut de recherche
CS	Méthode de recherche par coordonnées (*Coordinate Search*)
CSRNG	Conseil de recherches en sciences naturelles et en génie du Canada
DERs	Ressources énergétiques distribuées (*Distributed Energy Resources*)
DFO	Optimisation sans dérivées (*Derivative-Free Optimization*)
DSM	Méthode de recherche directe (*Direct Search Method*)
FFT	Transformée de Fourier rapide (*Fast Fourier Transform*)
Flops/s	Nombre d'opérations en virgule flottante par seconde (*Floating Point Operations Per Second*)
GPS	Recherche par motifs généralisée (*General Pattern Search*)
GPU	Processeur graphique (*Graphical Processing Unit*)
HPC	Calcul haute performance (*High Perfomance Computing)*
IREQ	Institut de Recherche d'Hydro-Québec
LES	Simulation aux grandes échelles (*Large Eddy Simulation*)
MADS	Recherche direct sur treillis adaptif (*Mesh Adaptive Direct Search*)
MOSFET	Transistor à effet de champ à grille isolée (*Metal Oxide Semiconductor Field Effect Transistor*)
NOMAD	Optimisation non linéaire par recherche directe sur treillis adaptatif (*Nonlinear Optimization by Mesh Adaptive Direct Search*)
OrthoMADS	MADS avec directions orthogonales (*Orthogonal* MADS)
RISC	Processeur à jeu d'instructions réduit (*Reduced Instruction Set Computer*)
TR	Régions de confiance (*Trust-Region*)

CHAPITRE 1 INTRODUCTION

Nothing takes place in the world whose meaning is not that of some maximum or minimum.

-Leonhard Euler

L'optimisation mathématique est une branche des mathématiques appliquées fournissant un cadre théorique et algorithmique afin de déterminer une solution optimale parmi un vaste ensemble de solutions admissibles. L'utilisation d'une application mobile afin de se rendre à une destination par le plus court chemin est un exemple concret de son utilisation dans notre quotidien. L'étape initiale de ce processus consiste à modéliser le problème, ce qui conduit à sa formulation mathématique et à sa classification. Il est primordial de classer correctement un problème puisqu'il existe une multitude de domaines distincts au sein de l'optimisation mathématique, tous ayant leur propre formalisme, conditions d'optimalité et algorithmes adaptés aux propriétés mathématiques exploitables. Par exemple, l'optimisation combinatoire se focalise sur des problèmes impliquant des ensembles de solutions discrets, en s'appuyant sur des approches efficaces utilisant les graphes. Une classification inadéquate peut alors compromettre la qualité de la solution obtenue et l'efficacité computationnelle.

Ce mémoire considère les problèmes d'optimisation de la forme suivante :

$$\min_{\boldsymbol{x}\in\Omega} f(\boldsymbol{x}), \tag{\textbf{P}}$$

où $\Omega = \left\{\boldsymbol{x} \in \mathcal{X} \;\middle|\; c_j(\boldsymbol{x}) \leq 0,\ j \in \mathcal{J} \triangleq \{1, 2, \ldots, m\}\right\} \subseteq \mathbb{R}^n$ est l'ensemble admissible de solutions et $\mathcal{X} \subseteq \mathbb{R}^n$, est le domaine des variables généralement défini par des contraintes de bornes. Les fonctions $f : \mathcal{X} \to \overline{\mathbb{R}}$ et $c_j : \mathcal{X} \to \overline{\mathbb{R}}$, avec $\overline{\mathbb{R}} \triangleq \mathbb{R} \cup \{\infty\}$ sont évaluées par le biais d'une boîte noire, généralement une expérience de laboratoire ou une simulation numérique.

L'introduction de l'ensemble $\overline{\mathbb{R}}$ est motivée par la possibilité que l'évaluation de la boîte noire échoue. Cette situation est modélisée par un retour à une valeur égale à l'infini pour f et c_j, $j \in \mathcal{J}$.

De par l'absence de formulation analytiques sur f et c_j, $j \in \mathcal{J}$, très peu d'hypothèse sur la régularité et la convexité sont considérées, ce qui réduit les propriétés mathématiques exploitables par un solveur et complexifie grandement la résolution dudit problème d'optimisation.

Ceci nous place dans le cadre de l'optimisation de boîtes noires (*Blackbox Optimization*, BBO), où les méthodes classiques recourant au gradient sont écartées au profit des approches sans dérivées (*Derivative-Free Optimization*, DFO). Pour ce mémoire, les boîtes noires considérées sont uniquement des simulations numériques. Dans ce contexte, un problème-type de BBO présente typiquement les caractéristiques suivantes :

1. Un coût élevé en temps de calcul ;
2. Des besoins importants en mémoire ;
3. Une probabilité d'échec de la simulation ;
4. Une fonction f comportant de nombreux minima locaux et potentiellement affectée par du bruit.

De plus, on suppose que la boîte noire peut être sujette à une hétérogénéité en temps de calcul, en fonction du point à évaluer, comme illustré à la figure 1.1.

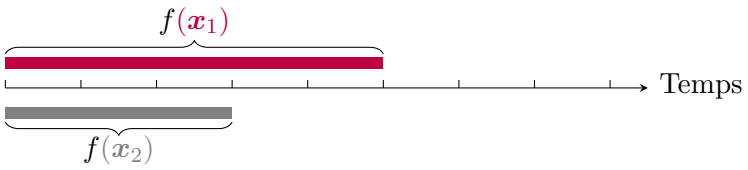

FIGURE 1.1 Boîte noire à temps hétérogène.

Un algorithme BBO doit donc être en mesure de performer en étant sujet à ces défis. Cela souligne une distinction importante à faire entre DFO et BBO. La DFO constitue l'étude mathématique d'algorithmes n'utilisant pas de dérivées [3]. Dans ce cadre, la fonction objectif peut être analytique, telle que $f(\cdot) = |\cdot|$, ce qui diffère radicalement des fonctions objectifs typiques en BBO. Ainsi, l'optimisation de boîte noires est un cas particulier de la DFO.

La BBO connaît un essor majeur depuis les 20 dernières années [4], notamment en raison de l'utilisation accrue des jumeaux numériques en recherche. Les domaines d'applications sont multidisciplinaires [4], en génie biomédical, génie aérospatial, science des matériaux, etc.

Par l'entremise de l'algorithme *Mesh Adaptive Direct Search* (MADS, [5]) et de son implémentation logicielle NOMAD [6], ce travail se concentre sur les méthodes de recherche directe (*Direct Search Methods*, DSM), qui constituent une classe de méthodes sans dérivées. Les DSM permettent de déterminer une direction de descente sans avoir à calculer ou à approximer le gradient, en interagissant uniquement avec les valeurs de la fonction objectif et des contraintes. Cependant, l'absence d'utilisation du gradient, qui est la méthode la plus efficace pour identifier une direction de descente, n'est pas sans conséquence. En effet, l'ensemble des

méthodes DFO présentent pour principal inconvénient le recours à un grand nombre d'évaluations de la boîte noire pour obtenir une convergence. Un grand budget d'évaluations devient alors inconcevable pour des boîtes noires coûteuses en temps de calcul, car le processus entraînerait un temps de résolution inadmissible.

Afin d'accélérer la résolution du problème (**P**), le calcul parallèle apparaît comme une solution viable, car suivant l'avènement du calcul haute performance (*High Performance Computing*, HPC), les ressources informatiques exploitent des architectures multicœurs offrant des possibilités d'accélération considérables.

Ce paradigme de programmation entraîne alors deux considérations importantes dans la résolution du problème (**P**) :

— La majorité des codes de simulation numérique récents exploitent le parallélisme ;

— Les DSM offrent un cadre algorithmique adapté à l'utilisation du calcul parallèle [7].

Par conséquent, le parallélisme peut être appliqué à la fois au sein de la boîte noire et du solveur.

1.1 Question de recherche

Dans un contexte de calcul parallèle, comment répartir les ressources de calcul entre le solveur d'optimisation et la boîte noire pour minimiser le temps de résolution tout en assurant la qualité de la solution ?

1.2 Motivation et objectifs de recherche

Cette recherche est motivée par le projet de l'Alliance CRSNG/MITACS : Optimisation du système énergétique du futur (OSÉF), auquel participent Polytechnique Montréal et l'Institut de recherche d'Hydro-Québec (IREQ). Afin d'intégrer les ressources énergétiques distribuées (*Distributed Energy Resources*, DERs) de manière sécuritaire dans le réseau électrique, des problèmes BBO complexes et coûteux en temps de calcul sont formulés, s'appuyant sur l'utilisation d'éventuels simulateurs parallélisables. Le solveur NOMAD, qui sera employé pour la résolution, bénéficie de versions parallèles développées dans l'optique d'obtenir de bonnes solutions en un temps réduit.

La question de recherche donnée à la section 1.1 émerge alors du fait que l'IREQ dispose d'une grappe de calcul (CASIR) de 5 000 CPU.

L'objectif principal de cette recherche est de vérifier s'il existe un intérêt à développer une stratégie dynamique en allocation des ressources de calcul. Un second objectif, conséquent du

premier est d'évaluer numériquement l'impact du parallélisme sur le comportement de MADS.

En dernier lieu, il convient de souligner que la question de recherche a été peu étudiée. À notre connaissance, un seul article l'aborde étant [8].

1.3 Plan du mémoire

La chapitre 2 propose une revue de littérature sur le calcul parallèle, les stratégies parallèles pour les algorithmes d'optimisation, les méthodes d'optimisation de boîtes noires et une description détaillée de l'algorithme MADS. Le chapitre 3 présente les variantes parallèles de l'algorithme MADS implémentées dans le logiciel NOMAD. Ce chapitre fournit la description algorithmique détaillée de ces variantes, chose qui n'avait pas été faite en littérature pour trois des quatre variantes. Le chapitre 4 présente une modélisation mathématique de la répartition des ressources de calcul entre la boîte noire et le solveur. L'objectif est d'approximer les gains de performance potentiels des différentes répartitions. De plus, deux cas limites sont présentés mettant de l'avant des situations pouvant favoriser une répartition de calcul statique plutôt que dynamique. Le chapitre 5 présente les tests numériques, les métriques de performance utilisées ainsi qu'une analyse des résultats. Enfin, le mémoire se conclut sur le chapitre 6 dédié à la conclusion, où un sommaire des travaux accomplis est donné ainsi que la limitation de ceux-ci et les perspectives de recherche pour des travaux futurs.

CHAPITRE 2 REVUE DE LITTÉRATURE

2.1 Calcul parallèle

La loi de Moore, véritable principe directeur pour l'industrie du semi-conducteur, énonce que le nombre de transistors dans un circuit intégré doublera tous les deux ans. Jusqu'en 2003, cette prévision a tenu grâce à la stratégie de l'échelle de Dennard (*Dennard scaling*) [9]. Celle-ci propose qu'à chaque génération technologique de MOSFET (*Metal Oxide Semiconductor Field Effect Transistor*), si la taille des transistors diminue de 30 %, alors il est possible de doubler le nombre de transistors et d'augmenter leur fréquence de 40 %, sans engendrer une augmentation de la puissance dissipée.

Les programmeurs pouvaient alors compter sur les avancées en matière de « hardware » et de compilateurs pour doubler les performances de leur programme informatique à tous les deux ans, sans avoir à modifier une ligne de code [10].

Cependant, avec la miniaturisation accrue des transistors, les courants de fuite jouent un rôle significatif dans la dissipation thermique, entraînant des exigences de refroidissement inadmissible. Par conséquent, il est devenu impossible de continuer à augmenter les performances d'un processeur en augmentant la fréquence.

Le déclin de l'échelle de Dennard justifie l'adoption d'architectures multicœurs, qui tirent parti de plusieurs unités de calcul (cœurs) afin d'accroître le débit d'instructions par seconde, tout en opérant à basse fréquence, limitant ainsi la dissipation thermique. En date d'aujourd'hui, les performances des processeurs s'améliorent continuellement grâce à l'augmentation du nombre de cœurs. De plus, tous les ordinateurs reposent sur des architectures parallèles, incitant les programmeurs à adapter leur code pour une utilisation maximale de la puissance de calcul disponible. L'ensemble des remarques de cette section est synthétisé dans la figure 2.1.

2.1.1 Architectures parallèles

De par leur grande diversité, les architectures des machines parallèles peuvent être catégorisées de plusieurs manières, selon divers critères. De ce fait, plusieurs taxonomies ont été proposées, notamment celles de Feng [11], Flynn [12,13], Handler [11] et Duncan [14]. Parmi elles, la taxonomie de Flynn est la plus adoptée par la communauté scientifique. Elle caractérise les architectures parallèles, en introduisant deux dimensions indépendantes : le flot d'instructions et le flot de données traités par une unité de calcul. Chacune de ces dimensions

peut être soit unique (*single*), soit multiple (*multiple*), ce qui distingue quatre classes :

1. **SISD - *Single Instruction stream, Single Data stream***
   À l'heure actuelle, il s'agit des processeurs, lorsque considérés comme des ordinateurs séquentiels (monocœur) utilisant un flux unique d'instructions sur un seul ensemble de données, tout en étant capables d'exploiter le parallélisme au niveau des instructions (*Instruction Level Parallelism*, ILP, [9]). En effet, les processeurs modernes sont superscalaires et exploitent le pipelinage ainsi que diverses techniques d'optimisation, ce qui leur permet d'exécuter plusieurs instructions par cycle d'horloge, même dans un contexte de programmation séquentielle.
2. **SIMD - *Single Instruction stream, Multiple Data streams***
   En exploitant le parallélisme sur les données, une même instruction est appliquée sur plusieurs données au cours d'un cycle d'horloge. Les premières implémentations matérielles étaient les processeurs vectoriels, notamment ceux qui ont équipé les supercalculateurs TI ASC [15], CDC STAR-100 [16] et Cray-1 [17], dans les années 70. Les processeurs modernes intègrent des unités de calcul SIMD et des jeux d'instructions permettant de simuler ce fonctionnement, par exemple sur les architectures `x86` avec les jeux d'instructions Intel® Streaming SIMD Extensions (`SSE`) et Intel® Advanced Vector Extensions (`AVX`). Les processeurs graphiques (*Graphical Processing Unit*, `GPU`) sont un second exemple actuel de matériel exploitant le SIMD.
3. **MISD - *Multiple Instruction streams, Single Data stream***
   Les architectures MISD sont considérées comme essentiellement théoriques, car elles n'ont jamais été mises en œuvre via des implémentations matérielles commerciales [18]. Pour certains, leur présence dans la taxonomie de Flynn ne vise qu'à compléter la matrice, afin d'accentuer le cadre conceptuel.
4. **MIMD - *Multiple Instruction streams, Multiple Data streams***
   Cette configuration a la capacité d'exécuter plusieurs flux d'instructions distincts, chacun avec son propre flux de contrôle, opérant sur des ensembles de données différents. Elle caractérise à la fois l'architecture multicœur au sein d'un même processeur, l'intégration de plusieurs processeurs dans un même ordinateur, ainsi que l'intégration de plusieurs ordinateurs (nœuds) dans une grappe de calcul (superordinateur). De plus, en fonction de la manière dont les données sont accessibles entre les différents éléments de calculs, on distingue les architectures MIMD à mémoire partagée et MIMD à mémoire distribuée.

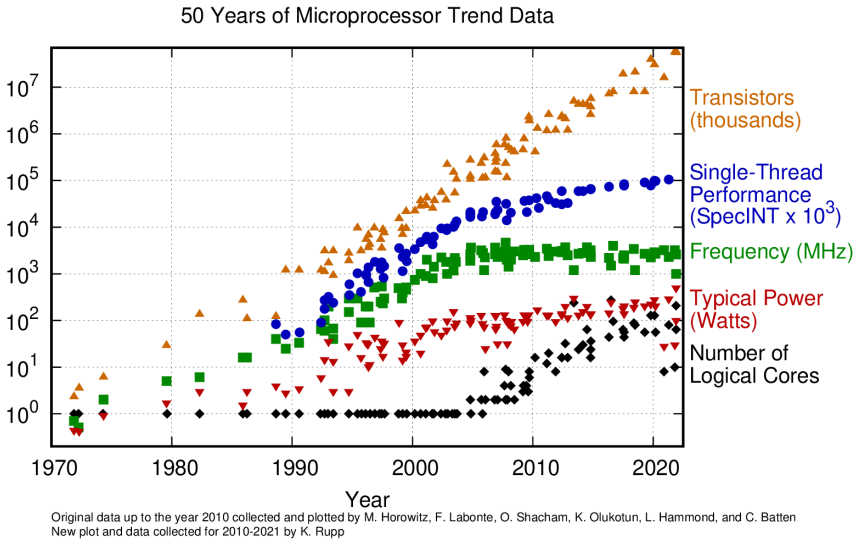

FIGURE 2.1 Évolution technologique des microprocesseurs (crédit image : https://www.karlrupp.net/).

2.1.2 Plateformes de calcul haute performance

Le calcul haute performance consiste à associer un grand nombre de cœurs de calcul, allant de plusieurs milliers à plusieurs millions, pour concevoir des architectures distribuées dites massivement parallèles. Dans le domaine du calcul scientifique, la puissance de calcul théorique d'un système est mesurée en nombre d'opérations en virgule flottante par seconde (*floating point operation per second*, `Flops/s`). Cette métrique reflète à la fois la capacité d'un système à effectuer des calculs complexes rapidement et son degré de parallélisme. En raison de leur constante évolution, les architectures `HPC` sont classées en fonction du nombre de `Flops/s`. Actuellement, les superordinateurs les plus puissants du Top500 [1] sont classifiés comme des architectures *Exascale*, avec une capacité de calcul de l'ordre de 10^{18} `Flops/s`. Ce type d'architecture a été introduit en 2021 avec le supercalculateur Frontier [19] développé au laboratoire national d'Oak Ridge (ORNL). Leur puissance de calcul, nettement supérieure à celle des précédentes architectures *Petascale* (10^{15}), est principalement due à l'intégration importante de `GPU` dans les nœuds de calcul.

1. https://top500.org/

2.1.3 Métriques de performance

Lorsqu'un utilisateur dispose d'un environnement de calcul massivement parallèle, une question clé se pose : combien de ressources de calcul sont nécessaires pour atteindre une accélération optimale ? L'enjeu est avant tout environnemental, car les supercalculateurs consomment une grande quantité d'énergie. L'objectif est donc d'utiliser le minimum de ressources en mesure d'apporter une accélération significative, tout en évitant de mobiliser des ressources supplémentaires qui, bien qu'inutiles pour le calcul, augmenteraient la consommation énergétique. Cette section présente les principales métriques permettant d'évaluer la performance d'un code de calcul parallèle.

Granularité

En calcul parallèle, un compromis fréquent concerne le surcoût lié à la parallélisation (*parallel overhead*) par rapport à la charge de calcul. Ce surcoût non négligeable inclut le temps occasionné par : (i) les mécanismes de synchronisation, (ii) la communication de données et (iii) la mise en place de l'environnement parallèle par le compilateur, la librairie et le système d'exploitation. En somme, il représente la durée totale durant laquelle aucun calcul n'est effectué. L'objectif est donc de minimiser ce temps, ce qui nécessite l'adoption de bonnes pratiques en programmation parallèle.

Une métrique quantifiant le compromis est la granularité, qui est une mesure qualitative du rapport entre le calcul et la communication. On distingue alors deux types de parallélisme, à savoir : le parallélisme à grain fin et le parallélisme à grain grossier. Le parallélisme à grain fin à un rapport faible de calcul par rapport à la communication, puisque entre chaque évènement de communication, très peu de calcul sont exécutés. À l'opposé, le parallélisme à grain grossier exécute un grand nombre de calculs avant chaque évènement de communication.

La granularité guide alors bien souvent la conception des algorithmes parallèles, en tenant compte des modèles de programmation parallèles et de leur architecture matérielle associée. La section 2.1.4 sera consacrée à ce sujet.

Évaluation des performances

La performance d'un algorithme parallèle est principalement mesurée par sa capacité à réduire le temps d'exécution en fonction des ressources de calcul, ce qui renvoie à la notion de mise à l'échelle (*scalability*) d'un programme parallèle. Deux métriques sont utilisées pour mesurer cette dernière.

La première métrique mesure l'accélération (*speedup*) obtenue par la parallélisation d'un programme :

$$S(\eta, p) \triangleq \frac{T_1(\eta)}{T_p(\eta)}, \tag{2.1}$$

avec η la taille du problème résolu, $T_1(\eta)$ le temps d'exécution de l'algorithme séquentiel et $T_p(\eta)$ le temps d'exécution de l'algorithme parallèle utilisant p unités de calculs. Il est également possible de choisir $T_1(\eta)$ comme le temps d'exécution de l'algorithme parallèle utilisant une unité de calcul. Dans ce cas, on parlera d'accélération relative plutôt qu'absolue.

La seconde métrique mesure l'efficacité (*efficiency*) obtenue par la parallélisation :

$$E(\eta, p) \triangleq \frac{S(\eta, p)}{p}. \tag{2.2}$$

Grâce à l'efficacité, il est possible de quantifier l'apport que les p unités de calcul ont sur l'accélération et donc implicitement d'évaluer dans quelle mesure ces unités sont utilisées à faire du calcul.

Les principales motivations incitant à l'utilisation du calcul parallèle sont : (i) accélérer le temps de calcul d'un problème donné et (ii) résoudre des problèmes de plus grande taille ou de plus grande complexité computationnelle. De ce fait, l'étude de la mise à l'échelle d'un programme peut être réalisée via des modèles, de manière à mesurer la capacité de ce dernier à satisfaire (i) ou (ii).

Un modèle mesurant (i) est dit à taille fixe (*fixed-size models*). Le plus répandu dans la communauté scientifique est celui d'Amdahl [20]. La loi d'Amdahl mesure la capacité d'un programme à accélérer le temps de résolution lorsque p augmente et que η reste fixe. Dans ce cadre, la loi d'Amdahl offre une borne supérieure sur l'accélération, soit l'accélération maximale :

$$S(\eta, p) \leq \frac{1}{f + (1 - f)/p}, \tag{2.3}$$

avec f la fraction non parallélisable du programme et $1 - f$ la fraction parallélisable du programme par p unités de calcul. À titre d'exemple, si 20% d'un algorithme n'est pas parallélisable, l'accélération est alors limitée à cinq, indépendamment du nombre d'unités de calcul utilisées.

À l'opposé, un modèle mesurant (ii) est dit à taille variable (*scaled-size models*). La loi de Gustafsson-Barsis [21] mesure la capacité d'un programme à garder un temps d'exécution constant en faisant augmenter η et p de manière proportionnelle [2]. Dans ce contexte, l'accé-

2. La charge de calcul, par unité de calcul doit rester constante.

lération maximale est :

$$S(\eta, p) \leq p + (1 - p)s, \tag{2.4}$$

avec s la fraction parallélisable du programme.

Ces deux modèles ne tiennent pas compte du surcoût occasionné par la parallélisation, ce qui fait qu'en pratique l'accélération mesurée sera inférieure à la borne fournit.

2.1.4 Modèles de programmation parallèles

Dans l'optique d'exploiter pleinement la capacité de calcul d'une architecture *Exascale*, trois modèles de programmation parallèles doivent être sollicités : la programmation en mémoire distribuée, la programmation en mémoire partagée et la programmation sur `GPU`. Afin de s'en convaincre, on prend en exemple l'architecture du supercalculateur Frontier mentionné plus tôt à la section 2.1.2. Au total, 9 408 nœuds de calcul composent le supercalculateur. Chaque nœud est constitué de :

1. 4 `GPU AMD Instinct MI250X` avec les caractéristiques suivantes :
   (a) 220 cœurs de calcul ;
   (b) Performance de pointe de 47,9 `TFlops/s` en double précision (FP64).
2. 1 `CPU AMD EPYC` avec les caractéristiques suivantes :
   (a) 64 cœurs de calcul ;
   (b) Performance de pointe de 2,0 `TFlops/s` en double précision (FP64).

Cela signifie que la puissance totale de calcul de $1,8214 \times 10^{18}$ `Flops/s` est répartie entre $1,8026 \times 10^{18}$ `Flops/s` provenant des `GPU` et seulement $0,018816 \times 10^{18}$ `Flops/s` provenant des `CPU`. Concernant les cœurs de calcul, parmi les 8 881 152 disponibles 8 279 040 proviennent des `GPU`, tandis que 602 112 proviennent des `CPU`.

Le modèle de programmation à mémoire partagée permettrait l'utilisation d'un unique nœud de calcul, soit 64 cœurs d'un `CPU` et 2,0 `TFlops/s`. Le modèle de programmation à mémoire distribuée permettrait l'utilisation de l'ensemble des `CPU` donc 602 112 cœurs et $0,018816 \times 10^{18}$ `Flops/s`. Finalement, le modèle de programmation sur `GPU` permettrait l'utilisation des quatre `GPU` d'un nœud de calcul, soit 880 cœurs et 191,6 `TFlops/s`.

On peut conclure qu'il est impératif d'utiliser les trois modèles pour avoir l'occasion d'exploiter la pleine puissance de calcul.

Programmation en mémoire distribuée

Ce paradigme permet l'utilisation de plusieurs nœuds de calcul ayant des espaces mémoires non communs. Par le biais de protocoles de communication, les différents nœuds de calcul peuvent s'échanger des données sur un réseau de communication, puisque chaque nœud n'a pas accès à l'espace mémoire des autres nœuds. Les communications sont coûteuses en temps (latence élevée), de sortes que l'on privilégie les applications à gros grains plutôt que celles à grains fins. Ce modèle est la référence en matière de décomposition de domaines [22], par exemple pour des simulations de mécanique des fluides numérique où un maillage peut être décomposé en plusieurs sous-domaines de calcul qui seront assignés à différents nœuds de calculs. En dernier lieu, grâce à ce modèle, la mise à l'échelle d'un grand nombre d'unités de calcul est possible. La bibliothèque de référence est `MPI`. Il existe toutefois différents langages de programmation dédiés à ce paradigme, soit `Chapel`, `UPC` et `Charm++`.

Programmation en mémoire partagée

Les applications à grains fins nécessitent une faible latence afin de minimiser les délais occasionnés par la communication. Avec le modèle de mémoire partagée, les unités de calcul ont accès à une mémoire commune locale par l'intermédiaire d'un médium de communication très rapide, à savoir le bus de données, ce qui permet d'obtenir une latence faible de communication.

Cependant, comme l'accès au bus est commun, il existe un risque de contention rapide pour toutes les unités de calcul, ce qui peut souvent se manifester comme un goulot d'étranglement. De plus, étant donné que la mémoire est partagée et qu'un certain nombre d'unités de calcul y ont accès, il est nécessaire d'appliquer un protocole de cohérence de la mémoire pour garantir que, lorsque des données sont modifiées, toutes les unités aient une vue identique du contenu de la mémoire. Sans l'application de la cohérence de mémoire, il ne serait pas possible d'avoir un programme parallèle déterministe.

La contention du bus de données et la cohérence de la mémoire sont les facteurs limitant la mise à l'échelle d'un grand nombre unité de calcul via ce paradigme. Les standards de programmation pour l'architecture en mémoire partagée incluent `OpenMP`, `OpenTBB` et `POSIX Threads`.

Programmation sur processeur graphique

Ce modèle de programmation est adapté pour du parallélisme massif de données. Cela est dû au fait que l'architecture des `GPU` intègre un grand nombre d'`ALU` (*Arithmetic Logic Unit*)

de l'ordre du millier comparativement aux CPU qui en intègrent rarement plus de 64. Un grand nombre d'unités de calcul est alors disponible, pour traiter des données. Afin d'obtenir des gains de performance considérables, le code doit être implémenté de façon à limiter les nombreux branchements (`if, else`), avoir une intensité arithmétique [3] élevée en raison de la latence importante des accès mémoires et finalement favoriser du calcul en simple précision afin de maximiser le nombre de données traitées simultanément et d'éviter une saturation hâtive de la mémoire. Les problèmes types sont issus du calcul scientifique, par exemple : (i) les opérations matricielles sur des matrices denses de grande taille, (ii) les simulations de Monte Carlo et (iii) l'algorithme de Transformée de Fourier rapide (*Fast Fourier Transform*, FFT).

Enfin, il existe un large éventails de plateformes dédiées à la programmation de ces accélérateurs tels que `oneAPI`, `CUDA`, `ROCm`, `Kokkos`, `OpenCL`, `OpenACC`, `RAJA` et `SYCL`. Le portage de code sur `GPU` demande bien souvent une réécriture complète comparativement à la programmation en mémoire partagée. Il s'agit du prix à payer pour obtenir des facteurs d'accélération pouvant dépasser 100.

En alliant les trois modèles, il est alors possible d'obtenir un code adapté à l'*Exascale*. Un exemple est `MFEM` [23, 24], une bibliothèque libre d'accès dédiée à la résolution d'équations aux dérivées partielles, par la méthode des éléments finis.

2.2 Méthodes parallèles pour `DSM`

Comme mentionné dans la section précédente, les scientifiques disposent d'une importante puissance de calcul. Bien qu'il soit possible d'en tirer parti grâce à certains paradigmes de programmation parallèle, le véritable enjeu réside dans la conception d'algorithmes adaptés à ces paradigmes. Dans cette section, il sera discuté de différentes stratégies algorithmiques enclines à l'exploitation du parallélisme, pour les `DSM`.

Un principe directeur pour une utilisation efficace du parallélisme dans la résolution du problème (**P**) est de considérer que le temps de calcul d'une évaluation de la boîte noire est nettement plus élevé que celui consacré à l'algèbre linéaire pour produire les solutions candidates. Par conséquent, la parallélisation la plus susceptible d'entraîner une grande accélération consiste à effectuer plusieurs évaluations simultanées de la boîte noire et/ou à mettre en œuvre une parallélisation interne de celle-ci [25]. Cette approche est commune aux méthodes présentées ci-dessous, décrites en détail dans [26], qui propose une revue de littérature récente ainsi qu'un cadre général pour le parallélisme en recherche opérationnelle. Bien que

3. Rapport du nombre d'opérations arithmétiques en virgule flottante par rapport aux accès mémoire.

ces stratégies soient formulées dans un contexte de recherche opérationnelle, leur domaine d'application peut être étendu aux DSM.

2.2.1 Parallélisme de données et de tâches

Cette stratégie repose sur deux approches de parallélisme. La première exploite le parallélisme au sein de la boîte noire pour réduire le temps de calcul, ce qui est particulièrement efficace pour les codes numériques très exigeants en temps CPU. On distingue alors deux types de parallélisme. Lorsque la boîte noire évalue simultanément plusieurs ensembles de données, comme dans une simulation de Monte-Carlo, on parle de parallélisme de données. En revanche, lorsque la boîte noire exécute plusieurs tâches indépendantes les unes des autres, on parle de parallélisme de tâches. Cette situation est courante dans les problèmes multiphysiques couplés, tels que la modélisation climatique [27].

La seconde approche répartit l'évaluation des solutions candidates sur les ressources de calcul afin de réaliser une parallélisation de données. Ces deux méthodes sont classifiées comme du **parallélisme intra-algorithme à grain fin**.

2.2.2 Parallélisme par recherche multiple

L'approche suivante consiste à effectuer des explorations simultanées dans l'espace des variables à l'aide de différents algorithmes paramétrés. Par exemple, on pourrait utiliser des algorithmes de régions de confiance avec différents paramètres seuil $\eta \in [0, \frac{1}{4})$. L'utilisation de paramètres distincts permet à chaque algorithme d'adopter un comportement différent et d'explorer ainsi une région différente de l'espace de solutions. Une première forme d'implémentation ne permet aucun échange d'information entre les instances pendant l'entièreté de l'exécution. Cette stratégie de parallélisation est appelée **recherche multiple indépendante**. La seconde méthode favorise l'échange d'information via des mécanismes de coopération, une stratégie désignée comme **recherche multiple coopérative**. Ces deux méthodes sont qualifiées de **parallélisme inter-algorithme à gros grain**.

2.2.3 Parallélisme par décomposition de domaine

Cette approche est conçue pour traiter des problèmes d'optimisation de grande taille. Elle décompose un problème de grande dimension en une série de sous-problèmes de plus petite dimension, plus faciles à résoudre. Ces sous-problèmes étant indépendants, ils peuvent être résolus en parallèle. Cette stratégie est classifiée comme du parallélisme **intra-algorithme à grain grossier**.

Plusieurs techniques peuvent être utilisées afin de déterminer les sous-problèmes. La première consiste à fragmenter l'espace des variables, en assignant à chaque sous-problème, un sous-espace de variables distinct $\mathcal{X}' \subseteq \mathcal{X}$. La seconde méthode consiste à distribuer les variables entre les sous-problèmes [28]. Chaque sous-problème se charge ensuite de mettre à jour son propre bloc de variables en parallèle. Généralement dans les deux approches, à chaque itération k, une étape de synchronisation a lieu où les solutions partielles de chaque sous-problème sont utilisées pour reconstruire une solution complète du problème original.

2.2.4 Parallélisme hybride

Les trois stratégies de parallélisation présentées sont mutuellement compatibles. Il est possible de combiner différentes de ces stratégies afin d'en concevoir un algorithme hybride comme illustré dans [29].

2.3 Méthodes pour l'optimisation de boîtes noires (BBO)

Les méthodes de BBO peuvent être divisées en deux familles, selon Audet et Hare [3] : les méthodes heuristiques et les méthodes dotées d'une analyse de convergence. Les méthodes heuristiques n'offrent aucune garantie d'optimalité, car elles ne reposent pas sur un critère permettant une analyse de convergence. La solution obtenue ne satisfait donc aucune condition nécessaire d'optimalité.

2.3.1 Méthodes heuristiques

Ces méthodes intègrent souvent une composante probabiliste dans leur cadre algorithmique. Leur popularité s'explique par leur simplicité d'implémentation et leur compréhension plutôt intuitive. De manière générale, ces algorithmes possèdent une structure facilement parallélisable adaptée à de nombreux paradigmes de programmation parallèle, notamment la programmation sur processeur graphique [30]. La littérature sur le parallélisme d'heuristiques est abondante et mature puisque ce sujet est étudié depuis plus de 20 ans.

Les stratégies évolutionnaires [31] sont un exemple d'heuristiques utilisées en BBO. Ces dernières s'inspirent de la théorie de l'évolution de Darwin et ont la capacité d'explorer globalement l'espace de solutions, leur permettant alors de détecter plusieurs minima locaux. Toutefois, leur processus requiert un budget conséquent en termes d'évaluations de la boîte noire pouvant, alors occasionner une limitation selon le temps de calcul requis d'une évaluation.

L'algorithme évolutionnaire *Covariance Matrix Adaptation Evolution Strategy* (`CMA-ES`, [32]) est une référence en `BBO`. Une implémentation parallèle [33] pour le calcul haute performance a été prouvée efficace jusqu'à 6144 cœurs de calcul, sur le supercalculateur Fugaku [34].

2.3.2 Méthodes basées sur des modèles

Cette approche vise à pallier l'absence d'accès aux dérivées en `DFO`, par la construction d'un modèle local lisse de la fonction objectif. Le modèle est ensuite utilisé par une méthode utilisant le gradient, qui à chaque itération k détermine une solution candidate $\boldsymbol{x}^k$ où évaluer la boîte noire.

Les méthodes basées sur des modèles ne présentent pas en général une structure favorable à l'exploitation du calcul parallèle. Elles peuvent être classées en deux familles : (i) les méthodes de régions de confiance (*Trust-Region*, `TR`, [35]) et (ii) les méthodes de *linesearch*.

L'idée d'une `TR` est de construire à chaque itération k un modèle local m^k (généralement quadratique) dans une boule $\mathcal{B}(r^k, \boldsymbol{x}^k)$ de rayon r^k centrée en la solution courante $\boldsymbol{x}^k$. Le modèle m^k est ensuite minimisé dans la boule, ce qui permet d'obtenir la prochaine solution candidate $\boldsymbol{x}^{k+1} \triangleq \arg\min_{\boldsymbol{x}\in\mathcal{B}(r^k,\ \boldsymbol{x}^k)} m^k(\boldsymbol{x})$. La boule $\mathcal{B}(r^k,\ \boldsymbol{x}^k)$ est appelée région de confiance, car nous estimons que m^k fournit une approximation précise de f dans cette zone. À la fin de l'itération k, la valeur de r^k est ajustée, selon la précision de m^k mesurée à l'itération k.

Les méthodes de *linesearch* utilisent le modèle afin d'approximer le gradient de f pour estimer une direction de descente $\boldsymbol{d}^k$ à chaque itération k. Une recherche linéaire est ensuite réalisée afin de déterminer la longueur de pas optimal α^k à prendre suivant $\boldsymbol{d}^k$. Le point candidat est alors obtenu comme : $\boldsymbol{x}^{k+1} \triangleq \boldsymbol{x}^k + \alpha^k \boldsymbol{d}^k$.

Généralement, les méthodes basées sur des modèles sont très efficaces lorsque les fonctions sont lisses (même lorsque les gradients ne sont pas disponibles analytiquement).

2.3.3 Méthodes de recherche directe (`DSM`)

Ces méthodes itératives se basent exclusivement sur les valeurs de sortie de la boîte noire pour guider le processus d'optimisation, sans tenter d'approximer le gradient ni de construire un modèle. Cela les rend particulièrement efficaces pour les problèmes non lisses de `BBO`, en plus d'être supportées par une analyse de convergence pour des fonctions non différentiables.

Chaque itération, d'un algorithme `DSM` est généralement séparée en deux étapes : (i) l'étape de recherche globale (`SEARCH`) et (ii) l'étape de recherche locale (`POLL`). La `SEARCH` est une étape flexible qui permet la localisation de régions prometteuses dans l'espace des solutions,

ce qui peut significativement accélérer la convergence vers une solution. De plus, elle confère à l'algorithme une capacité à s'échapper de minima locaux. À travers cette étape, un utilisateur peut intégrer sa connaissance approfondie du problème au sein du processus d'optimisation. À l'opposé, la `POLL` est une étape avec un cadre rigide, puisque c'est par l'intermédiaire de celle-ci que l'analyse théorique est construite.

Partant d'un point initial $\boldsymbol{x}^0$, ces méthodes génèrent à chaque itération k un ensemble fini de points candidats, avec pour objectif d'améliorer la solution courante $\boldsymbol{x}^k$. Ces points sont générés autour de la solution $\boldsymbol{x}^k$ à l'aide d'une recherche locale guidée par un ensemble fini de directions généré à chaque itération k.

Le terme *direct search* a été introduit par Hooke et Jeeves [36]. La première méthode pionnière est celle de Fermi et Metropolis via l'algorithme *Coordinate Search* (`CS`, [37]) en 1952. Une évolution de `CS` soit l'algorithme *General Pattern Search* (`GPS`, [38]) est proposé en 1997. En 2006, l'algorithme *Mesh Adaptive Search* est proposé, représentant la version la plus avancée des *Pattern Search Methods*. Comme mentionné en introduction, `MADS` est l'algorithme d'intérêt de ce manuscrit et donc une attention particulière lui est accordée à la section 2.3.4.

L'élément différenciateur principal entre les méthodes `CS`, `GPS` et `MADS` réside dans l'ensemble de directions. En effet, chacune de ces méthodes établit son ensemble de directions de manière différente, ce qui impacte alors la façon dont l'exploration locale de l'espace des variables est effectuée.

Un ensemble riche en direction est plus susceptible d'éviter qu'une méthode se retrouve coincée hâtivement dans un minimum local, ce qui est le problème principal des méthodes locales. Afin de former un ensemble de directions varié, certaines définitions sont introduites.

Définition 2.1 (Ensemble générateur positif). Soit $\mathbb{D} \subseteq \mathbb{R}^n$. L'espace engendré par l'ensemble des combinaisons linéaires positives des vecteurs de $\mathbb{D}$ est noté pspan($\mathbb{D}$) :

$$\text{pspan}(\mathbb{D}) = \left\{ \sum_{i=1}^{m} \lambda_i \boldsymbol{d}^i \;\middle|\; \lambda_i \geq 0,\ \boldsymbol{d}^i \in \mathbb{D},\ m \in \mathbb{N} \right\} \subseteq \mathbb{R}^n,$$

L'ensemble $\mathbb{D}$ est un ensemble générateur positif si et seulement si pspan($\mathbb{D}$) $= \mathbb{R}^n$.

Un ensemble générateur positif est donc un ensemble composé de suffisamment de vecteurs pour engendrer positivement tout $\mathbb{R}^n$. Cependant, il peut inclure des vecteurs redondants. On souhaiterait plutôt trouver le nombre minimal de vecteurs composant $\mathbb{D}$ de sortes à générer tout $\mathbb{R}^n$. Ceci nous motive à introduire le concept d'ensemble positivement linéairement indépendant.

Définition 2.2 (Indépendance linéaire positive [3]). Un ensemble $\mathbb{D}$ de vecteurs est positivement linéairement indépendant si et seulement si $\boldsymbol{d} \notin \text{pspan}(\mathbb{D} \setminus \{\boldsymbol{d}\})$ pour tout $\boldsymbol{d} \in \mathbb{D}$.

En dernier lieu, si un ensemble est à la fois un ensemble générateur positif et est positivement linéairement indépendant, il constitue une base positive. La cardinalité d'une base positive est donnée par le théorème ci-contre.

Théorème 2.1 (Base Positive Maximale [3], **Théorème 6.2**). Soit $\mathbb{D}$ une base positive de $\mathbb{R}^n$. Alors $\mathbb{D}$ contient au moins $n+1$ vecteurs et au plus $2n$ éléments.

2.3.4 Recherche directe par treillis adaptatif

Parmi les algorithmes `DSM` présentés, `MADS` est le seul à pouvoir traiter des problèmes avec contraintes d'inégalité.

À des fins d'analyse de convergence, `MADS` introduit la notion de treillis, qui correspond à un ensemble discret et infini de points pouvant être sélectionnés. La géométrie du treillis est étroitement liée au choix d'une matrice de base positive, notée $\mathbf{D} = \mathbf{GZ} \in \mathbb{R}^{n \times p}$, où $\mathbf{G} \in \mathbb{R}^{n \times n}$ est une matrice inversible et $\mathbf{Z} \in \mathbb{Z}^{n \times p}$ a ses colonnes qui forment une base positive pour $\mathbb{R}^n$.

Définition 2.3 (Treillis [3]). Soit $\mathcal{M}^k \subset \mathbb{R}^n$ le treillis à l'itération k défini comme l'ensemble :

$$\mathcal{M}^k \triangleq \left\{ \boldsymbol{x}^k + \delta^k \mathbf{D}\boldsymbol{y} \;\middle|\; \boldsymbol{y} \in \mathbb{N}^p \right\},$$

où $\delta^k > 0$ est la taille du treillis.

Le paramètre de taille du treillis δ^k correspond au plus petit incrément entre deux points de $\mathcal{M}^k$. En ajustant sa valeur, le treillis peut être raffiné ou grossi. De plus, il est possible d'utiliser différentes valeurs de δ^k, afin de créer un treillis adaptatif. Il devient alors possible de traiter des variables granulaires, c.-à-d., des variables avec un nombre de décimales contrôlé [39]. Étant donné que le treillis couvre l'ensemble de l'espace des variables, sa cardinalité est infinie. Pour cette raison, l'étape de `POLL` explore localement $\mathcal{M}^k$ autour d'une solution courante $\boldsymbol{x}^k$. La notion de localité est définie par Δ^k, le paramètre de taille du cadre, qui délimite le sous-espace $\mathcal{F}^k$ dans lequel la `POLL` sélectionne les points d'essai.

Définition 2.4 (Cadre [3]). Soit $\mathcal{F}^k \subseteq \mathcal{M}^k$ le cadre à l'itération k défini comme l'ensemble :

$$\mathcal{F}^k \triangleq \left\{ \boldsymbol{x} \in \mathcal{M}^k \;\middle|\; \|\boldsymbol{x} - \boldsymbol{x}^k\|_\infty \leq b\Delta^k \right\},$$

où $b = \max_{d' \in \mathbb{D}} \|d'\|_\infty$ et $\mathbb{D}$ est l'ensemble contenant les colonnes de $\mathbf{D}$.

Enfin, une fois la zone d'exploration locale définie, il ne reste plus qu'à sélectionner les directions pour générer les points candidats de l'itération k. Cela implique la formation d'un ensemble générateur positif $\mathbb{D}_\Delta^k$, permettant à la sonde de restreindre son exploration de $\mathcal{F}^k$ à un sous-espace noté $\mathcal{P}^k$.

Définition 2.5 (Sonde [3]). Soit $\mathcal{P}^k \subset \mathcal{F}^k$ la sonde à l'itération k est défini par l'ensemble suivant :

$$\mathcal{P}^k \triangleq \left\{ \boldsymbol{x}^k + \delta^k \boldsymbol{d} \;\middle|\; \boldsymbol{d} \in \mathbb{D}_\Delta^k \right\},$$

où $\mathbb{D}_\Delta^k$ est l'ensemble générateur positif à l'itération k.

La construction de $\mathbb{D}_\Delta^k$ est faite par l'algorithme `OrthoMADS` [40] qui génère à chaque itération k, un vecteur normalisé $\boldsymbol{v}^k$ de manière aléatoire étant élément de la suite $\{\boldsymbol{v}^k\}_{k=0}^\infty$ qui est dense dans la sphère unitaire. Ce vecteur est ensuite utilisé afin de générer la matrice de Householder $\mathbf{H}^k$.

Définition 2.6 (Matrice de Householder [41]). Soit $\boldsymbol{v} \in \mathbb{R}^n$ un vecteur normalisé. La matrice de Householder $\mathbf{H}$ associée à $\boldsymbol{v}$ est :

$$\mathbf{H} \triangleq \mathbf{I} - 2\boldsymbol{v}\boldsymbol{v}^T \in \mathbb{R}^{n \times n},$$

où $\mathbf{I}$ est la matrice identité $n \times n$.

Les colonnes de la matrice $\mathbf{H}^k = [\boldsymbol{h}_1 \; \boldsymbol{h}_2 \; ... \; \boldsymbol{h}_n]$ obtenue subissent une transformation, ce qui construit l'ensemble $\mathbb{B}^k \triangleq \{\boldsymbol{b}_1, \boldsymbol{b}_2, ..., \boldsymbol{b}_n\}$ avec :

$$\boldsymbol{b}_j \triangleq \texttt{round}\left(\frac{\Delta^k}{\delta^k} \frac{\boldsymbol{h}_j}{\|\boldsymbol{h}_j\|_\infty} \right) \in \mathbb{Z}^n \text{ pour tout } j \in \{1, \ldots, n\}.$$

Finalement, $\mathbb{D}_\Delta^k \triangleq \mathbb{B}^k \cup (-\mathbb{B}^k)$ constitue une base positive maximale.

À chaque itération k, les paramètres δ^k et Δ^k sont redimensionnés en respectant $0 < \delta^k \leq \Delta^k$. En faisant diminuer δ^k plus rapidement que Δ^k, cela permet de générer des directions qui sont denses dans la sphère unitaire, au fil des itérations k, étant un élément essentiel à l'analyse de convergence de `MADS`. Une description détaillée de `MADS` pour le cas non contraint est donnée par l'algorithme 1, ainsi qu'une représentation visuelle de l'étape de `POLL` à la figure 2.2.

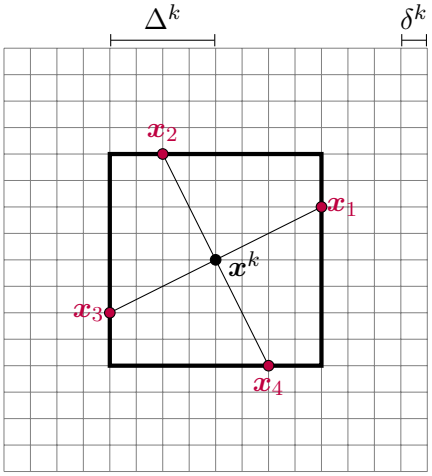

FIGURE 2.2 Treillis $\mathcal{M}^k$ (—), le cadre $\mathcal{F}^k$ (□) et la sonde $\mathcal{P}^k = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4\}$ (•) générés autour du point courant $\boldsymbol{x}^k$ (•). Inspirée de [1].

Algorithme 1 : Recherche directe par treillis adaptatif (`MADS`)

Suivant $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$, $\boldsymbol{x}^0 \in \mathbb{R}^n$
Initialisation
  $\mathbf{D} = \mathbf{GZ}$, $\Delta^0 > 0$, $\tau \in (0,1) \cap \mathbb{Q}$
for $k = 0, 1, 2, \ldots$ **do**
  $\delta^k = \min\{\Delta^k, (\Delta^k)^2\}$
  `SEARCH` (optionnel)
    Construire un ensemble fini de points $\mathcal{S}^k \subset \mathcal{M}^k$
    if $\exists \boldsymbol{x} \in \mathcal{S}^k \mid f(\boldsymbol{x}) < f(\boldsymbol{x}^k)$; **then**
      $\boldsymbol{x}^{k+1} \leftarrow \boldsymbol{x}$, $\Delta^{k+1} \leftarrow \tau^{-1}\Delta^k$, aller à **Terminaison**
    end
  `POLL`
    Choisir $\mathbb{D}_\Delta^k$ pour construire $\mathcal{P}^k$
    if $\exists \boldsymbol{x} \in \mathcal{P}^k \mid f(\boldsymbol{x}) < f(\boldsymbol{x}^k)$; **then**
      $\boldsymbol{x}^{k+1} \leftarrow \boldsymbol{x}$, $\Delta^{k+1} \leftarrow \tau^{-1}\Delta^k$
    else
      $\boldsymbol{x}^{k+1} \leftarrow \boldsymbol{x}^k$, $\Delta^{k+1} \leftarrow \tau\Delta^k$
    end
  Terminaison
    if CRITÈRE ARRÊT ; **then** Terminer `MADS`
end

Dans les prochaines lignes, on présente brièvement les éléments nécessaires afin de présenter le résultat de convergence de `MADS` pour les cas non contraint et contraint donné respectivement par la proposition 2.2 et 2.3.

Définition 2.7 (Continuité Lipschitz). La fonction $g : \mathbb{R}^n \to \mathbb{R}$ est dite Lipschitz continue sur l'ensemble $\mathcal{X} \subseteq \mathbb{R}^n$ si et seulement si, il existe un scalaire $L > 0$ tel que :

$$\forall (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X}^2, \quad |g(\boldsymbol{x}) - g(\boldsymbol{y})| \leq L \|\boldsymbol{x} - \boldsymbol{y}\|_2.$$

Définition 2.8 (Dérivée de Clarke). Pour tout $\boldsymbol{x} \in \mathbb{R}^n$ et $\boldsymbol{d} \in \mathbb{R}^n$, la dérivée de Clarke de $f : \mathbb{R}^n \to \mathbb{R}$ en $(\boldsymbol{x}; \boldsymbol{d})$ est

$$f_{\mathsf{C}}^{\circ}(\boldsymbol{x}; \boldsymbol{d}) \triangleq \limsup_{\boldsymbol{y} \to \boldsymbol{x},\ h \searrow 0} \frac{f(\boldsymbol{y} + h\boldsymbol{d}) - f(\boldsymbol{y})}{h}.$$

Pour le cas non contraint, `MADS` fournit un point stationnaire au sens de la dérivée de Clarke soit la proposition 2.2.

Proposition 2.2. Soit $\boldsymbol{x} \in \mathbb{R}^n$ et $f : \mathbb{R}^n \to \mathbb{R}$ Lipschitz continue en $\boldsymbol{x}$. Si $\boldsymbol{x}$ est un minimiseur local de f, alors $f_{\mathsf{C}}^{\circ}(\boldsymbol{x}; \boldsymbol{d}) \geq 0$ pour tout $\boldsymbol{d} \in \mathbb{R}^n$.

Dans un contexte `DFO` où le gradient n'existe pas, la notion de cône hypertangent doit être introduite pour traiter des problèmes avec contraintes.

Définition 2.9 (Cône hypertangent). Un vecteur $\boldsymbol{v} \in \mathbb{R}^n$ est dit hypertangent à un ensemble $\Omega \subset \mathbb{R}^n$ en un point $\boldsymbol{x} \in \Omega$ si, et seulement si, il existe un scalaire $\epsilon > 0$ tel que :

$$y + tw \in \Omega \ \text{ pour tout } \ y \in \mathcal{B}(\boldsymbol{x}, \epsilon) \cap \Omega,\ w \in \mathcal{B}(\boldsymbol{v}, \epsilon) \text{ et } 0 < t < \epsilon\ .$$

L'ensemble des vecteurs hypertangents à Ω en $\boldsymbol{x}$ est appelé le cône hypertangent à Ω en $\boldsymbol{x}$ et est noté $T_{\Omega}^{H}(\boldsymbol{x})$.

Un point stationnaire au sens de la dérivée de Clarke dans le cas contraint est donné par la proposition suivante :

Proposition 2.3. Soit $\boldsymbol{x} \in \mathbb{R}^n$ et $f : \mathbb{R}^n \to \mathbb{R}$ Lipschitz continue en $\boldsymbol{x}$. Si $\boldsymbol{x}$ est un minimiseur local de f, alors $f_{\mathsf{C}}^{\circ}(\boldsymbol{x}; \boldsymbol{d}) \geq 0$ pour tout $\boldsymbol{d} \in T_{\Omega}^{H}(\boldsymbol{x})$.

Gestion des contraintes

En BBO les contraintes rencontrées diffèrent radicalement de celles en optimisation non linéaire. Elles peuvent être classées en neuf catégories suivant la taxonomie de Le Digabel et Wild [42] :

1. Connue ou cachéee : Une contrainte est dite connue si elle est explicitement spécifiée dans la formulation du problème (**P**). En revanche, elle est dite cachée si elle n'est pas fournie au solveur de manière explicite, de sorte à exclure implicitement une portion de l'espace des variables de décision, qui pourrait pourtant être admissible vis-à-vis des contraintes connues. Par exemple, une simulation pourrait inclure une opération arithmétique comme $\log(\boldsymbol{x})$, mais si le vecteur candidat possède des composantes négatives, alors la simulation échouera. Ce type de contrainte est fréquent en BBO ; dans [43, 44], la simulation échoue 60 % du temps pour des vecteurs d'entrées admissibles.
2. À priori ou simulée : Une contrainte est dite à priori si sa faisabilité peut être vérifiée sans nécessiter d'exécuter la boîte noire. Il s'agit souvent de contraintes de bornes agissant sur $\boldsymbol{x}$, de sortes à définir $\mathcal{X}$ comme un polytope dans $\mathbb{R}^n$. Les contraintes simulées sont spécifiques au contexte BBO.
3. Relaxable ou non relaxable : Les contraintes relaxables sont celles pour lesquelles l'évaluation de f et de c_j, $j \in \mathcal{J}$ retourne des valeurs utilisables pour tout $\boldsymbol{x} \in \mathcal{X}$, même lorsque $\boldsymbol{x} \notin \Omega$. À l'opposé, les contraintes non relaxables exigent $\boldsymbol{x} \in \Omega$, car toute évaluation effectuée en dehors de cette région produit des informations non valides. Ces contraintes sont souvent liées au respect de lois physiques garantissant la cohérence de la simulation.
4. Quantifiable ou non quantifiable : Une contrainte est dite quantifiable si elle fournit une mesure de son degré de faisabilité ou d'infaisabilité. Dans le cas contraire, si la contrainte renvoie une valeur binaire indiquant simplement si elle est satisfaite ou non, elle est non quantifiable.

MADS propose deux approches distinctes pour la gestion des contraintes d'inégalité : une approche minimaliste, i.e, (i) la barrière extrême (EB) qui rejette systématiquement tout point $\boldsymbol{x} \notin \Omega$, et une approche sophistiquée, i.e, (ii) la barrière progressive (PB) qui tolère tout point $\boldsymbol{x} \in \mathcal{X} \setminus \Omega$. Bien que ces deux approches soient assez différentes, elles ont pour point commun l'utilisation de la fonction de *violation de contraintes*.

Définition 2.10 (Fonction de violation de contraintes [3]). La fonction de violation de contraintes $h : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ est définie comme

$$\forall \boldsymbol{x} \in \mathbb{R}^n, \quad h(\boldsymbol{x}) \triangleq \begin{cases} \sum_{j \in \mathcal{J}} \left(\max\{c_j(\boldsymbol{x}), 0\}\right)^2 & \text{si } \boldsymbol{x} \in \mathcal{X}, \\ \infty & \text{sinon.} \end{cases}$$

Cette fonction non négative présente une valeur nulle si $\boldsymbol{x} \in \Omega$ et mesure le degré par lequel les contraintes relaxables sont violées, ceci est une information importante pour guider MADS vers une région faisable, comme il sera vu avec la PB.

Définition 2.11 (Fonction barrière extrême [3]). La fonction barrière extrême $f_\Omega : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ est définie comme

$$\forall \boldsymbol{x} \in \mathbb{R}^n, \quad f_\Omega(\boldsymbol{x}) \triangleq \begin{cases} f(\boldsymbol{x}), & \text{si } \boldsymbol{x} \in \Omega, \\ \infty & \text{sinon.} \end{cases}$$

Lorsque la barrière extrême (EB) est utilisée, le problème (**P**) est reformulé sous la forme $\min_{\boldsymbol{x} \in \mathcal{X}} f_\Omega(\boldsymbol{x})$. À ce stade, le point initial du processus d'optimisation $\boldsymbol{x}^0$ se doit d'appartenir à Ω. Autrement dit, l'intégralité des contraintes relaxables et non relaxables doit être satisfaite. Si ce n'est pas le cas, une stratégie en deux phases de l'EB est employée dans l'optique de trouver un point initial faisable. La première étape appelée phase de faisabilité consiste à résoudre le problème $\overline{\boldsymbol{x}} \in \arg\min_{\boldsymbol{x} \in \mathcal{X}} h(\boldsymbol{x})$ pour chercher un point $\overline{\boldsymbol{x}}$ tel que $h(\overline{\boldsymbol{x}}) = 0$. Si un tel point $\overline{\boldsymbol{x}}$ est trouvé, il est alors utilisé comme point initial dans la deuxième phase (phase d'optimisation) via $\boldsymbol{x}^0 \triangleq \overline{\boldsymbol{x}}$, pour résoudre $\min_{\boldsymbol{x} \in \mathcal{X}} f_\Omega(\boldsymbol{x})$. Le cas échéant, l'algorithme se termine en concluant qu'aucun point faisable n'a été trouvé dans Ω. Cette façon de traiter les contraintes est simple, mais n'utilise pas l'information sur les points non réalisables à partir de la seconde phase. Dans le contexte BBO où les évaluations sont coûteuses, une stratégie plus viable devrait utiliser l'information sur les points non réalisables admissibles (contraintes relaxables) dès que possible afin d'aider le solveur à s'orienter vers Ω.

L'approche PB comble ce besoin en introduisant une suite $(h^k_{\max})_{k \in \mathbb{N}}$ décroissante strictement positive de seuils de barrière et mise à jour à chaque itération k. Tout point évalué à toute itération k et tel que $h(\boldsymbol{x}) > h^k_{\max}$ est rejeté. Initialement, $h^0_{\max} \triangleq \infty$, et au fil des itérations, $\lim_{k \to \infty} h^k_{\max} = 0$, si au moins un point faisable est trouvé durant l'optimisation. Cette stratégie vise à approcher une solution faisable en explorant autour de deux solutions courantes à chaque itération k : $\boldsymbol{x}^k_{feas}$, le point faisable avec la plus petite valeur de l'objectif, et $\boldsymbol{x}^k_{inf}$, le point infaisable avec la plus petite valeur de l'objectif tel que $h(\boldsymbol{x}^k_{inf}) < h^k_{\max}$.

Si $f(\boldsymbol{x}_{inf}^k) < f(\boldsymbol{x}_{feas}^k)$, l'exploration locale autour du point $\boldsymbol{x}_{inf}^k$ peut s'avérer judicieuse afin de possiblement trouver de bonnes solutions faisables.

Afin de mettre à jour la valeur de $h_{\max}^{k+1}$ pour l'itération suivante, une comparaison par paire des points de l'itération k est nécessaire, ce qui requiert la définition de points dominés en optimisation sous contraintes.

Définition 2.12 (Points dominés en optimisation sous contrainte [3]).
- Un point réalisable $\boldsymbol{x} \in \Omega$ domine $\boldsymbol{y} \in \Omega$, noté $\boldsymbol{x} \prec_f \boldsymbol{y}$, si $f(\boldsymbol{x}) < f(\boldsymbol{y})$.
- Un point non réalisable $\boldsymbol{x} \in \mathcal{X} \setminus \Omega$ domine $\boldsymbol{y} \in \mathcal{X} \setminus \Omega$, noté $\boldsymbol{x} \prec_h \boldsymbol{y}$, si $f(\boldsymbol{x}) \leq f(\boldsymbol{y})$ et $h(\boldsymbol{x}) \leq h(\boldsymbol{y})$ avec au moins une égalité stricte.
- Un point $\boldsymbol{x}$ d'un ensemble quelconque $\mathcal{S} \subset \mathcal{X}$ est non-dominé s'il n'existe pas de point $\boldsymbol{y} \in \mathcal{S}$ qui domine $\boldsymbol{x}$.

La valeur $h_{\max}^{k+1}$ est alors mise à jour suivant :

1. Si à l'itération k, un point $\boldsymbol{y} \in \Omega$ pour lequel $\boldsymbol{y} \prec_f \boldsymbol{x}_{feas}^k$ ou un point $\boldsymbol{y} \in \mathcal{X} \setminus \Omega$ pour lequel $\boldsymbol{y} \prec_h \boldsymbol{x}_{inf}^k$ est trouvé, alors $h_{\max}^{k+1} \triangleq h(\boldsymbol{x}_{inf}^k)$.
2. Si à l'itération k, un point $\boldsymbol{y} \in \mathcal{X} \setminus \Omega$ pour lequel $0 < h(\boldsymbol{y}) < h(\boldsymbol{x}_{inf}^k)$ est trouvé, alors $h_{\max}^{k+1} \triangleq \max\{h(\boldsymbol{v}) \mid h(\boldsymbol{v}) < h(\boldsymbol{x}_{inf}^k), \boldsymbol{v} \in \mathcal{V}^k\}$, où $\mathcal{V}^k$ est l'ensemble de tous les points d'essai à la fin de l'itération k, pour lesquels f et h ont été calculés.
3. Si à l'itération k, aucun point $\boldsymbol{y}$ n'est trouvé comme dominant par rapport aux points de $\mathcal{V}^k$, alors $h_{\max}^{k+1} \triangleq h(\boldsymbol{x}_{inf}^k)$.

Par ailleurs, la PB relaxe la contrainte $\boldsymbol{x} \in \Omega$ imposée par la EB en imposant que $\boldsymbol{x} \in \mathcal{X}$, ce qui est moins contraignant à satisfaire.

Aucune approche spécifique n'est dédiée à la gestion des contraintes d'égalité avec `MADS`, il s'agit actuellement d'un sujet ouvert en `BBO`.

CHAPITRE 3 VARIANTES PARALLÈLES DE L'ALGORITHME DE RECHERCHE DIRECTE PAR TREILLIS ADAPTATIF

Cette section présente les quatre méthodes parallèles basées sur la Version `3.9.1` de NOMAD [45]. Celles-ci sont en outre :

- `pMADS-S` : Parallel `MADS` Synchronous (3.1.2) ;
- `pMADS-A` : Parallel `MADS` Asynchronous (3.1.1) ;
- `COOP-MADS` : Cooperative `MADS` (3.2) ;
- `PSD-MADS` : Parallel Decomposition `MADS` (3.3) ;

Alors que `PSD-MADS` a été l'objet de [46], les trois autres méthodes n'ont jamais été décrites ni évaluées numériquement dans la littérature, ce qui motive la rédaction de cette section.

Ces méthodes utilisent le modèle de programmation à mémoire distribuée via la bibliothèque `Message Passing Interface (MPI)` [47] et suivent le paradigme *master-worker*. Les propriétés de convergence de `MADS` s'appliquent à toutes les versions parallèles.

En suivant les stratégies parallèles décrites dans la Section 2, `pMADS-A` et `pMADS-S` sont classées comme parallélisme de données, tandis que `PSD-MADS` est classée comme parallélisme par décomposition de domaine. Enfin, `COOP-MADS` est classée comme parallélisme par recherche multiple.

3.1 `pMADS`

Dans NOMAD, la file d'évaluations, qui est une structure de données linéaire, contient les points à évaluer pour une étape donnée de l'itération k. Ainsi, lorsqu'une étape propose plusieurs points, il y a une opportunité directe d'exploiter le parallélisme des données. Les premières versions parallèles de `MADS` nommées `pMADS` implémentent cette stratégie. Le *master* est alors responsable de distribuer les points d'évaluations aux *workers*, qui sont ensuite chargés de les évaluer. Cela est résumé à la figure 3.1. Deux versions de `pMADS` sont disponibles : (i) la variante asynchrone (`pMADS-A`) et (ii) la variante synchrone (`pMADS-S`).

3.1.1 `pMADS-A`

Cette version est dite asynchrone en raison, que le *master* est en mesure de mettre fin à une itération k dès qu'un point succès a été obtenu par un *worker* et d'ensuite passer à l'itération suivante $k+1$ en générant de nouveaux points d'évaluations, même si des évaluations sont

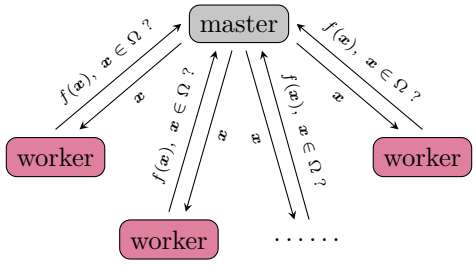

FIGURE 3.1 Organisation de la communication du paradigme *master-worker*.

toujours en cours chez d'autres *workers* de l'itération k. De plus, si une évaluation réalisée lors d'une itération précédente $k-1$ trouve une meilleure solution $\boldsymbol{x}^{k-1}$, que la solution actuelle $\boldsymbol{x}^k$ de l'itération courante k, l'algorithme a la capacité de faire un retour sur $\boldsymbol{x}^{k-1}$, par une projection sur $\mathcal{M}^{k-1}$ afin d'hériter de la convergence de `MADS`.

Il n'y a donc aucune barrière de synchronisation entre les processus `MPI`, ce qui rend ce mode particulièrement efficace pour les boîtes noires à temps hétérogène, car aucun processus `MPI` n'attend que le plus lent termine l'évaluation de sa solution. Il est impératif d'éviter l'inactivité des ressources de calcul, car cela entraîne de mauvaises performances et des goulots d'étranglement dans un contexte de programmation parallèle. Cette assertion est approfondie à la section 5, où un résultat numérique vient étayer cette affirmation.

Le principal inconvénient de ce mode est le comportement non déterministe de l'algorithme, ce qui limite la reproductibilité des résultats. `pMADS-A` est le mode par défaut utilisé dans NOMAD lorsque le parallélisme est activé. De par sa grande complexité algorithmique, on ne formule pas d'algorithme pour `pMADS-A`.

3.1.2 `pMADS-S`

La version synchrone de `pMADS` conserve le même objectif que la version asynchrone, à savoir paralléliser la file d'évaluation, mais avec l'introduction d'une barrière de synchronisation, ce qui est le prix à payer pour obtenir une version déterministe. De cette façon, chaque *worker* évalue un point, puis attend que les autres *workers* aient terminé leur évaluation. L'algorithme 2 détaille `pMADS-S` pour le cas non contraint.

3.2 `COOP-MADS`

Pour accélérer l'exploration de $\mathcal{X}$ et éviter la convergence rapide de `MADS` vers un optimum local, plusieurs chemins peuvent être générés en exécutant différentes instances de `MADS` en parallèle. Cela peut être particulièrement bénéfique pour les problèmes multimodaux, où

Algorithme 2 : pMADS-S

Suivant $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$, $\boldsymbol{x}^0 \in \mathbb{R}^n$, c CPU
Initialisation
  $\mathbf{D} = \mathbf{GZ}$, $\Delta^0 > 0$, $\tau \in (0,1) \cap \mathbb{Q}$
for $k = 0, 1, 2, \ldots$ **do**
  $\delta^k = \min\{\Delta^k, (\Delta^k)^2\}$
  SEARCH (optionnel)
    Construire un ensemble fini de points $\mathcal{S}^k \subset \mathcal{M}^k$
    if $\exists \boldsymbol{x} \in \mathcal{S}^k \mid f(\boldsymbol{x}) < f(\boldsymbol{x}^k)$ **then**
      $\boldsymbol{x}^{k+1} \leftarrow \boldsymbol{x}$, $\Delta^{k+1} \leftarrow \tau^{-1}\Delta^k$, aller à **Terminaison**
    end
  POLL
    Choisir $\mathbb{D}_\Delta^k$ pour construire $\mathcal{P}^k$
    while $\forall \boldsymbol{x} \in \mathcal{P}^k$ n'ont pas été évalué **do**
      $\mathcal{W} \triangleq \{\boldsymbol{x}_{w_i} \mid i \in \{1 \ldots c\}\} \subset \mathcal{P}^k$
      Évaluations parallèles de $f(\boldsymbol{x})$ $\forall \boldsymbol{x} \in \mathcal{W}$
      $\boldsymbol{x}^*_{w_i} \in \arg\min_{x \in \mathcal{W}} f(\boldsymbol{x})$
      if $f(\boldsymbol{x}^*_{w_i}) < f(\boldsymbol{x}^k)$ **then**
        $\boldsymbol{x}^{k+1} \leftarrow \boldsymbol{x}^*_{w_i}$, $\Delta^{k+1} \leftarrow \tau^{-1}\Delta^k$, aller à **Terminaison**
      end
    end
    $\boldsymbol{x}^{k+1} \leftarrow \boldsymbol{x}^k$, $\Delta^{k+1} \leftarrow \tau\Delta^k$
  Terminaison
    if Critère arrêt **then** Terminer MADS
end

l'objectif est de converger vers un optimum global. Chaque instance de MADS est initialisée avec une graine aléatoire unique, ce qui entraîne la sélection de directions variées à partir d'OrthoMADS [40]. Cela donne lieu à des comportements de recherche distincts.

Ces instances fonctionnent indépendamment et sans mécanismes de synchronisation entre elles. Pour optimiser l'utilisation des évaluations, une distribution dynamique du budget global d'évaluation est mise en place entre les instances. Il est à noter qu'une allocation budgétaire statique, consistant à répartir les évaluations de manière égale entre les instances, entraîne souvent une sous-utilisation des ressources en raison des taux de convergence variables des instances individuelles.

Dans COOP-MADS, le seul échange d'informations entre les instances de MADS se fait par le biais d'une cache partagée, qui fournit un historique des évaluations effectuées. Avant de procéder à l'évaluation d'un point, une instance effectue une recherche dans la cache pour déterminer si le point a déjà été évalué ou est actuellement en cours d'évaluation, évitant

ainsi les évaluations redondantes. De plus, une option `CACHE_SEARCH` permet à une instance de comparer son point courant, au meilleur point de la cache et de l'échanger advenant le cas que celui de la cache offre une meilleure solution. Une étude numérique sera menée à savoir si cette option offre un avantage ou non, au chapitre 5. On décrit la structure algorithmique de `COOP-MADS` via l'algorithme 3 pour le cas non contraint.

Algorithme 3 : `COOP-MADS`

Suivant $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$, $\boldsymbol{x}^0 \in \mathbb{R}^n$, c `CPU`
Initialisation
  $\mathbf{D} = \mathbf{GZ}$, $\Delta^0 > 0$, $\tau \in (0,1) \cap \mathbb{Q}$
  Assigner une graine aléatoire différente à chaque c.
parallel for $i \in 1, \ldots, c$ **do**
  for $k = 0, 1, 2, \ldots$ **do**
    $\delta_i^k = \min\{\Delta_i^k, (\Delta_i^k)^2\}$
    `SEARCH` (optionnel)
      Construire un ensemble fini de points $\mathcal{S}_i^k \subset \mathcal{M}_i^k$
      if $\exists \boldsymbol{x} \in \mathcal{S}_i^k \mid f(\boldsymbol{x}) < f(\boldsymbol{x}_i^k)$ **then**
        $\boldsymbol{x}_i^{k+1} \leftarrow \boldsymbol{x}$, $\Delta_i^{k+1} \leftarrow \tau^{-1}\Delta_i^k$, aller à **Terminaison**
      end
    `POLL`
      Choisir $\mathbb{D}_{\Delta,i}^k$ pour construire $\mathcal{P}_i^k$
      if $\exists \boldsymbol{x} \in \mathcal{P}_i^k \mid f(\boldsymbol{x}) < f(\boldsymbol{x}_i^k)$ **then**
        $\boldsymbol{x}^{k+1} \leftarrow \boldsymbol{x}$, $\Delta_i^{k+1} \leftarrow \tau^{-1}\Delta_i^k$
      else
        $\boldsymbol{x}_i^{k+1} \leftarrow \boldsymbol{x}_i^k$, $\Delta_i^{k+1} \leftarrow \tau\Delta_i^k$
      end
    Terminaison
      if Critère arrêt **then** Terminer l'instance i `MADS`
  end
end

3.3 `PSD-MADS`

La résolution de problèmes `BBO` à grande dimension est une tâche computationnellement exigeante, principalement en raison du fléau de la dimensionnalité. `PSD-MADS` [46] se distingue des trois autres versions parallèles de `MADS`, car il a été spécifiquement conçu pour traiter des problèmes `BBO` à grande échelle. Il exploite une approche par décomposition de domaine parallèle.

Le concept central de `PSD-MADS` repose sur le *master* qui génère pour chaque *worker* p un sous-problème d'optimisation noté $\mathbf{P}_p$, en sélectionnant aléatoirement un sous-espace de variables

N_p à partir de l'espace de variables initial $\mathcal{X}$, c'est-à-dire $N_p \subset \mathcal{X}$. Chaque sous-problème est ensuite distribué à son *worker* par le *manager*. Chaque *worker* applique `MADS` sur son sous-problème assigné afin d'améliorer la solution actuelle commune notée $\boldsymbol{x}^*$. Une fois sa tâche terminée, un *worker* reçoit un nouveau sous-problème $\mathbf{P}_p$ du *master* indépendamment du statut de terminaison des autres *workers*, ce qui permet un fonctionnement asynchrone de l'algorithme.

Quatre rôles sont introduits pour assurer le fonctionnement de `PSD-MADS` : le *manager*, le *pollster*, les *workers* et le *cache server*. La figure 3.2 montre l'organisation et l'interaction des différents rôles au sein de l'algorithme.

Le *pollster* résout le problème original (**P**) en utilisant l'espace des variables complet $\mathcal{X}$ avec un algorithme `MADS` à direction de `POLL` unique, plutôt qu'avec les directions de `POLL` habituelles $2n$ ou $n+1$. Le *pollster* assure la convergence de `PSD-MADS`. Comme mentionné précedemment, le *manager* est responsable de la génération des sous-problèmes $\mathbf{P}_p$ mais aussi d'ajuster la taille du treillis principale et de mettre à jour la solution actuelle commune. Le *cache server* fournit un historique des point évalués. Enfin, le *worker* est une instance de `MADS` qui résout un sous problème $\mathbf{P}_p$.

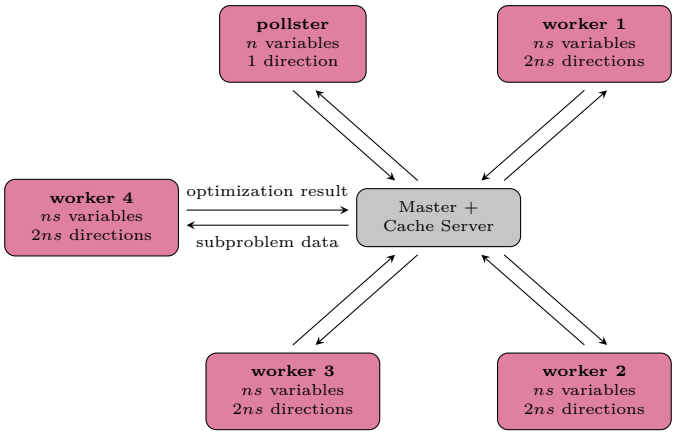

FIGURE 3.2 Organisation de la communication du paradigme *master-worker* pour `PSD-MADS`, où pour chaque *worker* p, le message associé au résultat d'optimisation (*optimization result*) contient la taille finale du treillis δ^p_{fin} et la solution finale $\boldsymbol{x}_p$, alors que le message associé aux données d'un sous-problème (*subproblem data*) contient la solution de départ $\boldsymbol{x}^0$, l'ensemble des variables N_p et la taille initiale et minimale du treillis soient respectivement δ^p_0 et δ^p_{min}.

CHAPITRE 4 ÉTUDE DE LA RÉPARTITION COMPUTATIONNELLE

On présente dans ce chapitre la modélisation du parallélisme de l'algorithme MADS afin de répondre à la question de recherche formulée à la section 1.1. On introduit également les hypothèses qui devront être vérifiées numériquement au chapitre 5.

La modélisation du parallélisme peut être scindée en deux axes, à savoir : (i) l'impact sur le temps de résolution (section 4.1) et (ii) l'impact sur la solution obtenue par les modèles quadratiques (section 4.2). Il existe donc un compromis, car idéalement, on souhaite disposer d'un algorithme qui minimise le temps de calcul tout en fournissant la meilleure solution possible.

Dans cette section, on propose d'utiliser une approche par modèle similaire à celle présentée dans la section 2.1.3, sur laquelle les utilisateurs se basent pour obtenir une borne supérieure de l'accélération parallèle (5.1). Cette approche sera adaptée à MADS, afin de juger si l'implémentation d'une stratégie dynamique en répartition des ressources de calcul est encline à apporter des gains d'accélération parallèle suffisants, motivant ou non une implémentation logicielle dans NOMAD.

4.1 Modélisation du temps de résolution

L'étape de POLL génère au maximum $2n$ points d'évaluation et au minimum $n+1$ points d'évaluation, selon le théorème 2.2. Étant donné que le nombre de cœurs de calcul dans un processeur est toujours un nombre pair, on préconise l'approche à base maximale.

Puisque la file d'évaluations contient $2n$ points, le nombre maximum de cœurs de calcul alloué au solveur doit être un diviseur de $2n$, afin d'éviter la situation où il y aurait moins de points à évaluer que de cœurs de calcul disponible, ce qui entraînerait une utilisation partielle de ces derniers.

À des fins de modélisation, on introduit les variables suivantes :

- Le nombre de cœurs de calcul alloué au solveur d'optimisation noté p_{solv} ;
- Le nombre de cœurs de calcul alloué à la boîte noire noté p_{bb} ;
- Le temps de calcul d'une évaluation de la boîte noire en séquentiel noté t_{bb} ;
- L'efficacité parallèle de la boîte noire notée $E(\eta, p_{bb})$ (2.2) ;
- Le nombre de points d'évaluation de la POLL à l'itération k, noté $|\mathcal{P}^k|$;
- Le coût en temps occasionné par la parallélisation du solveur d'optimisation noté $O(p_{solv})$.

On considère alors un ensemble d'algorithmes $\mathscr{A}$ qui sont des variantes parallèles de l'algorithme pMADS différenciées par leur choix des valeurs de p_{solv} et p_{bb}. Une contrainte est imposée afin de limiter le nombre d'algorithmes dans $\mathscr{A}$: (C1) chaque algorithme $a \in \mathscr{A}$ utilise le même nombre de cœurs de calcul [1].

Le temps pour une itération k de la POLL est estimé par l'équation suivante si les $2n$ points d'évaluation sont calculés, soit $|\mathcal{P}^k| = 2n$:

$$t^k_{\texttt{POLL}} \approx \left(\frac{|\mathcal{P}^k|}{p_{solv}}\right)\left(\underbrace{\frac{t_{bb}/p_{bb}}{E(\eta, p_{bb})}}_{R_{bb}} + O(p_{solv})\right). \tag{4.1}$$

Dans le contexte BBO, où une évaluation est coûteuse en temps de calcul, il est raisonnable de négliger $O(p_{solv})$, car le ratio R_{bb} qui représente le temps de calcul de la boîte noire parallélisée sur p_{bb} unités de calcul avec une efficacité de $E(\eta, p_{bb})$ est bien plus grand que $O(p_{solv})$, c'est-à-dire $R_{bb} \gg O(p_{solv})$.

L'équation 4.1 est alors simplifiée et réécrite comme :

$$t^k_{\texttt{POLL}} \approx \left(\frac{|\mathcal{P}^k|}{p_{solv}}\right)\left(\frac{t_{bb}/p_{bb}}{E(\eta, p_{bb})}\right) \tag{4.2}$$

Étant donné que les $2n$ évaluations sont effectuées, tous les algorithmes $a \in \mathscr{A}$ trouvent une solution $\boldsymbol{x}^*$ identique. En revanche, $t^k_{\texttt{POLL}}$ varie selon chaque algorithme, car la contrainte (C1) est imposée et donc ce qui diffère d'un algorithme à l'autre est uniquement la valeur $E(\eta, p_{bb})$ qui impacte directement le temps de calcul comme discuté dans la section 2.1.3.

Pour déterminer l'algorithme a minimisant le temps de calcul, une étude de mise à l'échelle du code informatique de la boîte noire doit être effectuée pour mesurer le ratio R_{bb} de chaque algorithme.

En pratique, l'évaluation d'une POLL complète à chaque itération k n'est pas une stratégie préconisée, car l'évaluation de la boîte noire requiert un temps de calcul élevé, en plus d'un nombre important de ressources de calcul. Pour réduire le nombre d'évaluations à chaque itération k et limiter les ressources de calcul impliquées, la stratégie opportuniste est habituellement adoptée.

Définition 4.1 (Stratégie opportuniste). La stratégie opportuniste d'un algorithme DSM à l'itération k désigne l'arrêt anticipé de l'étape de POLL ou de SEARCH de l'itération k et

1. $(p_{solv} \cdot p_{bb})$ est constant pour tous les algorithmes $a \in \mathscr{A}$.

ce dès la première obtention d'un point satisfaisant le critère de succès de l'algorithme suivant l'évaluation d'un bloc de points de taille b.

Pour mettre en place une stratégie opportuniste efficace, c'est-à-dire une stratégie qui permet de minimiser le nombre d'évaluations effectuées à chaque itération k, il est nécessaire d'ordonner les points de la file d'évaluation de manière à placer les points les plus enclins à satisfaire le critère de succès de l'algorithme, en tête de file, et les moins enclins à la fin. Ce processus de triage fait référence à la notion d'ordonnancement.

Dans la thèse [48], une étude numérique des stratégies d'ordonnancement proposées par le logiciel NOMAD est réalisée, et il est montré que l'ordonnancement en utilisant les modèles quadratiques a le plus grand impact sur la qualité des solutions. Ce point sera abordé plus en détail dans la section suivante, qui considère le cas parallèle, étant donné que [48] se concentre sur le cas séquentiel.

Lorsque la stratégie opportuniste est appliquée à `MADS` en programmation séquentielle, elle se manifeste au niveau d'un bloc de taille $b = 1$ soit sur chaque point individuel de la file d'évaluations. Ainsi, un minimum d'une évaluation peut suffire pour terminer l'itération k en cours, si le critère de décroissance simple pour `MADS` est satisfait pour le point évalué. En revanche, dans un environnement parallèle, les évaluations sont distribuées simultanément à plusieurs unités de calcul. Le nombre minimum d'évaluations effectuées avant l'interruption anticipée d'une itération k correspond au nombre d'unités de calcul utilisées par le solveur, soit p_{solv}. Par conséquent, le budget d'évaluations est consommé plus rapidement qu'en séquentiel, ce qui explique pourquoi un budget de temps de calcul est plus adéquat dans un environnement parallèle. L'opportunisme en parallèle qui se manifeste sur un bloc de taille $b = p_{solv}$ occasionne alors une perte d'efficacité de la stratégie à mesure que la valeur de p_{solv} est grande.

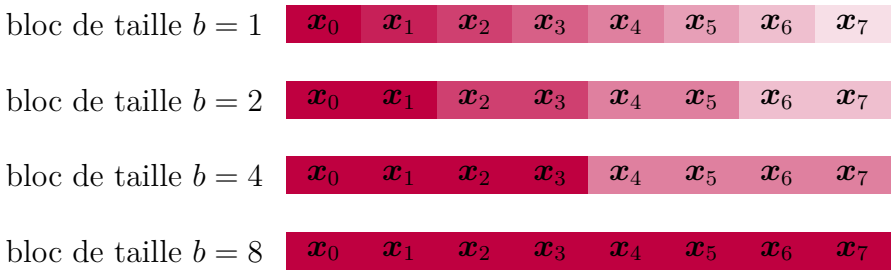

FIGURE 4.1 Stratégie opportuniste pour différentes tailles de bloc d'évaluations.

En somme, grâce à la stratégie opportuniste, une économie d'évaluations de la boîte noire peut être réalisée à chaque itération k, ce qui occasionne alors une économie des ressources

de calcul allouées au solveur.

Pour intégrer cette stratégie dans notre modélisation, la variable i_{ord} est introduite et indique l'indice du dernier point évalué par un algorithme séquentiel utilisant la stratégie opportuniste. Pour une itération où un point satisfaisant le critère de décroissance minimale de `MADS` est trouvé, $i_{ord} \leq 2n$ et le cas échéant pour lequel l'itération est considérée comme un échec, $i_{ord} = 2n$.

De plus, on doit également introduire une nouvelle fonction :

$$\forall (i_{ord}, p_{solv}) \in \mathbb{N}^* \times \mathbb{N}^*, \quad g(i_{ord}, p_{solv}) \triangleq \begin{cases} 1, & \text{si } p_{solv} > i_{ord}, \\ \left\lceil \frac{i_{ord}}{p_{solv}} \right\rceil, & \text{sinon.} \end{cases}$$

Cette fonction pénalise le surplus d'évaluations fait par un algorithme, selon le nombre de points minimal requis à être évalué pour satisfaire la stratégie opportuniste.

La fonction $\lceil \cdot \rceil$ arrondie à l'entier supérieur, puisqu'il n'est pas possible de savoir préalablement si p_{solv} est un diviseur de i_{ord}, lorsque la stratégie opportuniste est appliquée.

Le temps d'une itération k de la `POLL` est alors exprimé par l'équation suivante pour une valeur donnée de i^k_{ord} :

$$t^k_{\texttt{POLL}} \approx g(i^k_{ord}, p_{solv}) \left(\frac{t_{bb}/p_{bb}}{E(\eta, p_{bb})} \right). \tag{4.3}$$

Avec la présence de la stratégie opportuniste, on cherche à avoir $i_{ord} = p_{solv}$, de sorte à maximiser le nombre d'unités de calcul disponibles pour paralléliser la boîte noire. La situation où $i_{ord} < p_{solv}$ est considérée comme sous-optimale en termes de temps de calcul, puisque $(p_{solv} - i_{ord})p_{bb}$ unités de calcul auraient pu être allouées à la boîte noire.

En raison du comportement de la fonction g, il semble judicieux d'adopter une stratégie dite *conservatrice*, avec $p_{solv} = 1$. Cependant, à mesure que le ratio i_{ord}/p_{solv} augmente, cette approche devient discutable, car elle ne tire pas parti du potentiel de parallélisation des évaluations. En particulier, le cas limite, où le ratio $i_{ord}/p_{solv} = 2n$, se produit lorsque `MADS` converge en un point stationnaire. En vertu de la proposition 2.2, pour converger, `MADS` requiert de réaliser un nombre consécutif d'itérations k au cours desquelles une `POLL` complète est calculée, afin de remplir la sphère unitaire de directions. Dans cette situation, la stratégie de `POLL` complète, avec $p_{solv} = 2n$, s'avère optimale en termes de temps de calcul, tandis que la stratégie conservatrice est sous-optimale.

L'objectif des analyses suivantes est de quantifier l'écart de temps de calcul entre ces deux stratégies pour ce cas limite. Cela permettra de déterminer si adopter une répartition des

ressources de calcul différente peut entraîner une réduction significative du temps de calcul. Si l'on prend la situation en exemple avec deux algorithmes pour lesquels $i^k_{ord} = 2n$:

1. Algorithme $a \in \mathscr{A}$ avec $p_{solv} = 2n$, $p_{bb} = 1$;
2. Algorithme $b \in \mathscr{A}$ avec $p_{solv} = 1$, $p_{bb} = 2n$.

Pour l'algorithme a le temps d'une itération k de la `POLL` avec les paramètres indiqués sera de

$$t^{k,a}_{\texttt{POLL}} \approx \underbrace{g(2n,2n)}_{=1}\left(\frac{t_{bb}/p_{bb}}{E(\eta,p_{bb})}\right) = \left(\frac{t_{bb}/1}{\underbrace{E(\eta,1)}_{=1}}\right) = t_{bb}.$$

Pour l'algorithme b le temps d'une itération k de la `POLL` avec les paramètres indiqués sera de

$$t^{k,b}_{\texttt{POLL}} \approx \underbrace{g(2n,1)}_{=2n}\left(\frac{t_{bb}/p_{bb}}{E(\eta,p_{bb})}\right) = \frac{2n(t_{bb}/2n)}{E(\eta,2n)} = \frac{t_{bb}}{E(\eta,2n)}.$$

La différence de temps entre $t^{k,a}_{\texttt{POLL}}$ et $t^{k,b}_{\texttt{POLL}}$ est sujette à la valeur de $E_{bb}(2n,\eta)$ de l'algorithme b. En pratique, en raison de considérations énergétiques, une borne minimale est de 80 % comme expliqué dans la thèse [49]. Dès lors, il est possible de conclure que $t^{k,b}_{\texttt{POLL}} \leq \frac{10}{8} t^{k,a}_{\texttt{POLL}}$. Pour ce cas limite, il est notable de constater que même en optant pour la stratégie conservatrice, celle-ci présente un résultat de temps de calcul compétitif à celui de la stratégie optimale. En pratique, l'écart de temps de calcul entre $t^{k,a}_{\texttt{POLL}}$ et $t^{k,b}_{\texttt{POLL}}$ est revu à la baisse, car $p_{bb} \gg p_{solv}$. En effet, l'ordre de grandeur des ressources de calcul exploitables par la boîte noire est beaucoup plus grand que celui du solveur. Cela est notamment dû au fait que, généralement, les problèmes (**P**) impliquant des boîtes noires massivement parallèles contiennent rarement plus de 10 variables, car il s'agit de problèmes géométriques.

Par exemple, dans la thèse [50] qui se concentre sur la mécanique des fluides numérique appliquée à l'aérospatiale, les problèmes (**P**) formulés comportent au plus quatre variables, donc $p_{solv} \leq 8$. Alors que la mise à l'échelle d'un code de calcul en mécanique des fluides numérique peut facilement dépasser 1000 cœurs de calcul, donc $p_{bb} \geq 1000$. Afin d'occuper un nombre de cœurs de calcul conséquent, la stratégie de `POLL` complète ne peut uniquement se limiter à la parallélisation des points d'évaluations. Elle doit également paralléliser la boîte noire, ce qui implique que $p_{bb} > 1$ et donc $E(\eta, p_{bb}) < 1$. Cela diffère du cas précédemment traité avec $p_{bb} = 1$, où l'efficacité était maximale, c.-à-d., $E(\eta, 1) = 1$.

Jusqu'à présent dans la modélisation, on imposait l'hypothèse que le temps d'évaluation de la boîte noire était similaire pour tout point d'évaluation. En pratique, le temps de calcul présente souvent une nature hétérogène selon le point d'évaluation comme illustré à la figure 1.1 et discuté dans [51].

Un second cas propice à l'adoption de la stratégie conservatrice est lorsque la boîte noire présente un comportement à temps hétérogène, pour un algorithme parallèle synchrone comme l'implémentation de `pMADS` dans NOMAD 4.

À l'itération k, pour un bloc d'évaluations de taille $b^k = p^k_{solv}$ le temps maximal d'une évaluation de la boîte noire est défini comme

$$t^k_{\max} \triangleq \max\{t_1, \cdots, t_{b^k}\}.$$

Le temps d'attente d'une unité de calcul en pause (*idling time*) à l'itération k dans un bloc de taille $b^k = p^k_{solv}$ est défini comme

$$\Delta t^k_i \triangleq (t^k_{\max} - t^k_i) \ \forall i \in \{1, \cdots, b^k\}.$$

L'accélération parallèle (2.1) à l'itération k pour un bloc d'évaluations de taille $b^k = p^k_{solv}$ est alors donné par la formule suivante

$$S^k(\eta, b^k) = \frac{\sum_{i=1}^{b^k} t^k_i}{t^k_{\max}}.$$

On introduit des constantes $\beta_i \in]0, 1[\ \forall i \in \left\{1, \ldots, b^k\right\}$ représentant des pondérations de $t^k_{\max}$,

$$S^k(\eta, b^k) = \frac{\left(\sum_{i=1}^{b^k-1} \beta_i + 1\right) t^k_{\max}}{t^k_{\max}} = \sum_{i=1}^{b^k-1} \beta_i + 1.$$

Les intervalles de (2.1) et (2.2) sont alors, $S^k(\eta, b^k) \in \left]1, b^k\right[$ et $E^k(\eta, b^k) \in \left]\frac{1}{b^k}, 1\right[$, lorsque le solveur est parallélisé avec p^k_{solv} unités de calcul.

Pour cette même situation si on prend la stratégie conservatrice avec $b^k = p^k_{solv} = 1$, $p^k_{bb} > 1$ et $E^k(\eta, p^k_{bb}) = 80$ %, l'accélération parallèle obtenue est

$$S^k(\eta, p_{bb}) = \frac{\left(\sum_{i=1}^{b^k-1} \beta_i + 1\right) t^k_{\max}}{\left(\sum_{i=1}^{b^k-1} \beta_i + 1\right) t^k_{\max}/0,8 p^k_{bb}} = 0,8 p^k_{bb}.$$

Si la stratégie de parallélisation du solveur vise à obtenir une accélération similaire à celle de la stratégie conservatrice, alors la condition suivante sur les constantes $\beta_i \ \forall i \in \left\{1, \ldots, b^k\right\}$

$$S^k(\eta, b^k) = S^k(\eta, p^k_{bb}) \iff \sum_{i=1}^{b^k-1} \beta_i + 1 = 0,8 p^k_{bb} \iff \sum_{i=1}^{b^k-1} \beta_i = 0,8 p^k_{bb} - 1,$$

doit être satisfaite. Étant donné que $\sum_{i=1}^{b^k-1} \beta_i = p_{solv}^k - 1$ et que $p_{solv}^k = p_{bb}^k$ suivant la contrainte (C1) pour les deux stratégies. On remarque qu'à mesure que p_{solv} augmente à mesure que les valeurs moyennes des pondérations $\beta_i \; \forall i \in \left\{1, \ldots, b^k\right\}$ devraient converger vers une constante proche de $\lim_{p_{solv}^k \to \infty}(0,8 p_{solv}^k - 1)/(p_{solv}^k - 1) = 0,8$.

Ce qui signifie que le temps de calcul moyen doit être de 0,8 $t_{\max}^k$ pour les $b^k - 1$ évaluations. Cette condition peut être difficile à satisfaire en fonction de la distribution des temps d'exécution de la boîte noire. Par exemple, dans le cas du code solar présenté dans le chapitre suivant, les problèmes (**P**) sont particulièrement hétérogènes en termes de temps de calcul, une telle condition ne serait pas toujours respectée à chaque itération k.

Enfin, un scénario particulièrement problématique survient lorsque dans un bloc d'évaluations exécutées en parallèle, l'une des évaluations de la boîte noire échoue rapidement. Dans ce cas, des ressources de calcul sont gaspillées, car elles restent inactives pendant que d'autres évaluations continuent de s'exécuter. Ce cas ne survient pas avec la stratégie conservatrice.

En somme, il ne semble pas y avoir un apport considérable en termes de temps de calcul justifiant le déploiement d'une stratégie dynamique, selon notre modélisation. Du point de vue temps de calcul, on recommande la stratégie conservatrice, car elle nous permet de ne jamais être pénalisée par un surplus d'évaluation dans l'utilisation de la stratégie opportuniste, en plus d'être adaptée aux boîtes noires à temps hétérogène, tant pour les algorithmes parallèles synchrones qu'asynchrones ce qui assure de bonnes performances en parallèle.

4.2 Modèle quadratique

Comme mentionné précédemment, les modèles quadratiques ont un rôle clé sur l'efficacité de la stratégie opportuniste, car c'est via les modèles quadratiques que les points d'évaluations sont ordonnés dans la file d'évaluation à chaque itération k. Dans un idéal, on souhaite avoir les points d'évaluations ordonnés le plus précisément, de sortes à minimiser le nombre d'évaluations requis à chaque itération k, ce qui occasionne alors une diminution du temps de calcul. Une telle précision des modèles quadratiques requiert alors que la surface de réponse du modèle soit en adéquation avec l'espace des solutions approximé localement par ceux-ci. Le degré de parallélisme du solveur (p_{solv}) est alors un potentiel levier sur la qualité du modèle, car le parallélisme a un impact sur l'intensification de l'étape de `POLL`. Plus concrètement, soit l'algorithme a avec $p_{solv} = \alpha$, tel que $\alpha \in \mathbb{N}^*$ et l'algorithme b avec $p_{solv} = \beta$, tel que $\beta \in \{\lambda\alpha \mid \lambda \in \mathbb{N}^*\}$, alors pour un ensemble $\mathcal{P}^k$ identique, à une itération k donnée, la relation suivante est établie :

$$\mathcal{C}_a^k \triangleq \left\{\boldsymbol{x} \in \mathcal{P}^k \mid \boldsymbol{x} \text{ est évalué}\right\} \subseteq \mathcal{C}_b^k \triangleq \left\{\boldsymbol{x} \in \mathcal{P}^k \mid \boldsymbol{x} \text{ est évalué}\right\}.$$

L'inclusion survient uniquement dans le cas où une POLL complète est requise soient l'évaluation des $n+1$ ou $2n$ points.

Lorsque MADS utilise un modèle quadratique pour ordonner les points de la file d'évaluation, il construit un modèle quadratique local à l'itération k autour du point courant $\boldsymbol{x}^k$, noté m^k. Pour procéder à la sélection des points de m^k, MADS établit une boule d'un certain rayon r, notée $\mathcal{B}(r, \boldsymbol{x}^k)$. Comme l'algorithme b est plus enclin à évaluer un plus grand nombre de points sur le treillis à chaque itération k que l'algorithme a, alors pour un nombre d'itérations $K > 1$ identique pour les deux algorithmes :

$$\left|\mathcal{B}(r, \boldsymbol{x}_a^K) \cap \{\mathcal{C}_a^k\}_{k=1}^{K-1}\right| \;\leq\; \left|(\mathcal{B}(r, \boldsymbol{x}_b^K) \cap \{\mathcal{C}_b^k\}_{k=1}^{K-1}\right|,$$

avec $r > 0$ et $\mathcal{B}(r, \boldsymbol{x}) \triangleq \{\boldsymbol{z} \mid \|\boldsymbol{z} - \boldsymbol{x}\|_\infty < r\}$.

Ainsi, le modèle m^k de l'algorithme b est possiblement plus enrichi que celui de a ce qui devrait lui conférer une possible meilleure surface de réponse et par conséquent de meilleures capacités prédictives. On souhaite vérifier cette hypothèse à la section des tests numériques, via la comparaison des performances de l'ordonnancement quadratique entre les différents algorithmes de l'ensemble $\mathscr{A}$.

Enfin, les remarques précédentes donnes donc lieu à un compromis, c.-à-d., accélérer le temps de calcul de la boîte noire ou enrichir le modèle quadratique. Ces deux options nous permettent de sauver du temps de calcul. À nouveau, ce compromis fait intervenir la manière dont les ressources de calcul sont réparties entre la boîte noire et le solveur d'optimisation.

CHAPITRE 5 RÉSULTATS NUMÉRIQUES

Dans cette section, on réalise des expériences numériques afin d'évaluer la performance des algorithmes (*benchmarking*) décrits précédemment. La méthodologie suivie repose sur les bonnes pratiques proposées dans [52], qui fournit un cadre rigoureux pour la comparaison des algorithmes d'optimisation numérique. La section 5.1 présente les ensembles de problèmes tests sur lesquels les performances des algorithmes seront mesurées. Ensuite, la section 5.2 détaille la manière dont les performances seront analysées et comparées dans un contexte de calcul parallèle. Finalement, la section 5.3 fournit les résultats numériques obtenus ainsi qu'une analyse de ceux-ci.

5.1 Problèmes tests

Un problème test (**P**) est considéré soit comme réaliste ou analytique. Les problèmes réalistes, souvent très coûteux en termes de temps de calcul sont généralement utilisés uniquement lors de la phase finale des tests d'un algorithme d'optimisation. De plus, ces simulations réalistes ne sont pas toujours accessibles publiquement, ce qui limite le nombre de cas d'utilisation. Pour effectuer des tests intensifs sur un algorithme d'optimisation, les problèmes analytiques qui présentent un faible coût en temps de calcul sont privilégiés. Ils intègrent des propriétés mathématiques similaires à celles des problèmes réalistes, cherchant ainsi à reproduire leur complexité. Cependant, les problèmes réalistes contiennent souvent des structures cachées (*hidden structure*, [3]) susceptibles de biaiser l'analyse comparative des performances d'algorithmes. Par exemple, lorsque toutes les solutions d'un ensemble de problèmes tests sont à l'origine et que les points départs $\boldsymbol{x}^0$ sont des vecteurs d'entiers. Il revient donc à l'utilisateur d'y porter une attention particulière.

Ces deux types de problèmes sont essentiels au cycle d'évaluation de la performance d'algorithmes en optimisation numérique.

5.1.1 Problèmes analytiques

Les trois ensembles de problèmes tests analytiques utilisés sont :

1. Une vaste collection de problèmes d'optimisation non lisses contraints et non contraints est proposée dans [53, 54]. Des problèmes pratiques et analytiques sont disponibles. Pour ces problèmes, f est supposée Lipschitz continue sur l'ensemble faisable $\Omega \subseteq \mathbb{R}^n$. Dans notre cas, on se focalise sur les problèmes analytiques à grande échelle avec des

contraintes d'inégalité, où les problèmes peuvent être formulés avec un nombre quelconque de variables. Au total, l'ensemble de problèmes test comporte de 80 instances. De par leur complexité, aucune solution exacte n'est disponible, seulement des estimations numériques pour la dimension $n = 1000$. Cet ensemble est désigné par `Napsu` dans le cadre des tests numériques.

2. La collection Moré-Wild [55] est un ensemble de problèmes tests non contraint largement adopté par la communauté scientifique pour l'évaluation des performances d'algorithmes `DFO` en petite dimension. Le nombre n de variables varie de 2 à 12. Un total de 53 problèmes où f est continue mais non différentiable sont extraits pour les tests numériques.
3. Une collection de 30 problèmes analytiques sous contrainte a été constituée. Cette collection contient des problèmes moins difficiles à résoudre que ceux de la collection `Napsu`. Les informations sur les problèmes de test sont fournies dans le tableau 5.1, avec n la dimension et m le nombre de contraintes d'inégalité.

#	Nom	Source	n	m
1	CHENWANG-F2[§]	[56]	8	6
2	CHENWANG-F3[§]	[56]	10	8
3	CRESCENT	[57]	10	2
4	FLOUDAS	[58]	3	3
5	G2	[46]	10	2
6	G9	[46]	7	4
7	HS83	[58]	5	6
8	HS108	[58]	9	13

#	Nom	Source	n	m
9	HS114	[56]	9	6
10	MAD6[§]	[56]	5	7
11	OPTENG-RBF	[57]	3	4
12	PENTAGON	[58]	6	15
13	SPRING[§]	[57]	3	4
14	TAOWANG-F2[§]	[59]	7	4
15	ZHAOWANG-F5	[60]	13	9

TABLEAU 5.1 Description de l'ensemble de problèmes tests sous contraintes. L'exposant [§] indique l'utilisation de plusieurs points de départ pour le problème donné.

5.1.2 Problème réaliste

La boîte noire solar [2] modélise un simulateur de centrale solaire thermique comme illustré à la figure 5.1. Au total, dix instances nommées respectivement solar.1 à solar.10 sont disponibles via un unique programme `C++` disponible à `https://github.com/bbopt/solar`. Pour les résultats numériques, deux instances sont utilisées :

1. solar1.1 : Le problème d'optimisation associé consiste à maximiser l'énergie collectée par le récepteur sur une période de 24 heures, tout en respectant un budget de 50 millions de dollars et une surface de champ donnée. Il introduit 9 variables, dont une

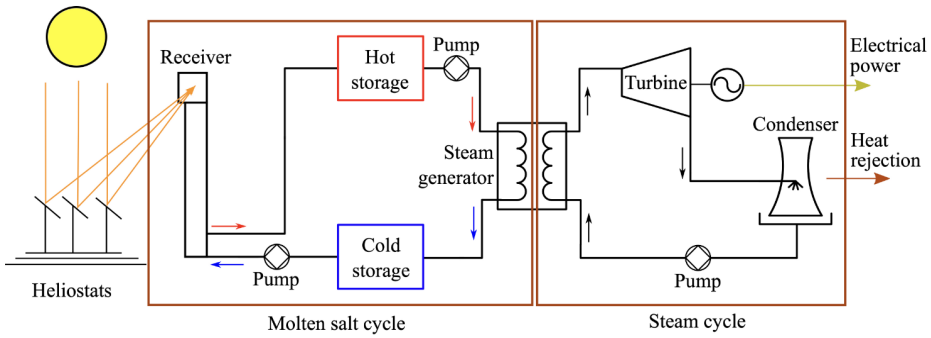

FIGURE 5.1 Processus de la centrale solaire thermique [2]

discrète et est soumis à 5 contraintes relaxables. De plus, la fonction objectif est de nature stochastique.

2. solar4.1 : Le problème d'optimisation minimise le coût total d'investissement de la centrale solaire. Il fait intervenir 29 variables et 16 contraintes. Au total six sorties sont stochastiques soit les contraintes $c_2, c_6, c_7, c_8, c_9, c_{13}$.

Lorsqu'une sortie est stochastique, une simulation de Monte-Carlo est requise, ce qui entraîne alors une précision variable sur la sortie selon le nombre de replications faites. Le nombre maximal de réplications est fixé à 50 000 dans solar et implique un long temps de calcul. Ne bénéficiant pas de parallélisme, on a jugé judicieux de paralléliser le code afin d'y quantifier le parallélisme sur une application réelle dans le contexte de calcul parallèle de ce travail. Une implémentation en mémoire distribuée a été faite afin d'avoir la flexibilité d'utiliser plusieurs ordinateurs.

5.2 Analyse de performance

Étant donné que f et c_j sont coûteux en temps et en ressources de calcul, cela impose une contrainte sur le budget d'évaluations alloué pour résoudre le problème (**P**). Par exemple, une simulation aux grandes échelles (*Large Eddy Simulation*, LES) en turbulence d'un phénomène de combustion industrielle peut nécessiter jusqu'à 750 000 heures de temps `CPU`, en mobilisant 360 cœurs de calcul. Les utilisateurs confrontés à des problèmes d'optimisation aussi coûteux s'intéressent souvent à la performance des solveurs selon le nombre d'évaluations de fonctions.

Les profils de données [55], introduits par Moré et Wild, répondent à ce besoin. Ils offrent une représentation visuelle concise de la performance de solveurs `DFO` pour un ensemble de problèmes tests, en fonction du nombre d'évaluations. Pour définir les profils de données, on considère un ensemble de problèmes tests $\mathscr{P}$ et un ensemble d'algorithmes d'optimisation

$\mathscr{A}$. Pour un algorithme $a \in \mathscr{A}$ et un problème $p \in \mathscr{P}$, il est possible de mesurer la précision de la solution obtenue par a après N évaluations notée $\boldsymbol{x}^N$ en comparaison à la meilleure solution disponible notée $\boldsymbol{x}^*$, via la formule suivante :

$$f_{acc}^N = \frac{f(\boldsymbol{x}^0) - f(\boldsymbol{x}^N)}{f(\boldsymbol{x}^0) - f(\boldsymbol{x}^*)} \tag{5.1}$$

L'équation 5.1 mesure également pour un algorithme $a \in \mathscr{A}$ sa capacité à améliorer la valeur de f après N évaluations par rapport à la valeur initiale de f pour le point initial $\boldsymbol{x}^0$.

De f_{acc}^N, une condition de convergence quantifiant la performance d'un solveur peut être déduite soit $f_{acc}^N \leq 1-\tau$, où $\tau \leq 0$ est une tolérance typiquement $\tau = 10^{-k}$ avec $k \in \{1, 3, 5, 7\}$. Dans l'optique de distinguer les performances des algorithmes entre eux, la variable $T_{a,p}$ est introduite et indique l'atteinte ou non de la condition de convergence pour chacun des problèmes $p \in \mathscr{P}$. Si un algorithme $a \in \mathscr{A}$ résout le problème $p \in \mathscr{P}$ avec une tolérance τ, alors $T_{a,p} = 1$, le cas contraire $T_{a,p} = 0$. De plus, lorsque $T_{a,p} = 1$, on note $N_{a,p}$ le nombre minimum d'évaluations requis tel que $f_{acc}^{N_{a,p}} \geq 1 - \tau$.

Définition 5.1 (Profil de données). La proportion de problèmes résolus avec succès par l'algorithme $a \in \mathscr{A}$ sur l'ensemble de problèmes tests $\mathscr{P}$, avec un budget de $k \in \mathbb{N}$ groupes d'évaluations de la fonction, est définie par la fonction $d_a : \mathbb{N} \to [0, 1]$

$$d_a(k) = \frac{1}{|\mathscr{P}|} \left|\{p \in \mathscr{P} \mid N_{a,p} \leq k(n_p + 1)T_{a,p}\}\right| ,$$

où $|\cdot|$ représente la cardinalité de l'ensemble. Le profil de données est créé en traçant la fonction d_a pour tous les algorithmes de $\mathscr{A}$ sur un même graphique.

Dans un contexte de calcul parallèle, nous sommes souvent plus intéressés par le temps minimal requis par un algorithme $a \in \mathscr{A}$ afin de résoudre un problème $p \in \mathscr{P}$, puisque l'apport du parallélisme se quantifie en termes de temps et non en termes d'évaluations.

Définition 5.2 (Profil de données en temps). La proportion de problèmes résolus avec succès par l'algorithme $a \in \mathscr{A}$ sur l'ensemble de problèmes tests $\mathscr{P}$, avec un budget de $t \in \mathbb{R}^+$ de temps, est définie par la fonction $d_a : \mathbb{R}^+ \to [0, 1]$

$$d_a(t) = \frac{1}{|\mathscr{P}|} \left|\{p \in \mathscr{P} \mid N_{a,p} \leq t\}\right| ,$$

où $|\cdot|$ représente la cardinalité de l'ensemble. Le profil de données est créé en traçant la fonction d_a pour tous les algorithmes de $\mathscr{A}$ sur un même graphique.

5.3 Résultats

La section des résultats est divisée en deux parties, chacune étant associée à une version de NOMAD. Pour l'analyse de performances des variantes parallèles de `MADS`, la version 3 de NOMAD est mise de l'avant puisque les implémentations numériques de `COOP-MADS` et `PSD-MADS` sont stables comparativement à celles de la version 4. Les tests numériques liés à la question de recherche sont faits avec NOMAD 4.

Les deux versions ont été utilisées car NOMAD 3 n'est plus développé depuis 2018, tandis que NOMAD 4 est en développement continu et intègre de nouvelles fonctionnalités annuellement.

5.3.1 NOMAD 3

Les tests numériques ont été exécutés sur plusieurs ordinateurs tous muni de 64 Go de mémoire vive `DDR5`, un processeur `Intel(R) Core(TM) i7-12700K @ 2.10 GHz` et la distribution Linux `AlmaLinux 9.4`. La version `3.9.1` de NOMAD est utilisée avec `OpenMPI 4.1.2`.

Les paramètres par défaut de NOMAD 3 pour les variantes parallèles sont utilisés, ce qui implique la désactivation des modèles quadratiques. De plus, le choix des directions de l'algorithme `OrthoMADS` est $2n$, car les directions dynamiques $n+1$ ne sont pas implémentées pour les variantes parallèles. Le nombre de ressources parallèles utilisées est choisi en fonction de la dimension du problème considéré et du nombre de *workers* parallèles capables de calculer une évaluation selon l'algorithme utilisé. En effet, selon l'algorithme, le nombre de *workers* parallèles peut varier. Pour `pMADS` et `COOP-MADS`, un *worker* est désigné comme *manager* parmi le total de processus fournis à `MPI`, ce qui conduit à l'utilisation de $n+1$ processus. En revanche, `PSD-MADS` utilise un *manager* et un *cache server* qui ne calculent pas d'évaluations, ce qui nécessite l'utilisation de $n+2$ processus.

Moré-Wild

Afin d'augmenter le nombre de problèmes tests fournit par cette collection, la graine aléatoire (paramètre `SEED`) de NOMAD 3 est fixée à $s \in \{0,\ 11,\ 743,\ 88,\ 5,\ 1021,\ 69,\ 33,\ 100,\ 55\}$ ce qui permet de générer un ensemble de 530 problèmes tests.

Le premier test numérique compare les deux variantes de `COOP-MADS` soit selon que le paramètre `CACHE_SEARCH` est activé ou non.

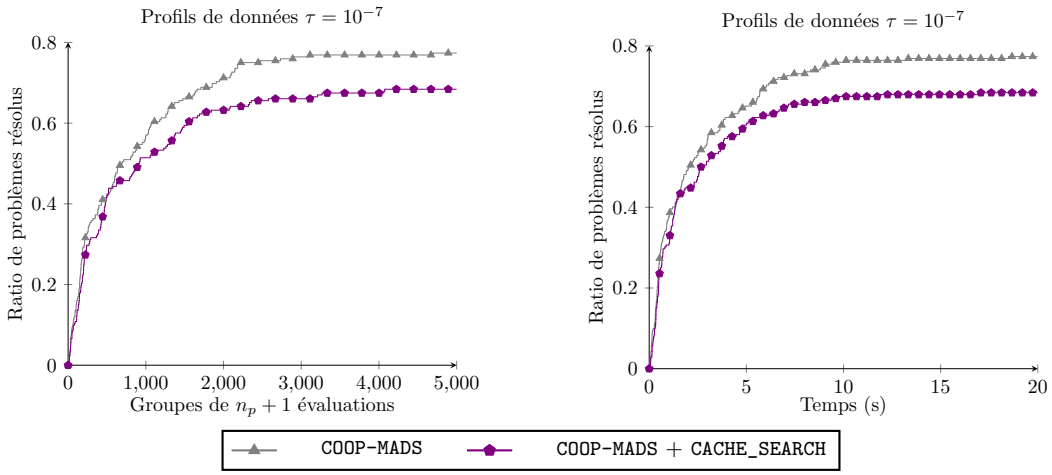

FIGURE 5.2 Impact de l'utilisation de l'option `CACHE_SEARCH` sur les performances de l'algorithme `COOP-MADS`.

Ces résultats sont cohérents avec le comportement attendu lors de l'activation du paramètre. En effet, lorsque ce paramètre est activé, les différentes instances parallèles de `MADS` sont moins susceptibles de suivre des trajectoires différentes, ce qui limite l'exploration de l'espace des variables et privilégie un comportement d'exploitation. À l'inverse, la variante ne faisant pas usage de la `CACHE_SEARCH` permet aux instances de travailler de manière indépendante sur l'ensemble de l'espace $\mathcal{X}$, ce qui favorise la génération de trajectoires distinctes et encourage l'exploration de bassins prometteurs. Compte tenu du résultat ci-haut indiquant que la méthode sans `CACHE_SEARCH` présente les meilleures performances, les tests suivants faisant intervenir `COOP-MADS` sont faits en désactivant l'option.

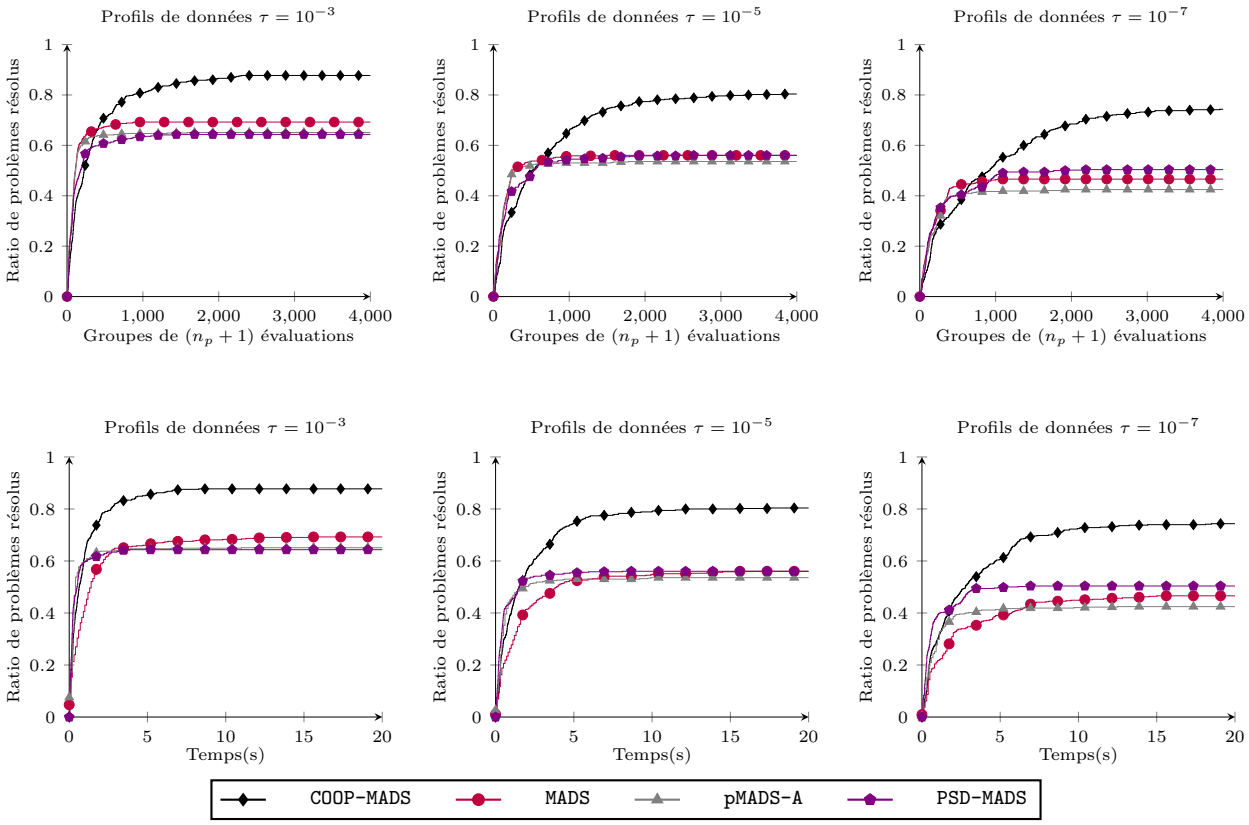

FIGURE 5.3 Comparaison des variantes parallèles de MADS sur l'ensemble de problème tests Moré Wild.

L'algorithme COOP-MADS est celui qui offre les meilleures performances. Cela s'explique par le fait qu'il utilise différentes instances, chacune travaillant sur un treillis distinct qui converge à des vitesses différentes. En revanche, les algorithmes MADS et pMADS-A ont tendance à converger rapidement vers un minimum local, leur treillis unique atteint la taille minimale, ce qui signifie la convergence. De même, PSD-MADS présente un comportement similaire, car il repose également sur un treillis commun. En somme, l'utilisation de treillis distincts permet d'éviter une convergence hâtive de l'algorithme. Cependant, COOP-MADS a un coût plus élevé en termes d'évaluations. Néanmoins, si l'on compare le temps de calcul, on constate que, au moment où MADS converge, COOP-MADS affiche un ratio d'environ 0,2 de plus que MADS pour le même temps de calcul, ce qui témoigne de l'efficacité de cette méthode, 20 % de 530 problèmes signifie que COOP-MADS résout 106 problèmes de plus que MADS.

Le résultat de PSD-MADS est surprenant, car cet algorithme conçu pour résoudre des problèmes de grande dimension parvient à offrir des performances similaires à celles de MADS. Le prochain test numérique porte sur des problèmes contraints, ce qui ajoute une complexité supplémentaire.

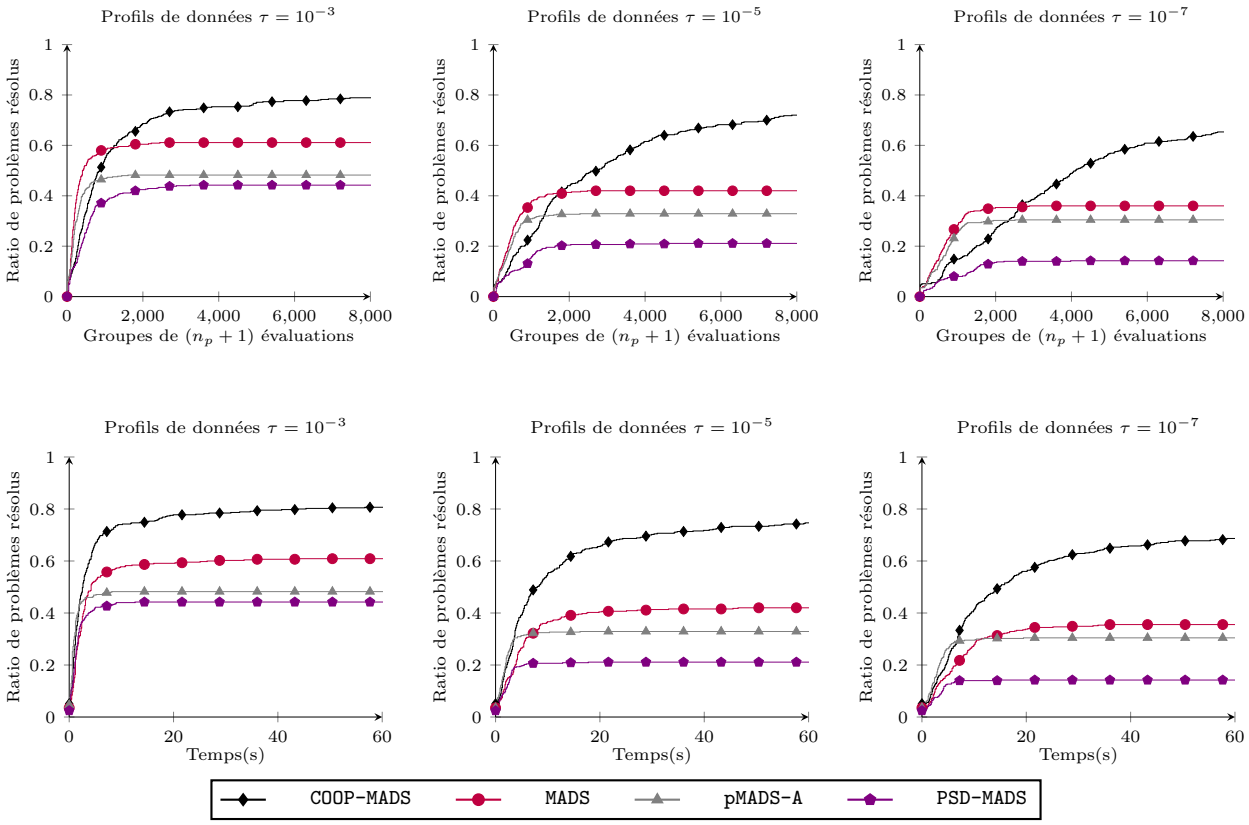

FIGURE 5.4 Comparaison des variantes parallèles de MADS sur un ensemble de problèmes tests avec contraintes d'inégalités.

L'ajout de contraintes affecte négativement les performances de pMADS-A, et surtout celles de PSD-MADS. La source potentielle de cette performance est liée à la barrière progressive, puisque ces deux variantes parallèles mettent à jour le seuil $h^k_{\max}$ différemment de MADS et COOP-MADS, ce qui affecte l'efficacité de la méthode. De son côté, COOP-MADS conserve sa supériorité, bien qu'il nécessite un grand nombre d'évaluations avant de converger. Après 8000 évaluations pour des précisions de $p \in \{10^{-5},\ 10^{-7}\}$, l'algorithme n'a toujours pas convergé. Cependant, en termes de temps, COOP-MADS reste la courbe la plus élevée sur toute la plage de temps mesurée, ce qui témoigne une nouvelle fois de son efficacité parallèle.

Boîte noire à temps hétérogène

Comme indiqué dans le chapitre 1, une boîte noire peut présenter un comportement à temps hétérogène. Si ce phénomène n'a pas d'impact dans un environnement de programmation séquentiel, il peut cependant affecter de manière significative les performances de l'algorithme dans un environnement de programmation parallèle.

Ce test vise à illustrer l'impact de la barrière de synchronisation sur le temps de calcul, en comparant les algorithmes `pMADS-A` et `pMADS-S` sur une boîte noire à temps hétérogène.

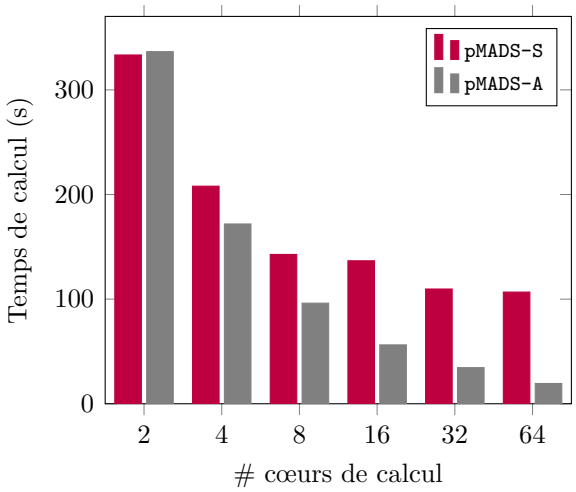

FIGURE 5.5 Temps de calcul de `pMADS-A` et `pMADS-S` avec un nombre différent de coeurs de calcul pour résoudre solar4.1 avec un budget de 500 évaluations de la boîte noire.

La figure 5.5 illustre qu'à mesure que le nombre de cœurs de calcul impliqués augmente, l'écart de temps entre les deux versions se creuse. Cela s'explique par la présence de la barrière de synchronisation dans `pMADS-S`, qui implique qu'il faut seulement une évaluation ayant un temps de calcul significativement plus élevé que les autres exécutées en parallèle, pour entraîner une pause des cœurs de calcul. Par conséquent, l'augmentation du nombre de cœurs conduit à davantage d'évaluations lancées en parallèle, augmentant ainsi la probabilité de rencontrer une évaluation plus coûteuse à chaque itération k. Ce résultat met en évidence l'importance de privilégier un algorithme parallèle asynchrone lorsqu'une boîte noire présente un comportement à temps hétérogène.

Mise à l'échelle

Les méthodes parallèles de `MADS` ont été conçues pour permettre d'éventuels gains en accélération. Afin de vérifier si ces gains sont effectivement réalisés, une étude de mise à l'échelle

est nécessaire, comme détaillé au chapitre 2. Étant donné que la seule variante parallèle de MADS disposant également d'une version séquentielle est pMADS-A, on présente des résultats pour cette variante uniquement.

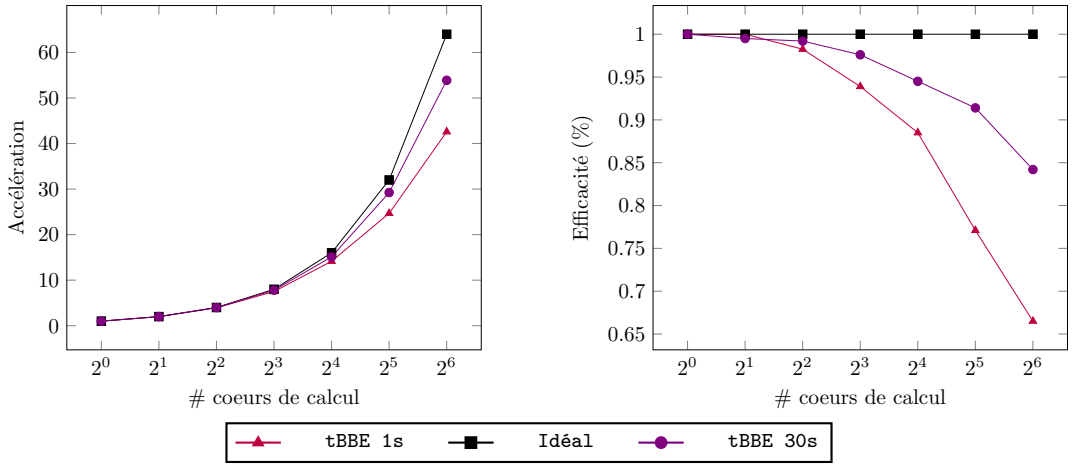

FIGURE 5.6 Accélération parallèle et efficacité parallèle de pMADS-A mesurées sur un problème d'optimisation de 32 variables avec un budget de 500 évaluations et un temps d'évaluation de la boîte noire (tBBE) de 1 seconde et 30 secondes.

Les résultats numériques montrent que l'implémentation parallèle permet d'accélérer de manière significative le temps de calcul. Même dans le cas de l'efficacité la plus faible mesurée, un facteur d'accélération de 43 est atteint en utilisant 64 cœurs de calcul. Dans le contexte de BBO, où le temps d'une évaluation dépasse largement la seconde, cette performance peut être considérée comme un seuil minimum. Pour des boîtes noires plus exigeantes en temps de calcul, on anticipe une amélioration supplémentaire des performances parallèles, puisque l'impact du surcoût lié à la communication devient négligeable. Une preuve de concept est présentée en ajoutant un délai artificiel permettant d'atteindre 30 secondes comme temps d'évaluation. Le résultat confirme nos attentes : à l'efficacité la plus faible, le facteur d'accélération passe de 43 à 54.

5.3.2 NOMAD 4

Un environnement de simulation identique à celui utilisé pour les tests de NOMAD 3 est employé. Cette fois, la version 4.4.0 de NOMAD est utilisée, avec le parallélisme en mémoire partagée pris en charge par le standard OpenMP. Les différents algorithmes composant l'ensemble $\mathscr{A}$ correspondent à pMADS-S et se distinguent par le nombre de *threads* alloué à NOMAD via le paramètre NB_THREADS_PARALLEL_EVAL.

De plus, on simule le parallélisme d'une évaluation de la boîte de sortes à fournir des courbes en fonction du temps.

Sommairement, les algorithmes sont les suivants :

1. `NOMAD_10` qui alloue 10 cœurs de calcul au solveur NOMAD ;
2. `NOMAD_5` qui alloue 5 cœurs de calcul au solveur NOMAD et qui divise le temps d'une évaluation de la boîte noire par deux avec une efficacité de 100 % ;
3. `NOMAD_2` qui alloue 2 cœurs de calcul au solveur NOMAD et qui divise le temps d'une évaluation de la boîte noire par cinq avec une efficacité de 90 % ;
4. `NOMAD_1` qui alloue 1 cœur de calcul au solveur NOMAD et qui divise le temps d'une évaluation de la boîte noire par dix avec une efficacité de 85 %.

Les tests numériques sont faits pour trois dimensions $d \in \{5, 10, 20\}$ afin de mesurer l'impact des performances en fonction de la dimension.

Pour chaque dimension afin de mesurer l'impact de l'utilisation du modèle quadratique avec la stratégie opportuniste, on effectue une première phase de tests en désactivant les modèles quadratiques dans NOMAD, tout en laissant les autres paramètres par défaut, à savoir la recherche globale par recherche spéculative (`SPECULATIVE_SEARCH`) et l'algorithme Nelder-Mead (`NM_SEARCH`), et l'ordonnancement de la file d'évaluations selon la direction de dernier succès (`DIR_LAST_SUCCESS`).

La seconde phase des tests consiste à activer le modèle quadratique pour ordonner les points de la file d'évaluation via le paramètre (`EVAL_QUEUE_SORT QUADRATIC_MODEL`).

Le choix des directions pour les deux cas est fixé à `OrthoMADS` $2n$ (`DIRECTION_TYPE 2N`).

L'ensemble de problèmes tests utilisé est la collection `Napsu`. Comme mentionné précédemment, ces problèmes sont difficiles à résoudre et donc un budget d'évaluations a été prévu en conséquence : 50 000 évaluations pour $n = 5$ et 100 000 évaluations pour $n \in \{10, 20\}$.

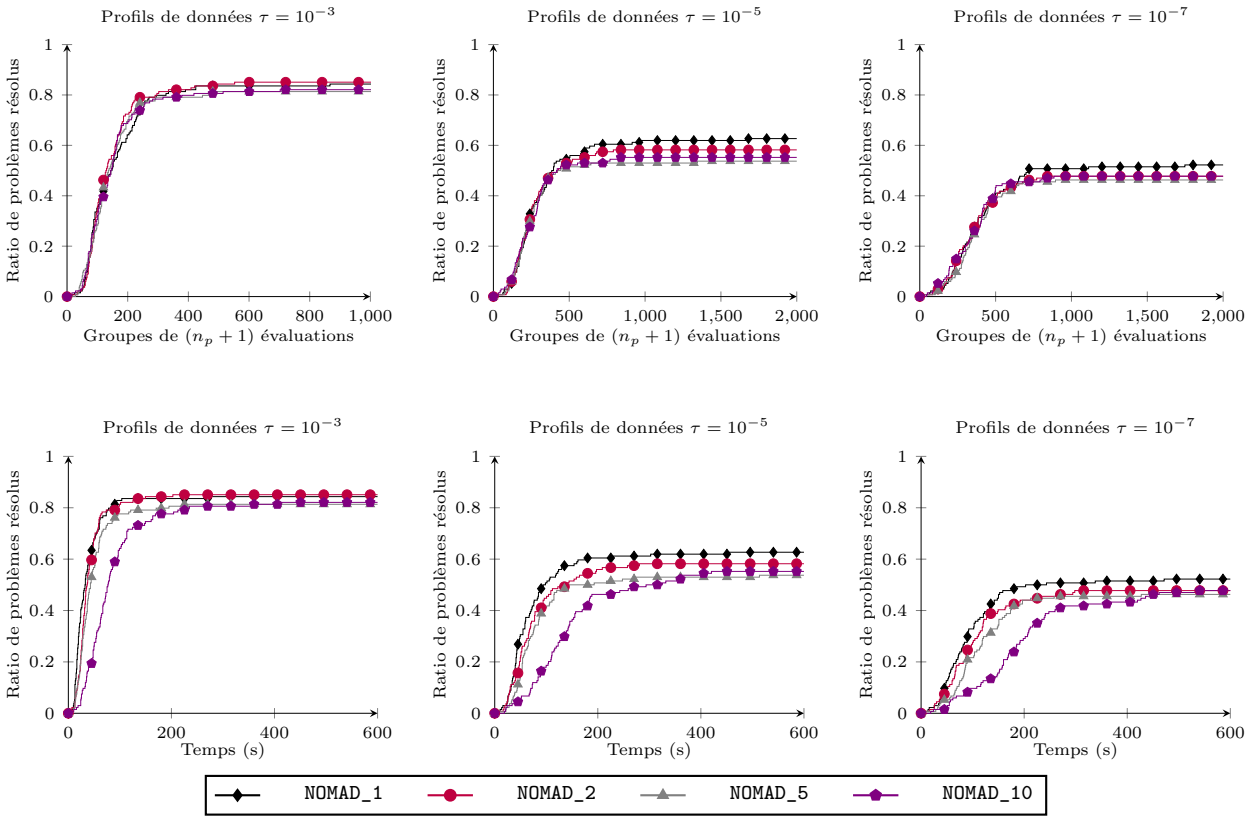

FIGURE 5.7 Modèles quadratiques désactivés pour la dimension $n = 5$.

Bien que les résultats montrent des tendances globalement comparables, `NOMAD_1` offre les meilleures performances, notamment pour des précisions élevées, $p \in \{10^{-5}, 10^{-7}\}$. Il convient de noter que l'algorithme `NOMAD_10` pour la dimension $n = 5$ ne bénéficie pas de la stratégie opportuniste, car le nombre de points d'évaluation dans la file d'évaluation est identique au nombre de cœurs alloués au solveur, ce qui constitue une situation potentiellement défavorable. En outre, il ressort également que `NOMAD_10` est l'algorithme qui met le plus de temps à converger, bien que tous les algorithmes utilisent le même nombre de ressources de calcul.

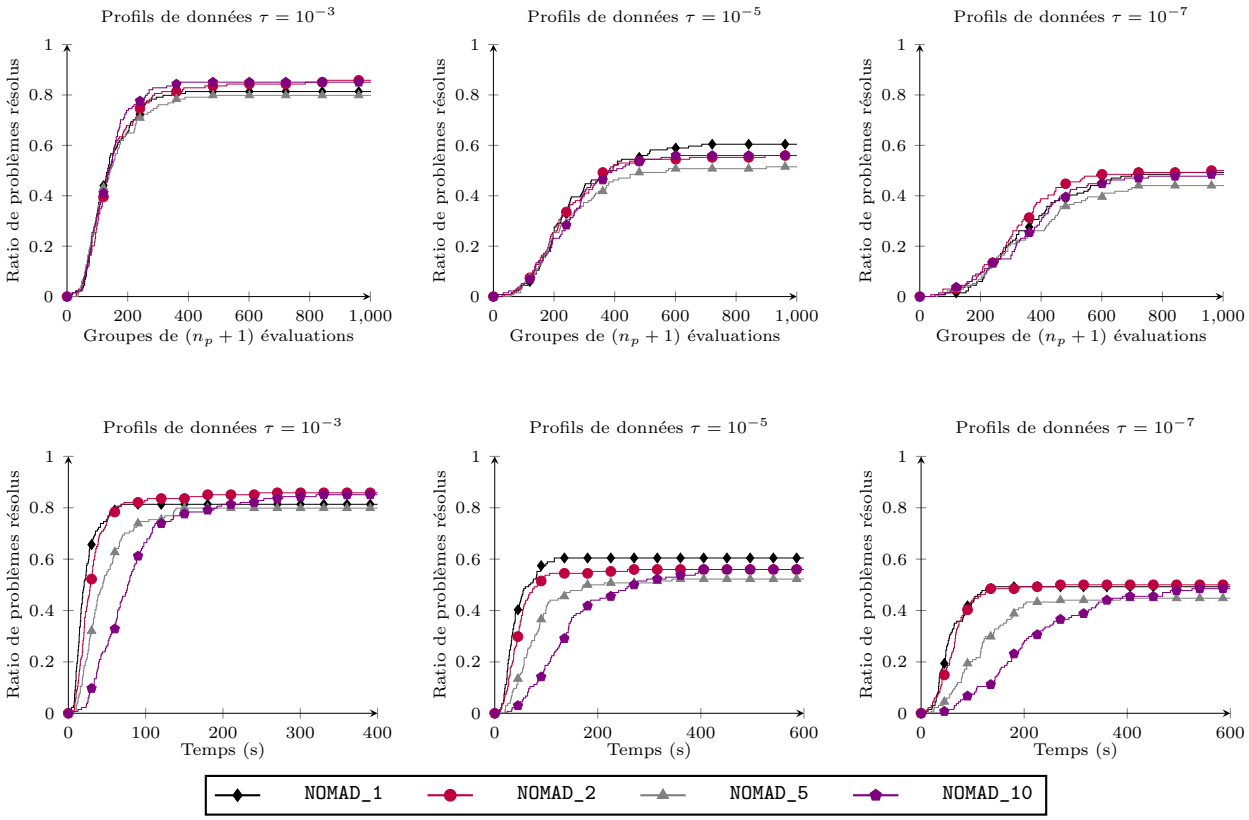

FIGURE 5.8 Modèle quadratique activé pour la file d'évaluations en dimension $n = 5$.

L'activation du modèle quadratique pour ordonner les points de la file d'évaluations ne semble pas avoir un impact majeur sur les performances de chaque algorithme, à l'exception de `NOMAD_5`, pour lequel une baisse de performance est observée. En termes de temps de calcul, on peut constater une différence notable dans la vitesse de convergence de `NOMAD_1` et `NOMAD_2` par rapport à leurs homologues, notamment `NOMAD_10`, qui affiche, pour la seconde fois, la convergence la plus lente.

En résumé, pour la dimension $n = 5$, l'ensemble des algorithmes présente des performances comparables, sans qu'un algorithme ne se distingue clairement des autres. Cela suggère implicitement que, pour cette dimension, l'algorithme `MADS` fait preuve d'une robustesse face au parallélisme en termes de qualité de solution. Les prochains tests numériques augmentent la dimension afin d'évaluer si une tendance similaire se maintient.

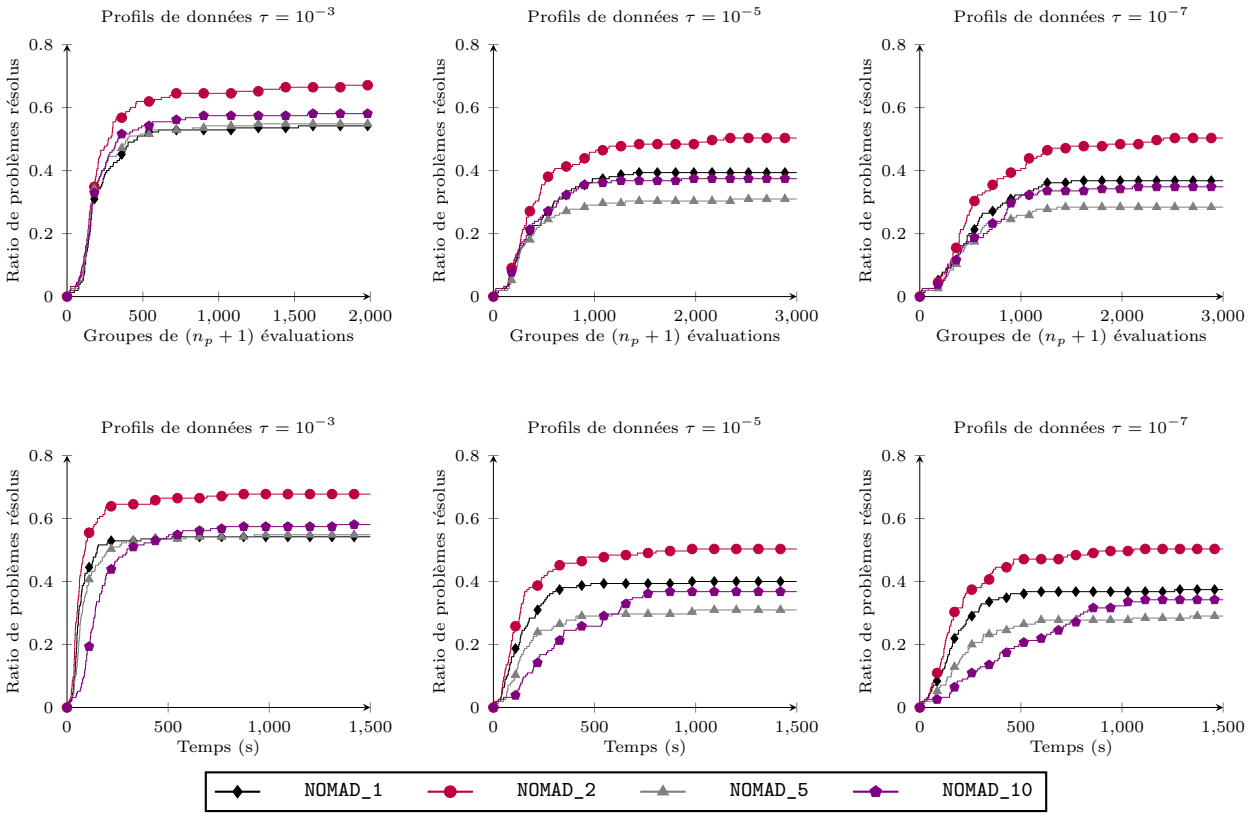

FIGURE 5.9 Modèles quadratiques désactivés pour la dimension $n = 10$

L'augmentation de la dimension a permis de mieux différencier les performances des algorithmes. En effet, `NOMAD_2` se distingue de manière significative sur les trois niveaux de précision, avec une différence d'environ 15 % en termes de problèmes résolus par rapport à `NOMAD_1`, qui occupe la deuxième place. Cette différence représente 24 problèmes résolus en plus pour `NOMAD_2`. Il apparaît donc évident que `NOMAD_2` est l'algorithme le plus performant.

En observant la rapidité avec laquelle la courbe de `NOMAD_2` progresse en fonction des évaluations, on constate que l'ordonnancement des points pour `NOMAD_2` est particulièrement efficace, car un nombre réduit d'évaluations suffit à résoudre un grand nombre de problèmes.

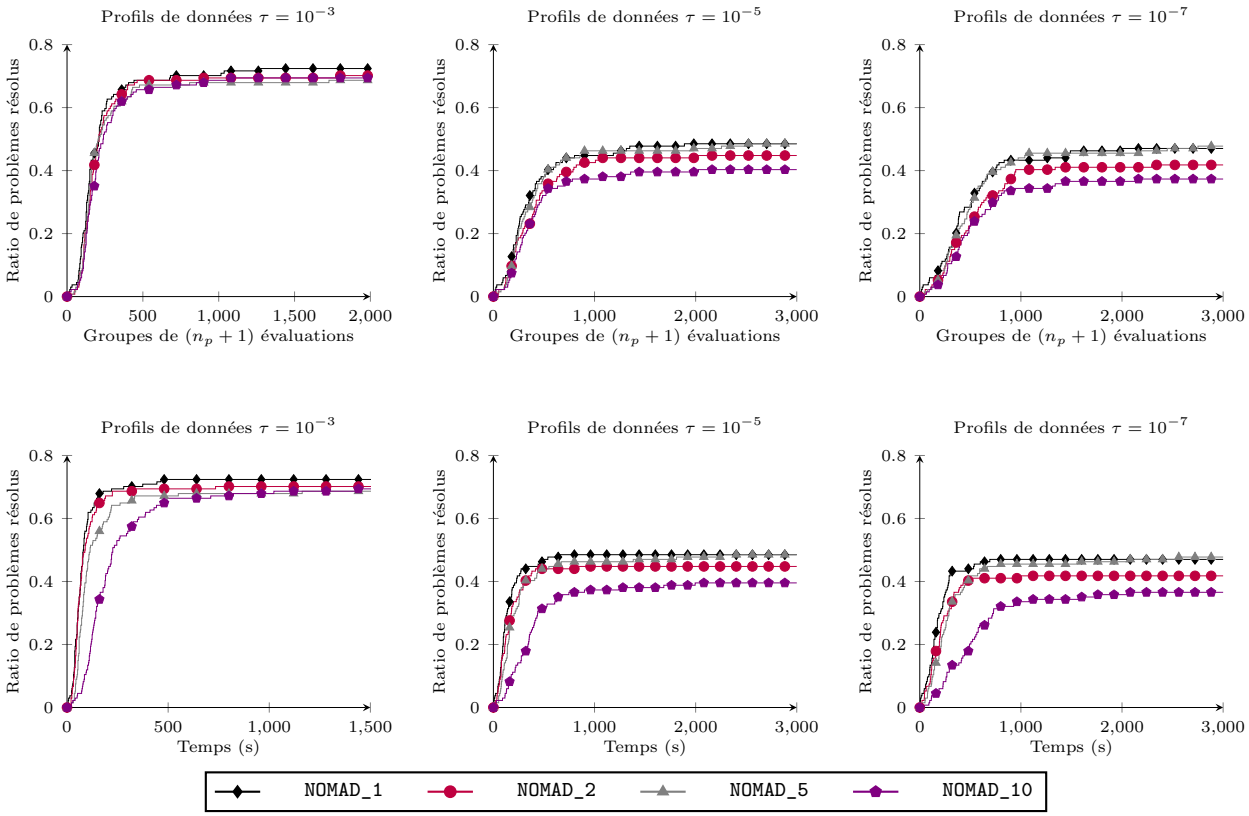

FIGURE 5.10 Modèle quadratique activé pour la file d'évaluations en dimension $n = 10$.

L'activation du modèle quadratique atténue les performances de `NOMAD_2` puisqu'il ne se distingue plus considérablement des autres. À l'inverse le modèle quadratique est bénéfique pour `NOMAD_5` qui au précédent test était en dernière position et qu'à présent se trouve en première position avec `NOMAD_1`.

La dimension $n = 10$ a permis de distinguer les performances de certains algorithmes entre eux sans toutefois permettre d'établir un algorithme favorable sur les deux tests.

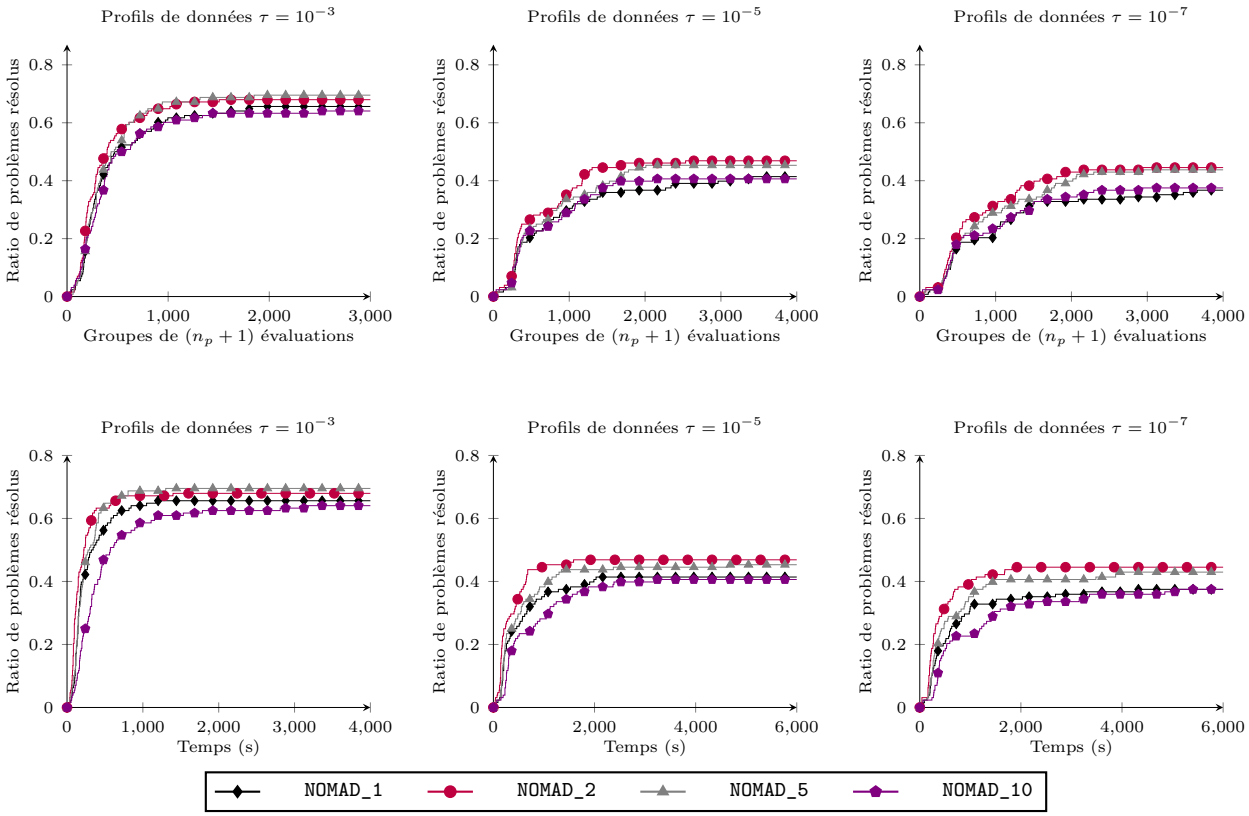

FIGURE 5.11 Modèles quadratiques désactivés pour la dimension $n = 20$.

La présence de problèmes avec un grand nombre de variables et de points candidats dans la file d'évaluation affecte `NOMAD_1` qui pour la première fois, se retrouve en dernière position. Il semble donc que la stratégie opportuniste par défaut de NOMAD, appliquée à un bloc de taille un ne soit plus aussi efficace pour cette dimension. Un résultat surprenant est que `NOMAD_10` obtient des performances identiques à celles de `NOMAD_1`. On pourrait intuitivement penser que, pour des dimensions plus grandes, effectuer davantage d'évaluations sur un treillis serait bénéfique, mais cela ne semble pas être le cas.

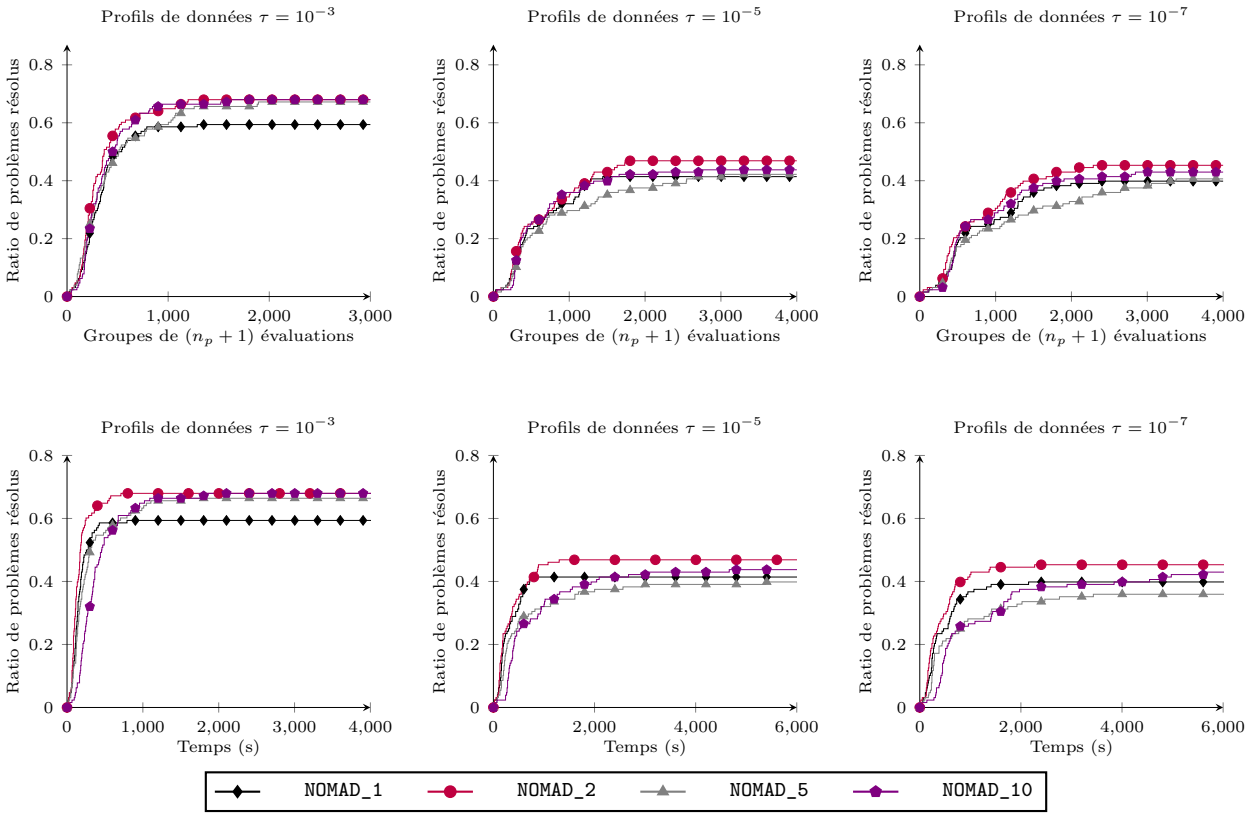

FIGURE 5.12 Modèle quadratique activé pour la file d'évaluations en dimension $n = 20$.

L'activation du modèle quadratique apporte un gain de performance notable à `NOMAD_1` et `NOMAD_10`. En revanche, le modèle entraîne une diminution significative des performances de `NOMAD_5`, qui présente une vitesse de convergence très lente en termes d'évaluations.

Afin de clore les tests numériques analytiques , on note que `NOMAD_2` se distingue comme l'algorithme offrant le meilleur classement cumulatif à travers tous les tests et pour chaque dimension. Cependant, il est difficile d'expliquer de manière systématique les raisons sous-jacentes aux comportements observés pour chaque algorithme.

Boîte noire solar

Le code solar a été parallélisé via `MPI`, ce qui nous permet de mesurer réellement l'effet du parallélisme, et non de le simuler comme précédemment. Le nombre de réplications utilisées est fixé à 10 000. De légères modifications ont été apportées dans le générateur de nombres aléatoires afin d'obtenir un code parallèle déterministe indépendamment du nombre de cœurs de calcul utilisé. L'instance solar1.1 fera l'objet de ce test numérique. Les algorithmes employés sont les suivants :

1. `NOMAD_8` qui alloue tous les 8 cœurs de calcul au solveur ;
2. `NOMAD_4` qui alloue 4 cœurs de calcul au solveur et 2 cœurs de calcul à la boîte noire ;
3. `NOMAD_2` qui alloue 2 cœurs de calcul au solveur et 4 cœurs de calcul à la boîte noire ;
4. `NOMAD_1` qui alloue tous les 8 cœurs de calcul à la boîte noire.

La répartition de cœurs de calcul a été modifiée en raison que les processeurs `Intel(R) Core(TM) i7-12700K @ 2.10 GHz` disposent de 8 cœurs fonctionnant à une fréquence élevée et 4 cœurs à une fréquence plus faible. Afin d'éviter des variations importantes de temps de calcul, on se limite à 8 cœurs de calcul par machine. La mise à l'échelle de l'implémentation `MPI` est présentée à la figure 5.13. Les résultats montrent que l'implémentation est performante pour 8 cœurs de calcul.

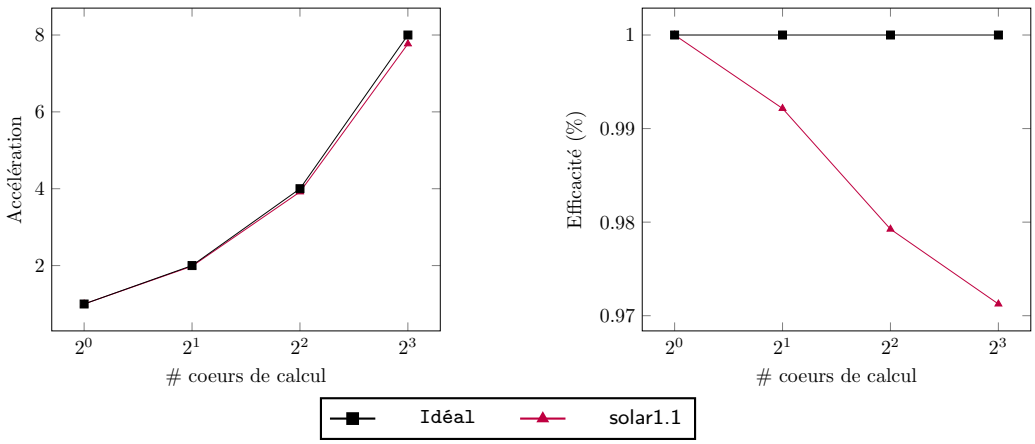

FIGURE 5.13 Accélération parallèle et efficacité parallèle de solar1.1 mesurées sur le point de départ $\boldsymbol{x}^0 = \begin{bmatrix} 8 & 8 & 150 & 7 & 7 & 250 & 45 & 0,55 \end{bmatrix}$.

Le temps de calcul de $f(\boldsymbol{x}^0)$ est d'environ 697 secondes soit 11m 37s. Un budget de temps de calcul de 60 heures est fourni par algorithme pour l'optimisation.

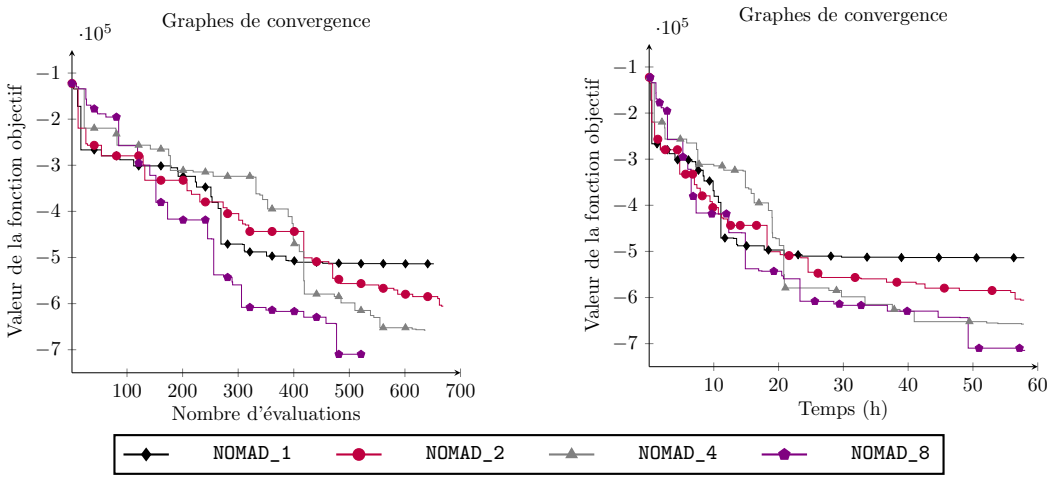

FIGURE 5.14 Optimisation de solar1.1.

Ce que l'on peut conclure du résultat de la figure 5.14, est que bien que `NOMAD_8` effectue le moins d'évaluations que les autres algorithmes, les évaluations réalisées par `NOMAD_8` sont plus efficaces pour faire diminuer f. De plus, le classement selon la valeur de f indique que plus un algorithme alloue un grand nombre de ressources de calcul au solveur mieux il performe. Le fait d'allouer plus de ressources de calcul au solveur confère à un algorithme une plus grande capacité d'exploration, car la stratégie opportuniste s'applique sur un bloc b de plus grande taille.

À l'opposé, la stratégie qui favorise l'exploitation en allouant un cœur de calcul au solveur, soit `NOMAD_1` semble converger hâtivement vers un minima local, puisqu'on remarque que pour plus de 250 évaluations (40 heures), la valeur de f n'est pas en mesure d'être améliorée, ce qui donne lieu à un plateau.

Les problèmes réalistes, de par leur complexité, permettent souvent de différencier plus facilement les performances des différents algorithmes, comme c'est le cas ici. Toutefois, afin de s'assurer de la généralisation des performances des algorithmes, il est nécessaire d'élargir le nombre de tests numériques en utilisant différentes instances de solar et en augmentant également le budget de temps de calcul afin de permettre à tous les algorithmes de converger.

5.3.3 Qualité des modèles quadratiques

Ce test utilise 20 problèmes issus de l'ensemble de problèmes tests `Napsu`. Il évalue l'efficacité des modèles quadratiques dans l'ordonnancement des points d'évaluation pour la stratégie opportuniste. Les paramètres du tableau incluent :

1. k, le nombre d'itérations nécessaires pour résoudre le problème d'optimisation ;

2. `succès`, le nombre total de succès cumulés au cours du processus d'optimisation ;
3. `BBE`, le nombre d'évaluations de la boîte noire requis ;
4. r_1, le ratio du nombre de succès par itération ;
5. r_2, le ratio du nombre d'évaluations par succès.

Les tests ont été réalisés pour des dimensions $n \in \{5, 10, 20\}$, avec un budget de $1000n$ évaluations. La valeur r_1 représente le pourcentage d'itérations qui ont été un succès et constitue notre principale métrique de performance. L'algorithme affichant la plus haute valeur de r_1 est mis en évidence, en surlignant en gris la cellule associée.

TABLEAU 5.2 Dimension $n = 5$.

NOMAD_1					NOMAD_2					NOMAD_5					NOMAD_10				
k	succès	BBE	r_1	r_2	k	succès	BBE	r_1	r_2	k	succès	BBE	r_1	r_2	k	succès	BBE	r_1	r_2
106	61	747	0,58	12,25	122	57	862	0,47	15,12	179	52	1026	0,29	19,73	303	52	1215	0,17	23,37
149	82	1096	0,55	13,37	375	106	879	0,28	8,29	394	73	1512	0,19	20,71	242	76	1706	0,31	22,45
172	90	1308	0,52	14,53	206	86	1519	0,42	17,66	527	84	1687	0,16	20,08	443	80	1970	0,18	24,62
512	438	1553	0,86	3,55	238	148	1710	0,62	11,55	258	150	2414	0,58	16,09	443	245	5006	0,55	20,43
164	98	1136	0,60	11,59	718	93	1316	0,13	14,15	148	91	1318	0,61	14,48	215	87	1873	0,40	21,53
968	877	2441	0,91	2,78	334	258	1671	0,77	6,48	251	156	2212	0,62	14,18	153	90	1684	0,59	18,71
199	120	1345	0,60	11,21	273	106	1625	0,39	15,33	263	123	1769	0,47	14,38	305	111	2252	0,36	20,29
325	209	2015	0,64	9,64	353	211	2200	0,60	10,43	443	193	2925	0,44	15,16	454	182	3624	0,40	19,91
214	122	1603	0,57	13,14	222	123	1697	0,55	13,80	299	114	2085	0,38	18,29	259	112	2445	0,43	21,83
157	95	1127	0,61	11,86	408	86	1356	0,21	15,77	547	82	1529	0,15	18,65	411	83	1797	0,20	21,65
124	71	877	0,57	12,35	197	62	1103	0,31	17,79	222	62	1271	0,28	20,50	180	63	1420	0,35	22,54
101	58	694	0,57	11,97	144	53	842	0,37	15,89	117	51	994	0,44	19,49	198	45	1164	0,23	25,87
153	83	1142	0,54	13,76	180	87	1209	0,48	13,90	205	73	1568	0,36	21,48	168	84	1712	0,50	20,38
196	110	1448	0,56	13,16	164	94	1243	0,57	13,22	191	115	1742	0,60	15,15	176	107	1925	0,61	17,99
145	88	963	0,61	10,94	236	81	1147	0,34	14,16	119	66	1089	0,55	16,50	195	72	1660	0,37	23,06
197	122	1246	0,62	10,21	171	98	1260	0,57	12,86	251	119	1789	0,47	15,03	310	108	2245	0,35	20,79
316	194	1972	0,61	10,16	167	97	1140	0,58	11,75	358	185	2921	0,52	15,79	213	148	2323	0,69	15,70
455	273	3119	0,60	11,42	395	265	2622	0,67	9,89	403	301	3300	0,75	10,96	350	255	3784	0,73	14,84
230	138	1651	0,60	11,96	224	128	1820	0,57	14,22	535	130	2205	0,24	16,96	298	137	2585	0,46	18,87
131	70	991	0,53	14,16	188	62	1220	0,33	19,68	177	65	1328	0,37	20,43	144	76	1484	0,53	19,53

TABLEAU 5.3 Dimension $n = 10$.

NOMAD_1					NOMAD_2					NOMAD_5					NOMAD_10				
k	succès	BBE	r_1	r_2	k	succès	BBE	r_1	r_2	k	succès	BBE	r_1	r_2	k	succès	BBE	r_1	r_2
331	210	4217	0,63	20,08	238	149	3168	0,63	21,26	166	104	2470	0,63	23,75	529	204	5868	0,39	28,76
261	168	3317	0,64	19,74	154	98	1861	0,64	18,99	156	105	2204	0,67	20,99	251	157	4363	0,63	27,79
763	630	4933	0,83	7,83	506	332	6727	0,66	20,26	217	130	3240	0,60	24,92	303	194	5198	0,64	26,79
816	737	3680	0,90	4,99	212	128	2872	0,60	22,44	158	97	2208	0,61	22,76	344	228	5786	0,66	25,38
1180	991	7517	0,84	7,59	323	226	3492	0,70	15,45	457	309	6753	0,68	21,85	398	273	6745	0,69	24,71
226	149	2619	0,66	17,58	529	156	2532	0,29	16,23	207	139	2768	0,67	19,91	270	149	3884	0,55	26,07
299	199	3487	0,67	17,52	569	196	3932	0,34	20,06	180	132	2246	0,73	17,02	444	207	4970	0,47	24,01
37	0	726	0,00	-	37	0	726	0,00	-	37	0	726	0,00	-	37	0	726	0,00	-
511	322	6555	0,63	20,36	244	157	3114	0,64	19,83	301	195	4546	0,65	23,31	575	325	8890	0,57	27,35
133	77	1747	0,58	22,69	598	68	2106	0,11	30,97	183	69	2237	0,38	32,42	199	65	2488	0,33	38,28
173	102	2222	0,59	21,78	169	114	1952	0,67	17,12	518	100	2955	0,19	29,55	577	95	3060	0,16	32,21
184	130	2100	0,71	16,15	253	114	2541	0,45	22,29	188	121	2681	0,64	22,16	235	107	3324	0,46	31,07
913	726	7385	0,80	10,17	522	341	6782	0,65	19,89	371	234	5763	0,63	24,63	573	355	10004	0,62	28,18
424	265	5563	0,62	20,99	197	130	2203	0,66	16,95	224	142	3233	0,63	22,77	244	150	4209	0,61	28,06
249	172	2713	0,69	15,77	315	153	3434	0,49	22,44	249	151	3744	0,61	24,79	247	168	4075	0,68	24,26
227	159	2427	0,70	15,26	260	163	2421	0,63	14,85	186	127	2435	0,68	19,17	549	133	4034	0,24	30,33
218	150	2410	0,69	16,07	579	128	3100	0,22	24,22	227	148	2992	0,65	20,22	253	158	3587	0,62	22,70
37	0	728	0,00	-	37	0	728	0,00	-	37	0	728	0,00	-	37	0	728	0,00	-
343	217	4323	0,63	19,92	610	211	4878	0,35	23,12	223	146	3243	0,65	22,21	411	217	6033	0,53	27,80
236	140	3206	0,59	22,90	189	127	2262	0,67	17,81	178	106	2676	0,60	25,25	332	136	4267	0,41	31,38

TABLEAU 5.4 Dimension $n = 20$.

NOMAD_1					NOMAD_2					NOMAD_5					NOMAD_10				
k	succès	BBE	r_1	r_2	k	succès	BBE	r_1	r_2	k	succès	BBE	r_1	r_2	k	succès	BBE	r_1	r_2
359	262	7653	0,73	29,21	883	248	8256	0,28	33,29	662	244	9014	0,37	36,94	736	237	10389	0,32	43,84
433	303	10305	0,70	34,01	891	296	10593	0,33	35,79	731	297	10769	0,41	36,26	742	326	11845	0,44	36,33
468	364	10580	0,78	29,07	384	276	9418	0,72	34,12	850	356	11168	0,42	31,37	608	330	13093	0,54	39,68
955	719	19999	0,75	27,82	620	443	15122	0,71	34,14	378	272	10022	0,72	36,85	612	457	16885	0,75	36,95
627	483	12898	0,77	26,70	764	508	13466	0,66	26,51	647	474	15241	0,73	32,15	497	377	13741	0,76	36,45
482	374	10926	0,78	29,21	442	358	9842	0,81	27,49	330	259	8816	0,78	34,04	376	280	11013	0,74	39,33
1065	864	19999	0,81	23,15	885	688	20000	0,78	29,07	599	479	14530	0,80	30,33	256	201	7326	0,79	36,45
37	0	1455	0,00	-	37	0	1455	0,00	-	37	0	1455	0,00	-	37	0	1455	0,00	-
897	681	19999	0,76	29,37	577	454	13540	0,79	29,82	614	495	14883	0,81	30,07	741	558	20008	0,75	35,86
552	451	12423	0,82	27,55	284	152	6418	0,54	42,22	308	214	7932	0,69	37,07	443	310	11610	0,70	37,45
447	343	9799	0,77	28,57	897	304	10929	0,34	35,95	360	288	9343	0,80	32,44	500	305	12839	0,61	42,10
360	286	8175	0,79	28,58	358	289	8429	0,81	29,17	577	234	9571	0,41	40,90	570	244	10379	0,43	42,54
433	308	10432	0,71	33,87	553	315	9715	0,57	30,84	570	308	10695	0,54	34,72	462	339	12209	0,73	36,01
726	592	15570	0,82	26,30	424	316	10610	0,75	33,58	784	521	18406	0,66	35,33	404	314	11712	0,78	37,30
711	582	14267	0,82	24,51	900	529	16187	0,59	30,60	548	418	13442	0,76	32,16	726	505	19674	0,70	38,96
295	235	6628	0,80	28,20	363	221	6403	0,61	28,97	716	206	7767	0,29	37,70	592	209	8459	0,35	40,47
742	575	16012	0,77	27,85	447	346	11061	0,77	31,97	761	623	17426	0,82	27,97	752	564	19726	0,75	34,98
37	0	1451	0,00	-	37	0	1451	0,00	-	37	0	1451	0,00	-	37	0	1451	0,00	-
596	458	13631	0,77	29,76	801	439	13212	0,55	30,10	830	434	14651	0,52	33,76	707	424	17065	0,60	40,25
220	168	5069	0,76	30,17	475	143	5341	0,30	37,35	381	145	5967	0,38	41,15	658	147	5887	0,22	40,05

En examinant les trois tableaux, il apparaît clairement que pour toutes les dimensions NOMAD_1 est l'algorithme générant le plus grand nombre d'itérations à succès, et ce, de manière significative. L'hypothèse selon laquelle l'utilisation d'un plus grand nombre de cœurs de calcul dans le solveur permettrait de produire un meilleur modèle quadratique, en l'enrichissant avec davantage de points, ne semble donc pas se confirmer.

De plus, il est notable de remarquer que NOMAD_1 présente la plus faible valeur de r_2, ce qui signifie qu'en plus de générer le plus d'itérations à succès, il nécessite également le moins d'évaluations pour générer une itération à succès.

CHAPITRE 6 CONCLUSION

6.1 Synthèse des travaux

Les travaux réalisés dans ce mémoire permettent d'abord de mettre en évidence, de manière numérique, l'apport des variantes parallèles de `MADS`. En effet, l'algorithme `COOP-MADS` constitue une option intéressante pour les utilisateurs disposant d'un environnement parallèle et cherchant à résoudre des problèmes d'optimisation de type `BBO` en petite dimension. Par ailleurs, ces travaux ont également souligné l'importance de recourir à un algorithme parallèle asynchrone dans les situations où les boîtes noires ont des temps de calcul hétérogènes. Enfin, la version `pMADS` a démontré une performance de mise à l'échelle efficace en exploitant jusqu'à 64 cœurs de calcul. Ce premier volet de recherche a conduit à des présentations dans deux conférences :

1. *Parallel Versions of the Mesh Adaptive Direct Search Methods*, 25$^{\text{th}}$ International Symposium on Mathematical Programming, Montréal, QC, Canada, Juillet 2024 ;
2. *Parallel Versions of the Mesh Adaptive Direct Search Methods*, Journée de l'Optimisation, Montréal, QC, Canada, Mai 2024.

De plus un article est en cours d'écriture :

1. **Samuel Mendoza**, Sébastien Le Digabel, et Antoine Lesage-Landry. *Parallel Versions of the Mesh Adaptive Direct Search Methods*, Optimization Letters. À soumettre.

La seconde partie de ce mémoire consacré à l'analyse de la répartition computationnelle entre une boîte noire et le solveur d'optimisation, a permis d'amorcer une réflexion sur un sujet peu étudié. Les résultats de la modélisation indiquent qu'un solveur qui n'exploite pas le parallélisme, lorsque la boîte noire est massivement parallèle, constitue l'approche la plus judicieuse en termes de temps de calcul. Les résultats numériques obtenus sur des problèmes analytiques ont révélé le caractère imprévisible induit par le parallélisme avec `MADS`, rendant difficile l'établissement de raisonnements systématiques quant à son effet. Les résultats numériques obtenus sur un problème réaliste ont permis de mettre en évidence une tendance : plus un algorithme alloue de ressources de calcul au solveur, meilleures sont ses performances. Toutefois, la généralisation de cette tendance reste à confirmer, car l'échantillon de problèmes tests utilisé se limitait à un seul problème. Dernièrement, les test numériques réalisés sur les modèles quadratiques de la `POLL` ont révélé que les performances de ces derniers ne sont pas corrélées avec le degré de parallélisme du solveur NOMAD.

6.2 Limitations de la solution proposée

La principale limitation réside dans le fait que l'on n'a jamais pu bénéficier d'un contexte de calcul haute performance avec l'accès à une boîte noire massivement parallèle et des ressources de calcul conséquentes. On a seulement pu créer une situation avec un faible degré de parallélisme étant bien différente de la réalité en recherche scientifique. Par ailleurs, comme mentionné au chapitre 2, le portage de codes de calcul sur `GPU` vient modifier significativement le contexte, alors que le mémoire s'inscrit dans un cadre basé sur une architecture `CPU`.

6.3 Améliorations futures

Les prochains travaux pourraient consister à intégrer un modèle probabiliste, tel que l'optimisation bayésienne, qui gère le compromis entre exploitation et exploration. En effet, l'exploitation consiste à allouer toutes les ressources de calcul à la boîte noire, tandis que l'exploration implique de dédier une partie des ressources au solveur.

Il serait également pertinent d'étudier l'impact du parallélisme sur la gestion des contraintes via la barrière progressive, car la mise à jour du paramètre $h^k_{\max}$ dépend du nombre de points évalués jusqu'à la fin de l'itération k étant influencé par le parallélisme. La façon dont la valeur de $h^k_{\max}$ est mise à jour influence les performances de la PB, d'où l'importance de mesurer l'impact du parallélisme sur celle-ci.

Enfin, au regard des excellentes performances de `COOP-MADS`, trois niveaux de parallélisme sont désormais envisagés, puisque le solveur NOMAD intègre deux niveaux (le nombre d'instances de `MADS` et la file d'évaluation de chacune des instances de `MADS`) et la boîte noire un niveau. Il devient ainsi possible d'exploiter davantage la puissance de calcul tout en garantissant de bonnes solutions. Nous proposons donc d'utiliser cette approche pour de prochains tests numériques, en particulier dans un environnement de calcul haute performance où un grand nombre d'unités de calcul sont disponibles.

RÉFÉRENCES

[1] S. Kojtych, "Contributions à l'optimisation de systèmes mécaniques non réguliers : reconception d'aubes de compresseur," Thèse de doctorat, Polytechnique Montréal, 2022. https://publications.polymtl.ca/10393/

[2] N. Andrés-Thió *et al.*, "solar : A solar thermal power plant simulator for blackbox optimization benchmarking," 2024. https://arxiv.org/abs/2406.00140

[3] C. Audet et W. Hare, *Derivative-Free and Blackbox Optimization*, ser. Springer Series in Operations Research and Financial Engineering. Cham, Switzerland : Springer, 2017. https://dx.doi.org/10.1007/978-3-319-68913-5

[4] S. Alarie *et al.*, "Two decades of blackbox optimization applications," *EURO Journal on Computational Optimization*, vol. 9, p. 100011, 2021. https://doi.org/10.1016/j.ejco.2021.100011

[5] C. Audet et J. Dennis, Jr., "Mesh Adaptive Direct Search Algorithms for Constrained Optimization," *SIAM Journal on Optimization*, vol. 17, n°. 1, p. 188–217, 2006. https://dx.doi.org/10.1137/040603371

[6] C. Audet *et al.*, "Algorithm 1027 : NOMAD version 4 : Nonlinear optimization with the MADS algorithm," *ACM Transactions on Mathematical Software*, vol. 48, n°. 3, p. 35 :1–35 :22, 2022. https://dx.doi.org/10.1145/3544489

[7] A. R. Conn, K. Scheinberg et L. N. Vicente, *Introduction to Derivative-Free Optimization*, ser. MOS-SIAM Series on Optimization. SIAM, 2009. https://dx.doi.org/10.1137/1.9780898718768

[8] C. Audet, C.-K. Dang et D. Orban, "Efficient use of parallelism in algorithmic parameter optimization applications," *Optimization Letters*, vol. 7, n°. 3, p. 421–433, Mar 2013. https://doi.org/10.1007/s11590-011-0428-6

[9] J. L. Hennessy et D. A. Patterson, *Computer Architecture, Sixth Edition : A Quantitative Approach*, 6e éd. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2017.

[10] D. A. Patterson et J. L. Hennessy, *Computer Organization and Design RISC-V Edition : The Hardware Software Interface*, 1er éd. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2017.

[11] P. Chawan *et al.*, "Parallel computer architectural schemes," *International Journal of Engineering Research & Technology (IJERT)*, vol. 1, 11 2012.

[12] M. Flynn, "Very high-speed computing systems," *Proceedings of the IEEE*, vol. 54, n°. 12, p. 1901–1909, 1966.

[13] M. J. Flynn, “Some computer organizations and their effectiveness,” *IEEE Transactions on Computers*, vol. C-21, n°. 9, p. 948–960, 1972.

[14] R. Duncan, “A survey of parallel computer architectures,” *Computer*, vol. 23, p. 5–16, 1990. https://api.semanticscholar.org/CorpusID:15036692

[15] W. J. Watson, “The ti asc : a highly modular and flexible super computer architecture,” dans *Proceedings of the December 5-7, 1972, Fall Joint Computer Conference, Part I*, ser. AFIPS ’72 (Fall, part I). New York, NY, USA : Association for Computing Machinery, 1972, p. 221–228. https://doi.org/10.1145/1479992.1480022

[16] W. C. Hohn et P. D. Jones, “The control data star-100 paging station,” dans *Proceedings of the June 4-8, 1973, National Computer Conference and Exposition*, ser. AFIPS ’73. New York, NY, USA : Association for Computing Machinery, 1973, p. 421–426. https://doi.org/10.1145/1499586.1499691

[17] R. M. Russell, “The cray-1 computer system,” *Commun. ACM*, vol. 21, n°. 1, p. 63–72, janv. 1978. https://doi.org/10.1145/359327.359336

[18] A. Tanenbaum et T. Austin, *Structured Computer Organization*. Pearson, 2013. https://books.google.ca/books?id=m0HHygAACAAJ

[19] S. Atchley *et al.*, “Frontier : Exploring exascale,” dans *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC ’23. New York, NY, USA : Association for Computing Machinery, 2023. https://doi.org/10.1145/3581784.3607089

[20] G. M. Amdahl, “Validity of the single processor approach to achieving large scale computing capabilities,” dans *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, ser. AFIPS ’67 (Spring). New York, NY, USA : Association for Computing Machinery, 1967, p. 483–485. https://doi.org/10.1145/1465482.1465560

[21] J. L. Gustafson, “Reevaluating amdahl’s law,” *Commun. ACM*, vol. 31, n°. 5, p. 532–533, mai 1988. https://doi.org/10.1145/42411.42415

[22] V. Dolean, P. Jolivet et F. Nataf, *An Introduction to Domain Decomposition Methods : Algorithms, Theory, and Parallel Implementation*. USA : Society for Industrial and Applied Mathematics, 2015.

[23] J. Andrej *et al.*, “High-performance finite elements with mfem,” *The International Journal of High Performance Computing Applications*, vol. 38, n°. 5, p. 447–467, 2024. https://doi.org/10.1177/10943420241261981

[24] R. Anderson *et al.*, “Mfem : A modular finite element methods library,” *Computers & Mathematics with Applications*, vol. 81, p. 42–74, 2021, development

and Application of Open-source Software for Problems with Numerical PDEs. https://www.sciencedirect.com/science/article/pii/S0898122120302583

[25] P.-M. Olsson, "Methods for Network Optimization and Parallel Derivative-Free Optimization," Thèse de doctorat, Linköping University, 2014. https://www.diva-portal.org/smash/get/diva2:695431/FULLTEXT02.pdf

[26] G. Schryen, "Parallel computational optimization in operations research : A new integrative framework, literature review and research directions," *European Journal of Operational Research*, vol. 287, nº. 1, p. 1–18, 2020. https://doi.org/10.1016/j.ejor.2019.11.033

[27] S. Valcke, "The oasis3 coupler : a european climate modelling community software," *Geoscientific Model Development*, vol. 6, nº. 2, p. 373–388, 2013. https://gmd.copernicus.org/articles/6/373/2013/

[28] M. C. Ferris et O. L. Mangasarian, "Parallel variable distribution," *SIAM Journal on Optimization*, vol. 4, nº. 4, p. 815–832, 1994. https://doi.org/10.1137/0804047

[29] M. Asgari *et al.*, "A review of parallel computing applications in calibrating watershed hydrologic models," *Environ. Model. Softw.*, vol. 151, p. 105370, 2022. https://doi.org/10.1016/j.envsoft.2022.105370

[30] E. Alba, G. Luque et S. Nesmachnow, "Parallel metaheuristics : recent advances and new trends," *Int. Trans. Oper. Res.*, vol. 20, p. 1–48, 2012. https://api.semanticscholar.org/CorpusID:16286419

[31] D. Dasgupta et Z. Michalewicz, *Evolutionary algorithms in engineering applications*. Springer Science & Business Media, 2013.

[32] N. Hansen, "The CMA evolution strategy : A tutorial," *CoRR*, vol. abs/1604.00772, 2016. http://arxiv.org/abs/1604.00772

[33] D. Redon *et al.*, "Massively parallel cma-es with increasing population," 2024. https://arxiv.org/abs/2409.11765

[34] D. Monroe, "Fugaku takes the lead," *Commun. ACM*, vol. 64, nº. 1, p. 16–18, déc. 2020. https://doi.org/10.1145/3433954

[35] A. R. Conn, N. I. M. Gould et P. L. Toint, *Trust Region Methods*. Society for Industrial and Applied Mathematics, 2000. https://epubs.siam.org/doi/abs/10.1137/1.9780898719857

[36] R. Hooke et T. A. Jeeves, "" direct search" solution of numerical and statistical problems," *J. ACM*, vol. 8, nº. 2, p. 212–229, avr. 1961. https://doi.org/10.1145/321062.321069

[37] E. Fermi, “Numerical solution of a minimum problem,” Los Alamos Scientific Lab., Los Alamos, NM, Rapport technique, 1952.

[38] V. Torczon, “On the convergence of pattern search algorithms,” *SIAM Journal on Optimization*, vol. 7, n°. 1, p. 1–25, 1997. https://doi.org/10.1137/S1052623493250780

[39] C. Audet, S. Le Digabel et C. Tribes, “The mesh adaptive direct search algorithm for granular and discrete variables,” *SIAM Journal on Optimization*, vol. 29, n°. 2, p. 1164–1189, 2019. https://doi.org/10.1137/18M1175872

[40] M. A. Abramson *et al.*, “Orthomads : A deterministic mads instance with orthogonal directions,” *SIAM J. Optim.*, vol. 20, p. 948–966, 2008. https://api.semanticscholar.org/CorpusID:12739979

[41] A. S. Householder, “Unitary triangularization of a nonsymmetric matrix,” *Journal of the ACM (JACM)*, vol. 5, n°. 4, p. 339–342, 1958.

[42] S. Le Digabel et S. Wild, “A taxonomy of constraints in black-box simulation-based optimization,” *Optimization and Engineering*, vol. 25, n°. 2, p. 1125–1143, 2024. https://dx.doi.org/10.1007/s11081-023-09839-3

[43] A. Booker *et al.*, “Optimization using surrogate objectives on a helicopter test example,” dans *Optimal Design and Control*, ser. Progress in Systems and Control Theory, J. Borggaard *et al.*, édit. Cambridge, Massachusetts : Birkhäuser, 1998, p. 49–58. http://www.birkhauser.com/detail.tpl?ISBN=0817640649

[44] ——, “Managing surrogate objectives to optimize a helicopter rotor design – further experiments,” AIAA Paper 1998–4717, Presented at the 8th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis, 1998.

[45] S. Le Digabel, “Algorithm 909 : NOMAD : Nonlinear Optimization with the MADS algorithm,” *ACM Transactions on Mathematical Software*, vol. 37, n°. 4, p. 44 :1–44 :15, 2011. https://dx.doi.org/10.1145/1916461.1916468

[46] C. Audet, J. Dennis, Jr. et S. Le Digabel, “Parallel Space Decomposition of the Mesh Adaptive Direct Search Algorithm,” *SIAM Journal on Optimization*, vol. 19, n°. 3, p. 1150–1170, 2008. https://dx.doi.org/10.1137/070707518

[47] M. Snir *et al.*, *MPI : The Complete Reference*. Cambridge, Massachusetts : The MIT Press, 1995.

[48] L. Sarrazin-Mc Cann, “Optimisation et ordonnancement en optimisation sans dérivées,” Mémoire de maîtrise, Polytechnique Montréal, 2018. https://publications.polymtl.ca/3099/

[49] T. Lunet, “Stratégies de parallélisation espace-temps pour la simulation numérique des écoulements turbulents,” Thèse de doctorat, Université de Toulouse, 2018, thèse

de doctorat dirigée par Gratton, Serge et Bodart, Julien Mathématiques appliquées Toulouse, ISAE 2018. http://www.theses.fr/2018ESAE0001

[50] M. Hamedi, "Gradient-free aeroacoustic shape optimization using high-order implicit large eddy simulation," Thèse de doctorat, Concordia University, January 2024, unpublished. https://spectrum.library.concordia.ca/id/eprint/993433/

[51] M. A. Abramson *et al.*, "An efficient class of direct search surrogate methods for solving expensive optimization problems with cpu-time-related functions," *Structural and Multidisciplinary Optimization*, vol. 45, nº. 1, p. 53–64, Jan 2012. https://doi.org/10.1007/s00158-011-0658-3

[52] V. Beiranvand, W. Hare et Y. Lucet, "Best practices for comparing optimization algorithms," *Optimization and Engineering*, vol. 18, p. 815 – 848, 2017. https://api.semanticscholar.org/CorpusID:67069800

[53] A. Bagirov, N. Karmitsa et M. M. Mkel, *Introduction to Nonsmooth Optimization : Theory, Practice and Software*. Springer Publishing Company, Incorporated, 2014.

[54] N. Karmitsa, "Test problems for large-scale nonsmooth minimization," Department of Mathematical Information Technology, University of Jyväskylä, Jyväskylä, Finland, Technical report B. 4/2007, 2007.

[55] J. Moré et S. Wild, "Benchmarking Derivative-Free Optimization Algorithms," *SIAM Journal on Optimization*, vol. 20, nº. 1, p. 172–191, 2009. https://dx.doi.org/10.1137/080724083

[56] J. Zhao et N. Wang, "A bio-inspired algorithm based on membrane computing and its application to gasoline blending scheduling," *Computers and Chemical Engineering*, vol. 35, nº. 2, p. 272–283, 2011. http://dx.doi.org/10.1016/j.compchemeng.2010.01.008

[57] C. Audet et J. Dennis, Jr., "A Progressive Barrier for Derivative-Free Nonlinear Programming," *SIAM Journal on Optimization*, vol. 20, nº. 1, p. 445–472, 2009. https://dx.doi.org/10.1137/070692662

[58] L. Lukšan et J. Vlček, "Test Problems for Nonsmooth Unconstrained and Linearly Constrained Optimization," ICS AS CR, Rapport technique V-798, 2000.

[59] J. Tao et N. Wang, "DNA Double Helix Based Hybrid GA for the Gasoline Blending Recipe Optimization Problem," *Chemical Engineering and Technology*, vol. 31, nº. 3, p. 440–451, 2008. https://dx.doi.org/10.1002/ceat.200700322

[60] J. Zhao et N. Wang, "A bio-inspired algorithm based on membrane computing and its application to gasoline blending scheduling," *Computers and Chemical Engineering*, vol. 35, nº. 2, p. 272–283, 2011. https://dx.doi.org/10.1016/j.compchemeng.2010.01.008