POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Sondes locales intensives lors de l'exécution de l'algorithme `MADS` dans un environnement parallèle

GUILLAUME LAMEYNARDIE
Département de mathématiques et génie industriel

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

Août 2020

© Guillaume Lameynardie, 2020.

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Sondes locales intensives lors de l'exécution de l'algorithme `MADS` dans un environnement parallèle

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

Louis-Martin ROUSEAU, président
Charles AUDET, membre et directeur de recherche
Sébastien LE DIGABEL, membre et codirecteur de recherche
Amira DEMS, membre externe

DÉDICACE

À tous mes amis du labos,
vous me manquerez...

REMERCIEMENTS

Texte / Text.

RÉSUMÉ

L'optimisation de boîte noire consiste en la résolution d'un problème $(\mathcal{P}) : \min\{f(x) : x \in \Omega\}$ pour lequel les valeurs de la fonction objectif et les contraintes sont le résultat de l'exécution d'une simulation numérique parfois coûteuse en temps ou en ressources. Lors de son étape de sonde, l'algorithme `MADS` évalue la fonction objectif et les contraintes autour du meilleur point connu suivant des directions qui forment un ensemble générateur positif de l'espace des solutions. Dans un soucis de performance, le logiciel `NOMAD` 4 qui implémente `MADS`, utilise une base positive i.e. un ensemble générateur positif de cardinalité minimale. Pour un problème de dimension n, cette base positive est constituée de $2n$ directions orthogonales, donc chaque étape de sonde conduit à $2n$ évaluations de l'objectif et des contraintes. L'accès aux infrastructures à fort degré de parallélisme est de plus en plus facile. Par exemple, le centre de recherche d'Hydro-Québec (IREQ) possède une grille de calcul (CASIR) à 152 noeuds, et 36 CPUs par noeud. Dans ce contexte, lors de l'étape de sonde locale, l'exécution de `NOMAD` 4 conduit à une sous-utilisation des ressources à disposition. En effet, le nombre d'évaluations à effectuer est beucoup plus petit que le nombre de processeurs disponibles. Il semble alors pertinent d'évaluer l'objectif et les contraintes suivant un plus grand nombre de directions.

Ce mémoire étudie l'influence de l'augmentation du nombre de directions de sonde, ainsi que celle de la géométrie donnée à ces directions.

Un critère d'intensification est établi pour augmenter le nombre de directions uniquement lorsque cela semble nécessaire. Ce critère a pour but d'éviter de fournir un grand nombre d'évaluations aux processeurs lorsqu'il semble relativement facile d'améliorer l'objectif et les contraintes. Quatre variantes du type d'intensification dynamique sont formulées.

Un cadre de travail est proposé pour répartir les évaluations fournies par les stratégies de sonde sur un ensemble de processeurs lorsque des fonctions oracles sont rendues disponibles pour connaître le temps ou le travail nécessaire à chaque évaluation.

Enfin, des essais numériques sont menés sur plusieurs problèmes analytiques sans contraintes, et sur un problème réaliste issu de la littérature. Les données obtenues sont interprétées à l'aide de profils de données en évaluations et en itérations. Le premier type de profil permet de juger des performances des stratégies implémentées selon le nombre d'appels à la boite noire. Le second type de profil permet d'estimer les performances des stratégies selon le nombre de cycles d'occupation de ressources parallèles.

Les résultats obtenus montrent que la géométrie donnée aux directions de sonde importe peu, seul le nombre de directions influe sur les performances de la résolution de $\mathcal{P}$. L'intensification dynamique permet d'économiser des évaluations, et dans certains cas, d'obtenir des résultats semblables au cas où le nombre de points générés est constant d'une itération à l'autre.

ABSTRACT

Blackbox optimization typically arises when the objectif function and constraints of a problem $(\mathcal{P}) : \min\{f(x) : x \in \Omega\}$ are obtained by executing a numerical simulation that may be time-consuming or expensive in terms of ressources needed to be completed. During the poll step of the `MADS` algorithm, f and c_j are evaluated around the best incubent following directions of a positive spanning set of the search space. As a performance issue, the software `NOMAD` 4 that implements `MADS`, uses a positive basis, which is a positive spanning set of minimal cardinality. For a problem of dimension n, this positive basis is made of $2n$ orthogonal directions and so the poll step leads to $2n$ evaluations of f and c_j. Nowadays, access to massively parallel ressources is easier. For instance, the research laboratory of Hydro-Québec (IREQ) owns a supercomputer made of 152 nodes and 36 CPUs per node. In this context, at the poll step, the execution of `NOMAD` 4 leads to an underuse of the available ressources. Indeed, the number of evaluations is far lower than the number of available processors. It seems relevant to evaluate f and c_j in a bigger number of polling directions.

This study focuses on the influence of increasing the number of polling directions and the geometry used to build them.

A criterion on increasing the number of polling directions is set up in order to feed processors with more evaluations only when it seems to be relevant. Hence, one avoid generating too much evaluations when improving the objective function seems to be easy. Four different kind of dynamic intensification are formulated.

A load balancing framework is described to order evaluations on a set of processors based on oracle functions providing information on the work or time needed to evaluate the objectif ent the constraints on a point.

Numerical results on several analytical problems without constraints, and one real world problem from the litterature are presented to draw the behaviour of the poll strategies. The data obtained are post-processed with data profile in terms of evaluations and iterations. The first kind of profil allows one to see the performance of each strategies in term of number of calls to the blackbox while the second kind of profils gives the performance results in term of parallel ressources occupation cycles.

Regarding the results obtained, the geometry given to the poll directions set seems to have neglictable effect on the runs. The factor with the most noticable influence is the number of polling directions generated. In some cases, dynamic intensification of the number of polling

directions provides nearly the same performance in improving f but with less evaluations than when a constant number of points is generated trough iterations.

TABLE DES MATIÈRES

DÉDICACE iii

REMERCIEMENTS iv

RÉSUMÉ v

ABSTRACT vii

TABLE DES MATIÈRES ix

LISTE DES TABLEAUX xi

LISTE DES FIGURES xii

LISTE DES SIGLES ET ABRÉVIATIONS xiv

CHAPITRE 1 INTRODUCTION 1
1.1 L'optimisation de boîte noire 1
1.2 Motivations et organisation 3

CHAPITRE 2 Revue de littérature 5
2.1 Différentes méthodes d'optimisation de boite noire 5
2.2 L'algorithme `MADS` 8
2.3 Utilisation de ressources parallèles dans l'optimisation de boite noire 15
2.4 Approche informatique de la programmation sur des ressources parallèles . . 18
2.5 Métriques de performance pour algorithmes et programmes parallèles 22
2.6 Parallélisme dans le logiciel `NOMAD` 4 26
2.7 Occupation d'un grand nombre de processeurs lors de l'étape de recherche globale 28

CHAPITRE 3 Stratégies de `POLL` intensives 33
3.1 Cadre de travail pour l'étape de `POLL` 33
3.2 Génération de points lors de l'étape de `POLL` 35
3.2.1 Multi-`POLL` 35
3.2.2 Sonde locale en Oignon 36
3.2.3 Sonde locale Enrichie 38

3.3 Intensification dynamique lors de l'étape de POLL 40
3.4 Ordonnancement optimal de la pile d'évaluations 41

CHAPITRE 4 Résultats théoriques et numériques 46
4.1 Métriques de performance 46
4.2 Description de l'ensemble de problèmes 49
4.2.1 Problèmes analytiques 49
4.2.2 Problèmes réalistes 50
4.3 Résultats numériques obtenus lors de l'exécution sur les problèmes analytique 50
4.3.1 Détails des valeurs des paramètres algorithmiques et de l'environnement numérique de tests 50
4.3.2 Intérêt de l'exploration de l'intérieur du cadre 51
4.3.3 Influence du nombre de points et de la géométrie sur la valeur optimale 53
4.3.4 Influence de l'intensification 56
4.4 Résultats numériques obtenus lors de la résolution d'un cas réaliste 63
4.4.1 Détails des valeurs des paramètres algorithmiques et de l'environnement numérique de tests 63
4.4.2 Comportement des stratégies d'intensification seules 64
4.4.3 Ajout de mécanismes d'aide à la résolution 65
4.4.4 Profils de données avec et sans mécanismes d'aide à l'optimisation . . 68
4.5 Discussion 69

CHAPITRE 5 Conclusion 71
5.1 Synthèse des travaux 71
5.2 Limites de la solution proposée 71
5.3 Améliorations et travaux futures 71

RÉFÉRENCES 73

LISTE DES TABLEAUX

4.1 Données finales obtenues pour les différentes stratégies utilisées sur `STYRENE` à partir de $x^0 = (54, 66, 86, 08, 29, 51, 32, 15)$. 67

LISTE DES FIGURES

1.1 Lien entre boîte noire, variables, fonction objectif et contraintes. f et les c_j sont des fonctions. Leurs valeurs pour un jeu de variables donné est obtenue par l'exécution de la boite noire. 1

2.1 Classification des méthodes existantes en optimisation de boîte noire 5

2.2 Graphe de dépendances de contrôle et de données de `MADS` avec évaluations en parallèle non opportuniste. 10

2.3 Si l'étape de `SEARCH` (gauche) ne parvient pas à améliorer la valeur de la fonction objectif, l'étape de `POLL` (droite) est exécutée. 11

2.4 Un échec de l'étape de `POLL` conduit à une réduction de δ^k et de Δ^k. 11

2.5 (a) une suite d'évaluations en séquentielle, (b) une suite d'évaluations parallèle avec moins de processeurs que d'évaluations à effectuer, (c) une suite d'évaluations avec plus de processeurs que d'évaluation à effectuer. 17

2.6 (a)-Exécution concurrente et parallèle de trois fils d'exécution vs. (b)-Exécution concurrente non parallèle. Inspiré de Tremblay [44] 20

2.7 Courbe illustrant l'effet général de la taille des grains (des tâches) sur le temps d'exécution. Inspiré de Tremblay [44] 21

2.8 Nuances entre fils d'exécution physiques, fils d'exécution logiciel et pile d'évaluations. 27

3.1 Comparaison entre `POLL` classique et Multi `POLL` avec le même nombre de directions dans chacune des sondes secondaire : $q = q_1 = q_2 = q_3 = 3$, et $\Delta^{k,1} = 4\delta^k, \Delta^{k,2} = \Delta^{k,3} = 3\delta^k$ 36

3.2 Des directions sont construites sur les cadres de sondes correspondant à deux réductions successives de Δ^k. 37

3.3 De gauche à droite : `POLL` classique, `POLL` enrichie sur le cadre, poll enrichie sur et à l'intérieur du cadre. 39

4.1 Profils de donnée en évaluation de la stratégie Enrichie lorsque les points de sonde sont uniquement sur le cadre ou aussi à l'intérieur du cadre, toutes dimensions confondues. 52

4.2 Profils de donnée en évaluation de la stratégie Enrichie en dimension 8 lorsque les points de sonde sont uniquement sur le cadre ou aussi à l'intérieur du cadre. 53

4.3 Influence du nombre de points sur la moyenne (cercle) et le maximum (triangle) des valeurs finales f^*. Stratégies statiques. 54
4.4 Profils de données (a,b) et profils de performances (c,d) en évaluation toutes dimensions confondues : Influence de la géométrie de génération de points. 55
4.5 Profils de données (a,b) et profils de performance (c,d) en itérations toutes dimensions confondues : Influence de la géométrie de génération de points. 56
4.6 Profils de donnée en évaluation et en itération des stratégies Oignon (a, b) et Enrichie (c,d) 58
4.7 Profils de données en évaluations de la stratégie Enrichie pour différentes dimensions. 60
4.8 Profils de données en itérations de la stratégie Enrichie pour différentes dimensions. 61
4.9 Profils de données (a,b) et de performance (c,d) en itérations de la POLL Enrichie avec $n_p^{max} \in \{8 \times 2n, 64 \times 2n\}$, et différents types d'intensification, toutes dimensions confondues. 62
4.10 Graphe de convergence lors de la résolution de STYRENE pour les stratégies Oignon et Enrichie avec différents types d'intensification seules. Les graphes sont en évaluation (a,b), en itérations (c,d), $n = 8$, $n_p^{max} = 64 \times 2n$ 65
4.11 Graphe de convergence lors de la résolution de STYRENE pour les stratégies Oignon et Enrichie avec différents types d'intensification, munies des mécanismes d'aide à la résolution. Les graphes sont en évaluation (a,b), en itérations (c,d), $n = 8$, $n_p^{max} = 64 \times 2n$ 66
4.12 Profils de données en évaluation sur STYRENE pour les stratégies Oignon et Enrichie seules (a,b), et avec les mécanismes d'aide à la résolution supplémentaires (c,d). 68
4.13 Profils de données en itérations sur STYRENE pour les stratégies Oignon et Enrichie seules (a,b), et avec les mécanismes d'aide à la résolution supplémentaires (c,d). 69

LISTE DES SIGLES ET ABRÉVIATIONS

MADS	Mesh Adaptive Direct Search
NOMAD	Nonlinear Optimization with the MADS Algorithm
CS	Coordinate Search
GPS	Generalized Pattern Search
TR	Trust Region

CHAPITRE 1 INTRODUCTION

L'optimisation consiste en l'amélioration d'une quantité d'intérêt dépendant de paramètres, sous le respect d'un certain nombre, possiblement nul, de contraintes. Selon ce que modélise la quantité d'intérêt, les paramètres et les contraintes, les propriétés de ces objets mathématiques donnent lieu à des regroupement en branches de l'optimisation. Par exemple, l'étude de l'amélioration d'une quantité d'intérêt dont tout ou partie des paramètres sont entiers, est appelée optimisation en nombre entiers. L'optimisation linéaire traite le cas où la quantité d'intérêt et les contraintes sont des combinaisons linéaires des paramètres. Ces branches de l'optimisation, bien qu'ayant le même but, ont des approches radicalement différentes.

1.1 L'optimisation de boîte noire

L'optimisation de boîte noire est une branche de l'optimisation dans laquelle la fonction objectif et les contraintes ne sont pas connues de manière analytique : l'utilisateur n'a pas accès à un lien explicite entre les valeurs d'entrée et les valeurs de sortie. En effet, les valeurs de sorties peuvent être le résultat d'un processus dont le fonctionnement est inconnu, car il est trop complexe, ou tout simplement inaccessible. En pratique, une boîte noire est un processus de nature quelconque, par exemple, une expérience de laboratoire, une simulation numérique, etc, dont la complexité ou l'opacité reflète la difficulté d'établir un lien explicite entre l'entrée et la sortie [11]. Ce processus est paramétré par $n \in \mathbb{N}^*$ variables $x = (x_1, x_2, \ldots, x_i, \ldots, x_n)$ et retourne une liste de sorties, parmi lesquelles se trouve une valeur d'intérêt que l'on cherche à améliorer (maximiser ou minimiser) ainsi que $m \in \mathbb{N}$ contraintes, comme le représente la figure 1.1. Souvent, ce retour peut être obtenu après des minutes voir des heures de fonctionnement

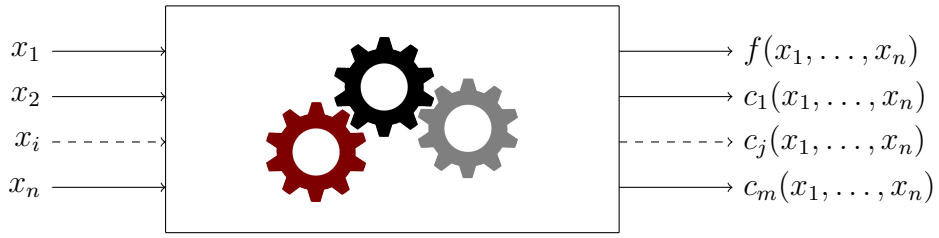

Figure 1.1 Lien entre boîte noire, variables, fonction objectif et contraintes. f et les c_j sont des fonctions. Leurs valeurs pour un jeu de variables donné est obtenue par l'exécution de la boite noire.

Dans la suite on fixe comme convention que l'on cherche à minimiser la valeur d'intérêt, tout en ayant des valeurs de contraintes négatives ou nulles. Cela s'écrit :

$$(\mathcal{P}) : \begin{cases} \min\limits_{x \in \mathcal{X}} & f(x) \\ \text{s.c} & c_j(x) \leq 0, \quad j \in [\![1, m]\!]. \end{cases} \tag{1.1}$$

Avec :

- $\mathcal{X}$ est l'ensemble des points réalisables pour f : $\mathcal{X}$ est formé par les contraintes de bornes. Par exemple, si x_1 représente l'épaisseur d'une aile d'avion, alors : $0 \leq x_1 \leq d$, où d est le diamètre de la cabine, si c'est la seule contrainte, alors $\mathcal{X} = \{x \in \mathbb{R}^n, 0 \leq x_1 \leq d\}$.)
- $f : \mathcal{X} \subset \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\} = \overline{\mathbb{R}}$. En effet, même si les sorties de la boîte noire représentent les contraintes, il se peut que cette liste ne soit pas exhaustive, et, que pour certains paramètres d'entrée, la valeur de la quantité d'intérêt renvoyé lors de l'exécution de la boîte noire n'ait aucun sens, même si les valeurs fournies en entrée paraissent acceptables. Par exemple dans le cas d'une simulation numérique, il se peut que pour certains paramètres, un comportement inattendu de la simulation numérique ait lieu, comportement non anticipé par le concepteur de la boîte noire, et qui ne peut être interprété que par une valeur infinie d'un point de vue mathématique, car ne donnant aucune information exploitable.
- Pour $j \in [\![1, m]\!]$, $c_j : \mathcal{X} \subset \mathbb{R}^n \to \overline{\mathbb{R}}$. Pour les mêmes raisons que dans le point précédent, les contraintes peuvent prendre des valeurs infinies.
- Toute information en lien avec la notion des dérivées de f et des c_j est considérée comme inaccessible et inutilisable. Leur estimation n'est pas considérée comme pertinente, en raison d'aucune garantie de leur existence, de la pertinence de l'information qu'elles délivrent, et du coût de l'estimation.
- Une évaluation de f et des c_j est coûteuse en temps et en ressources informatiques, logistiques, car cela nécessite l'exécution du processus que délimite la boite noire.
- f et les c_j sont déterministes : fournir deux fois la même entrée fournit deux fois la même sortie. [1]

L'ensemble des points réalisables est définit par :

$$\Omega = \{x \in \mathcal{X} : \forall j \in [\![1, m]\!], c_j(x) \leq 0\}. \tag{1.2}$$

1. Il existe des problèmes industriels faisant intervenir des composantes non déterministes, ayant pour effet de rendre f et les c_j non déterministes. C'est le cas par exemple lorsque la méthode de Monte-Carlo est utilisée pour obtenir une série de données sur laquelle qui servent ensuite à des calculs.

Comme vu plus haut, même si $x \in \Omega$, f peut prendre la valeur $+\infty$ en x car Ω ne prend pas en compte les contraintes cachées dans sa définition.

1.2 Motivations et organisation

Aucune hypothèse n'a été faite sur le nombre n de variables du problème $\mathcal{P}$. En pratique, il existe des problèmes à moins de dix variables, tout comme d'autres qui en possèdent plusieurs centaines. De plus, il est possible de mener plusieurs exécutions de la boîte noire en parallèle, par exemple lorsque la boîte noire est une simulation numérique, que l'utilisateur dispose de ressources informatiques suffisantes, ou lorsque la boîte noire est une expérience de laboratoire qui nécessite des ressources dont l'utilisateur dispose en plusieurs fois.

Ainsi il est possible d'étendre la notion de processeur en informatique à une notion plus générale définie comme suit :

Définition 1 (Processeur)**.** Pour une boite noire et un utilisateur donné, un *processeur* est un ensemble de ressources que l'utilisateur juge à priori capable de mener à terme n'importe laquelle des évaluations de cette boite noire en une durée inférieure à une borne fixée par ce même utilisateur.

De cette façon, un processeur peut être un téléphone portable, un banc d'essai instrumenté en laboratoire, la moitié des ressources d'un super calculateur, etc. En bref, un processeur n'est pas nécessairement un CPU.

Dans ce mémoire, Le cas où le nombre de variables du problème est petit devant le nombre de processeurs est étudié. En effet, il existe des infrastructures de calcul possédant des centaines, voir des milliers de CPUs. Par exemple, l'Institut de recherche d'Hydro-Québec (IREQ) possède une grappe de calcul (CASIR) de 152 noeuds, chacun muni de 36 CPUs. Cependant, certains des problèmes d'optimisation rencontrés par l'IREQ ont quelques dizaines de variables.

Le but de ce travail est d'étudier et de modifier l'algorithme `MADS` (Mesh Adaptive Direct Search) [9] de sorte à tirer pleinement partie d'un grand nombre de processeurs devant le nombre de variables, tant sur le plan du nombre d'itérations nécessaire à l'optimisation, donc pour satisfaire un critère de convergence donné, que sur l'amélioration de la fonction objectif. Autrement dit, on espère qu'avoir plus de processeurs à disposition va nous permettre de résoudre $\mathcal{P}$ plus rapidement ou d'obtenir une meilleure solution.

Par la suite, l'hypothèse est faire que l'utilisateur a accès à un nombre $p \in \mathbb{N}^*$ fixé de processeurs pour effectuer les différentes évaluations de f et des c_j, et que ce nombre est

grand devant le nombre de variables n du problème d'optimisation.

Dans la section 2, une revue de littérature sur les méthodes d'optimisation de boite noire et la programmation parallèle est exposée. Les regroupement par famille de méthodes y sont mentionnés, et le fonctionnement de l'algorithme `MADS` [9] est détaillé. Ensuite, les travaux sur l'utilisation de ressources parallèles dans les différentes méthodes d'optimisation de boite noire sont décrits. La description est centrée sur les méthodes de recherches directes. Puis des notions de programmation parallèle sont exposées, et le fonctionnement du parallélisme dans `NOMAD` 4 [1, 13] est détaillé. S'en suit une présentation des métriques de performances pour algorithmes parallèles. Enfin, les travaux menés pour permettre une occupation d'un grand nombre de processeurs lors de l'étape de `SEARCH` de l'algorithme `MADS` sont détaillés.

Comme expliqué dans la section 2, une itération de l'algorithme `MADS` se décompose en deux étapes : la `SEARCH` et la `POLL`. Aux vues de l'état de l'art présenté en section 2, la section 3 établit différentes stratégies envisageables pour générer des points candidats à l'évaluation lors de l'étape de `POLL`. Le but est d'utiliser pleinement les ressources disponibles lors de cette étape. Un critère d'intensification de la `POLL` est également présenté. En effet, même si de nombreuses ressources sont disponibles, il est possible qu'améliorer f ne nécessite pas l'utilisation complète de ces ressources, car l'optimisation est rendu à un stade où il est «simple» d'améliorer f. Enfin, une discussion sur l'ordonnancement du travail fournit à un ensemble de processeurs lors de l'étape de `POLL` est présentée. Un cadre de travail est établit pour que l'ordonnancement se fasse à l'aide de fonctions oracles indiquant la charge de travail ou le temps nécessaire à chaque évaluation. Ce dernier point semble pertinent lorsqu'il est possible d'estimer simplement la durée ou le coût d'une évaluation à partir de x.

La section 4 présente différentes métriques de performances applicables à `MADS` muni des stratégies décrites en section 3, ainsi que des résultats numériques obtenu à l'aide de ces métriques. Le but de cette section est de déterminer la pertinence de sonder plus intensément l'espace à chacune des étapes de `POLL`, et si tel est le cas, quel peut être le gain espéré avec les différents mécanismes et stratégies établis en section 3. Pour cela, une banque de problèmes analytique ainsi qu'un cas réaliste sont utilisés pour réaliser des essais.

CHAPITRE 2 Revue de littérature

Dans cette section, l'état de l'art en optimisation de boite noire dans un environnement parallèle est exposé. Le but de cette revue de littérature est de montrer un manque de travaux en ce qui concerne l'occupation d'un grand nombre de processeurs par rapport au nombre de variables. En effet, depuis les années 2000, les infrastructures de calcul à l'aide de ressources parallèles se sont largement répandues, ouvrant la voie vers une nouvelle utilisation des algorithmes d'optimisation. Il est alors envisageable de résoudre des problèmes de plus grande dimension, mais dans le cas des méthodes de recherche directe, les problèmes de petite dimension produisent des itérations qui n'occupent pas les ressources disponibles à leur plein potentiel.

2.1 Différentes méthodes d'optimisation de boite noire

Comme dans les autres branches de l'optimisation, des méthodes spécifiques aux caractéristiques du problème $(\mathcal{P})$ énoncées en introduction 1 ont été développées depuis les années 1950. La figure 2.1 donne une vue d'ensemble des types de méthodes existantes pour optimiser des problèmes de type boite noire.

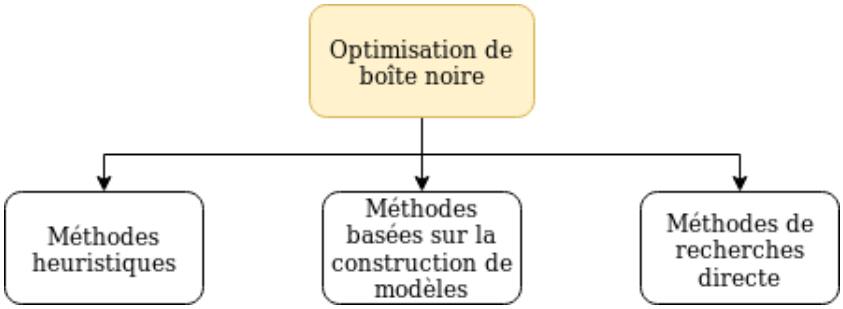

Figure 2.1 Classification des méthodes existantes en optimisation de boîte noire

Les méthodes heuristiques. Ce sont pour la plupart des méthodes évolutionnaires [16], dont le concept fondateur est basé sur la création de nouvelles solutions à partir d'anciennes estimées relativement bonne grâce aux valeurs retournées par la fonction objectif et les contraintes. Ces méthodes sont populaires du fait de leur fonctionnement qui est semblable à l'évolution et à l'adaptation du vivant dans un environnement incertain. Cependant, pour être efficace, elles nécessitent un grand nombre d'évaluations de f et des c_j, ce qui est

limitant dans le contexte boîte noire. Ainsi les méthodes évolutionnaires possèdent des propriétés de convergence intéressantes mais limitées à certains contextes. D'autres méthodes, telle que la méthode de Nelder et Mead (NM) [37] possèdent des propriétés de convergence plus adaptées au contexte boite noire car elles ne nécessitent pas autant d'évaluations qu'une méthode évolutionnaire pour retourner une solution de $\mathcal{P}$ intéressante. Mais parfois, de telles méthodes échouent sur des problèmes relativement simples : McKionn [35] exhibe des cas où la méthode NM converge vers un point non stationnaire sur plusieurs problèmes convexes en dimension 2

Les méthodes basées sur l'utilisation d'un modèle. Ces classes de méthodes se montrent intéressantes, entre autre, lorsque le problème $\mathcal{P}$ est coûteux à évaluer. Le modèle utilisé peut avoir des propriétés de régularité exploitables telles que la différentiabilité, ou être beaucoup moins coûteux à évaluer que la boite noire. Deux types de modèles se distinguent. Il existe des modèles dynamiques, construits de manière analytique par interpolation à l'aide des évaluations déjà terminées de f et des c_j. Parmi les modèles dynamiques, différentes classes existent [45]. Parfois, il se peut aussi que l'accès à des modèles statiques soit possible. Ces modèles sont des simplifications de la boîte noire, qui donnent lieu à des évaluations de f et des c_j moins coûteuse. Par exemple, si le calcul de f et des c_j fait appel à un processus itératif dans le focntionnement de la boîte noire, un modèle $\mathcal{S}$ de $\mathcal{P}$ peut être obtenu en effectuant moins d'itérations lors de l'exécution de la boite noire pour le calcul de f et des c_j. Ce type de modèle est généralement fourni par l'opérateur qui a construit la boîte noire. D'après Vu, D'Ambrosio, et Liberti [45], il existe diverses manières d'utiliser un modèle pour construire une méthode d'optimisation. Dans le cas de modèles dynamiques, l'idée fondatrice est de se servir du modèle pour identifier les régions prometteuses de l'espace, puis, après avoir évalué la boite noire dans ces régions prometteuses, mettre à jour le ou les modèles utilisés, et recommencer un nouveau cycle tant qu'un critère d'arrêt n'est pas atteint. Parmi ces méthodes, figurent entre autre, les méthodes dite de région de confiance (TR) [19]. Talgorn [40] expose une librairie de construction de modèles dynamiques de nature diverses.

Les méthodes de recherche directe. Ce sont des méthodes itératives qui évaluent f et les c_j sur une partie des noeuds d'un maillage de l'espace dans différentes directions autour de la meilleure solution connue. Suivant les résultats obtenus, le pas du maillage est réduit ou agrandit pour y effectuer de nouvelles évaluations. Une revue détaillée de ces méthodes est donnée par Audet [7]. Trois principales générations de méthodes de recherches directes ont vu le jour. La première d'entre elle, la recherche par coordonées (CS) [22] sonde l'espace suivant

les directions de la base canonique de $\mathbb{R}^n$ et leur opposés autour de la meilleure solution connue, à une certaine distance $\Delta \in \mathbb{R}_+^*$. En cas d'échec de l'amélioration de la valeur de f, Δ est réduit et cette étape de sonde est à nouveau effectuée. En cas de succès le centre de sonde est déplacé de Δ suivant la direction de succès. Cette génération s'applique principalement à l'optimisation sans contraintes. La seconde génération de méthodes de recherche directe, la recherche par motifs généralisée, (GPS) [43], introduit trois améliorations d'après Audet [7]. (1) L'ensemble des directions de sonde à chaque itération n'est plus restreint aux directions de coordonnées, mais est un sous ensemble de directions fixées en début d'optimisation. Ce mécanisme permet donc de s'échapper de zones dans lesquelles la diminution de f ne se fait pas suivant les directions de coordonnées. (2) Le pas de sonde peut être augmenté en cas de succès pour accélérer l'exploration de l'espace. (3) Une étape de recherche globale est ajoutée à chacune des itérations de l'algorithme, permettant d'évaluer des points ailleurs dans l'espace, et donc de prendre en compte le savoir de l'opérateur qui a construit le problème. La troisième génération d'algorithme, la recherche directe par treillis adaptatif (`MADS`) [9] vient compléter GPS en introduisant un paramètre de maillage $\delta \in \mathbb{R}_+^*$ en plus du paramètre de sonde $\Delta \geq \delta$. Il est alors possible de sonder l'espace dans un nombre de directions qui grandit à mesure que l'algorithme converge. Nous décrivons en détail le fonctionnement de ce type de méthode dans la section 2.2.

Ces classes de méthodes ne sont pas cloisonnées, et certains algorithmes empruntent des idées à plusieurs classes de méthodes. Par exemple, Conn et Le Digabel [20] exposent une manière de construire des modèles quadratiques pour guider une optimisation dans un contexte boite noire à l'aide d'une méthode de type recherche directe, De son côté, Regis [39], propose une utilisation de modèle de type fonctions à base radiale pour guider la résolution de $\mathcal{P}$ à l'aide d'un algorithme évolutionnaire. Audet et *al.* [12] proposent une utilisation des modèles pour guider une optimisation à l'aide de `MADS`, et Audet et Tribes [15] exposent une version de `NM` combiné à la mécanique de maillage utilisée dans `MADS`.

2.2 L'algorithme MADS

L'algorithme MADS [9] est un algorithme itératif de recherche directe qui comporte deux étapes : la SEARCH et la POLL. Ces deux étapes sont répétées jusqu'à satisfaire des critères de convergence spécifiés par l'utilisateur. Sa structure peut être résumée comme sur la figure 2.2. Les deux étapes font appel aux notions de maillage, de cadre de POLL et d'ensemble générateur positif définie comme suit :

Définition 2 (Ensemble générateur positif [6])**.** Un ensemble de vecteurs $\{d^1, ..., d^N\} \subset \mathbb{R}^n$ est dit générateur positif de $\mathbb{R}^n$ si ses combinaisons linéaires positives engendrent $\mathbb{R}^n$. Autrement dit si :

$$\forall x \in \mathbb{R}^n, \exists \lambda^1, ..., \lambda^N \in \mathbb{R}_+,\ x = \sum_{i=1}^{N} \lambda^i d^i$$

L'ensemble $\{d^1, ..., d^N\}$ est qualifié de base positive si aucun de ses sous ensembles strictes n'est un ensemble générateur positif. Contrairement aux bases, dont la cardinalité est nécessairement de n, une base positive peut comporter entre $n+1$ et $2n$ éléments [6].

Définition 3 (Maillage et pas du maillage [11])**.** Soit $G \in \mathbb{R}^{n\times n}$ inversible, et $Z \in \mathbb{Z}^{n\times p}$ dont les colonnes forment un ensemble générateur positif de $\mathbb{R}^n$. Posons $D = GZ$. Le maillage de pas $\delta^k > 0$ généré par D, centré en $x^k \in R^n$, est défini par :

$$M^k \ := \ \{\ x^k + \delta^k D y \ :\ y \in \mathbb{N}^p\ \} \subset \mathbb{R}^n$$

Définition 4 (Cadre et pas de POLL [11])**.** Soit $G \in \mathbb{R}^{n\times n}$ inversible, et $Z \in \mathbb{Z}^{n\times p}$ dont les colonne forment un ensemble générateur positif de $\mathbb{R}^n$. Posons $D = GZ$, $\delta^k > 0$ ainsi que $\Delta^k \geq \delta^k$. Le cadre de POLL de taille Δ^k généré par D centré en $x^k \in \mathbb{R}^n$ est défini par :

$$F^k \ := \ \{\ x \in M^k \ :\ \left\|x - x^k\right\|_\infty \leq \Delta^k b\ \} \subset \mathbb{R}^n$$

avec $b = \max\{\|d'\|_\infty, d' \in \mathbb{D}\}$, et $\mathbb{D}$ l'ensemble des colonnes de D.

Avec cette définition, F^k est un sous ensemble fini de M^k. C'est en effet l'intersection de M^k et de la boule de rayon $\Delta^k b$, centrée en x^k pour la norme $\|.\|_\infty$ Souvent, $D = [I_n, -I_n]$ et donc $b = 1$. Afin d'alléger les explications, on considère le cas sans contrainte obtenu à l'aide

de la barrière extrême [11] :

$$f_\Omega(x) = \begin{cases} f(x) & \text{si } x \in \Omega \\ +\infty & \text{sinon} \end{cases}$$

Pour débuter le fonctionnement de l'algorithme `MADS`, il est nécessaire de fournir un point réalisable x^0 : $f_\Omega(x^0) < +\infty$ ainsi qu'une taille initiale de cadre de `POLL` Δ^0 et une tolérance ϵ positive ou nulle. La figure 2.2 illustre le fonctionnement logique de `MADS`. L'algorithme 1 détaille la mise à jour des différents paramètres lors de l'exécution.

L'étape de recherche globale (`SEARCH`) à l'itération k consiste en l'évaluation de f_Ω sur un ensemble $\mathbb{S}^k \subset M^k$ **fini** de points. La logique qui permet de générer les points de `SEARCH` peut donc être de nature quelconque : elle peut faire appel à un autre algorithme d'optimisation, être un tirage aléatoire de points, être la volonté de l'opérateur, etc.

L'étape de sonde locale (`POLL`) à l'itération k centré en x^k consiste en l'évaluation de f_Ω sur un ensemble de points $y^1 = x^k + d^1, ..., y^i = x^k + d^i, ..., y^{n_p^k} = x^k + d^{n_p^k}$ appartenant au cadre de sonde F^k, de sorte que

$$\mathbb{D}^k := \left\{ d^1, ..., d^i, ..., d^{n_p^k} \right\}$$

forme un ensemble générateur positif de $\mathbb{R}^n$. La méthode utilisée [2] dans `NOMAD` 4 pour construire $\mathbb{D}^k$ fait appel à l'opérateur de Householder pour générer une base orthogonale : pour $v^k \in \mathbb{R}^n, \left\|v^k\right\|_2 = 1$, les colonnes de

$$H^k = I - 2(v^k)(v^k)^\top$$

forment une base orthogonale de $\mathbb{R}^n$. L'ensemble des colonnes de H^k et leur opposé forment donc une base positive de $\mathbb{R}^n$ de cardinalité $2n$, et donc $n_p^k = 2n$. Dans `NOMAD` 4 :

$$\mathbb{D}^k \;=\; \left\{ \pm\, \delta^k \text{ROUND}\left(\frac{\Delta^k}{\delta^k} \frac{h_j^k}{\left\|h_j^k\right\|_\infty} \right) \;:\; j \in [\![1, n]\!] \right\}$$

où h_j^k est la $j-$ième colonne de H^k, et l'opérateur $\text{ROUND}(\cdot)$ arrondit chacune des coordonnées d'un vecteur à l'entier le plus proche. Enfin, la suite de vecteurs unitaires $(v^k)_{k\in\mathbb{N}}$ est choisie dense sur la sphère unité. De ce fait, comme il est montré par Abramson et *al.* [2], $(h_i^k)_{k\in\mathbb{N}}$ est dense dans la sphère unité pour tout $i \in [\![1, n]\!]$.

La `SEARCH` permet l'évaluation de points en nombre fini n'importe où dans M_k (diversifica-

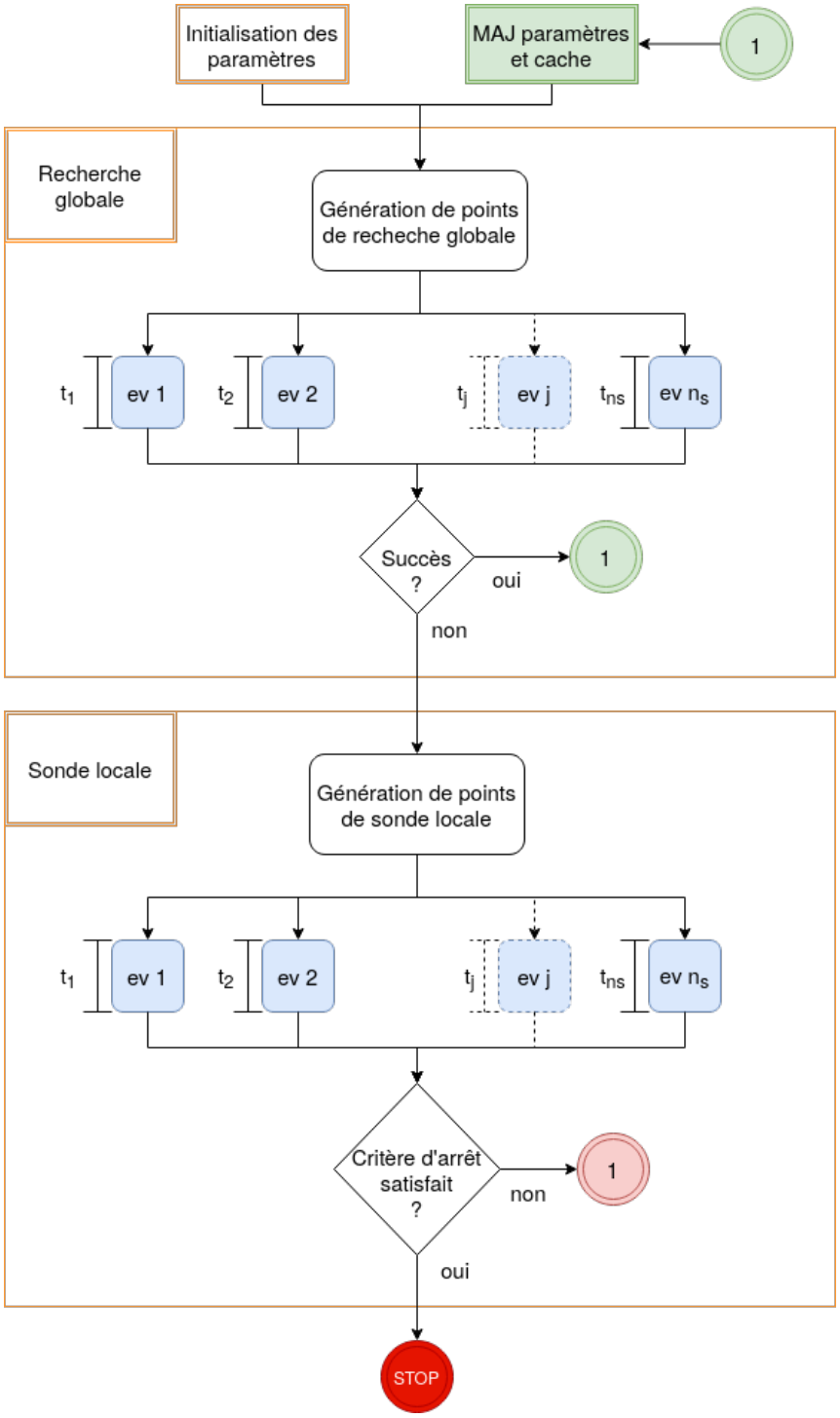

Figure 2.2 Graphe de dépendances de contrôle et de données de `MADS` avec évaluations en parallèle non opportuniste.

tion), tandis que la POLL permet uniquement l'évaluation de points de F_k (intensification). La figure 2.3 montre un maillage sur lequel les étapes de SEARCH et de POLL génèrent des points à évaluer. Le maillage est l'ensemble des points qui forment l'intersection des lignes fines. A chacune de ces deux étapes, il est possible d'obtenir des points candidats dans $\mathcal{X} \setminus \Omega$. L'évaluation par f_Ω permet de savoir si ces points sont réalisable ou non.

Comme vu plus haut, les règles qui définissent les étapes de SEARCH et de POLL sont flexibles sur la manière de choisir des points candidats à l'évaluation. De plus, lorsque l'algorithme converge, à chaque étape de SEARCH et de POLL, le nombre de points sont candidats à être évalué croit.

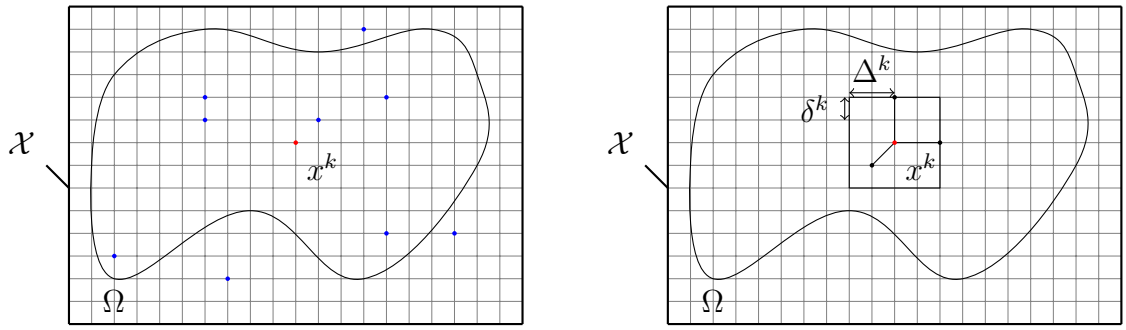

Figure 2.3 Si l'étape de SEARCH (gauche) ne parvient pas à améliorer la valeur de la fonction objectif, l'étape de POLL (droite) est exécutée.

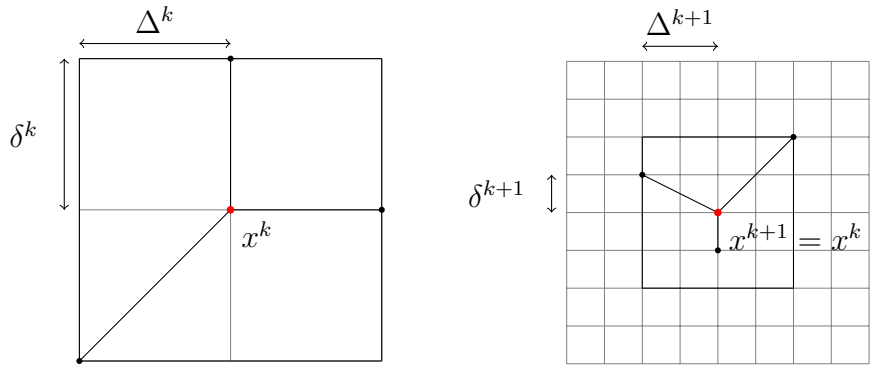

Figure 2.4 Un échec de l'étape de POLL conduit à une réduction de δ^k et de Δ^k.

Une étape de SEARCH ou de POLL est un échec lorsqu'elle ne parvient pas à améliorer la valeur de la fonction objectif. Afin d'éviter d'évaluer plusieurs fois f_Ω aux mêmes points lors de l'exécution de MADS, une cache des évaluations est maintenue à jour :

$$V := \left\{ \; (x^i, f_\Omega(x^i)) \;\; : \;\; i \in [\![1, q]\!] \; \right\}$$

où q est le nombre de points deux à deux distincts en lesquels f_Ω a été évalué. Ainsi, avant chaque évaluation de f_Ω une vérification est faite pour savoir si le point candidat est présent dans V. L'information détenue dans la cache permet aussi de construire des modèles dynamiques. L'algorithme 1 présente un fonctionnement détaillé de `MADS` dans le cas sans contrainte.

Algorithm 1 Mesh Adaptive Direct Search (`MADS`)

[0]. Initialisation :

$k \leftarrow 0$: Compteur d'itération.
$x^k \leftarrow x^0$: Point de départ.
$\Delta^k \leftarrow \Delta^0$: Pas de cadre.
$\delta^k \leftarrow \Delta^k$: Pas de maillage.
$\mathbb{S}^k \leftarrow \varnothing$: Ensemble des points de `SEARCH` à l'itération k
$\mathbb{P}^k \leftarrow \varnothing$: Ensemble des points de `POLL` à l'itération k
$V \leftarrow \{(x^k, f_\Omega(x^k))\}$: Cache des évaluations.
$\epsilon \in [0, +\infty[$: Tolérance

[1]. Recherche globale (optionnel) :

Construire $\mathbb{S}^k \subset M^k$.
Si $\exists t \in \mathbb{S}^k,\ f_\Omega(t) < f_\Omega(x^k)$:
  $(\Delta^{k+1}, \delta^{k+1}) \leftarrow \text{INCREASE}(\Delta_k, \delta_k)$.
  $x^{k+1} \leftarrow t$.
  Aller à [3].
Sinon :
  Aller à [2].

[2]. Sonde locale :

Construire $\mathbb{P}^k \subset F^k$.
Si $\exists t \in \mathbb{P}^k,\ f_\Omega(t) < f_\Omega(x^k)$:
  $(\Delta^{k+1}, \delta^{k+1}) \leftarrow \text{INCREASE}(\Delta_k, \delta_k)$.
  $x^{k+1} \leftarrow t$.
Sinon :
  $(\Delta^{k+1}, \delta^{k+1}) \leftarrow \text{DECREASE}(\Delta_k, \delta_k)$.
  $x^{k+1} \leftarrow x^k$.
Aller à [3]

[3]. Mise à jour :

  Mettre à jour V avec les évaluations effectuées.
  $\mathbb{S}^k \leftarrow \varnothing$.
  $\mathbb{P}^k \leftarrow \varnothing$.
  $k \leftarrow k + 1$.
Si $\delta^k < \epsilon$:
  STOP.
sinon :
  Aller à [1].

Les opérateurs INCREASE$(\cdot,\cdot)$ et DECREASE$(\cdot,\cdot)$ peuvent être définies de plusieurs manières. Dans la version originale exposée par Audet et Denis [9], la multiplication pour la diminution, ou la division pour l'augmentation, par un rationnel [1] $\tau \in]0,1[$ est utilisé :

$$\begin{aligned}\text{DECREASE}(\Delta^k,\delta^k) &= (\tau\Delta^k, \min\{\tau\Delta^k, (\tau\Delta^k)^2\})\\ \text{INCREASE}(\Delta^k,\delta^k) &= (\tau^{-1}\Delta^k, \min\{\tau^{-1}\Delta^k, (\tau^{-1}\Delta^k)^2\})\end{aligned}$$

Typiquement $\tau = \frac{1}{2}$. Tandis que dans la version exposée par Audet et *al.* [14], c'est un système utilisant une représentation compacte en base 10 qui sert à construire les opérateurs INCREASE$(\cdot,\cdot)$ et DECREASE$(\cdot,\cdot)$:

$$\text{INCREASE}((a\times(10)^b,(10)^{b-|b|}) = \begin{cases} 2\times(10)^b,(10)^{b-|b|} & \text{si } a=1\\ 5\times(10)^b,(10)^{b-|b|} & \text{si } a=2\\ 1\times(10)^{b+1},(10)^{(b+1)-|b+1|} & \text{si } a=5\end{cases}$$

$$\text{DECREASE}((a\times(10)^b,(10)^{b-|b|}) = \begin{cases} 5\times(10)^{b-1},(10)^{(b-1)-|b-1|} & \text{si } a=1\\ 1\times(10)^b,(10)^{b-|b|} & \text{si } a=2\\ 2\times(10)^b,(10)^{b-|b|} & \text{si } a=5\end{cases}$$

Δ^k et δ^k sont alors initialisés en conséquence. Dans les deux cas, les mises à jour de Δ^k et δ^k font en sorte que si $\Delta^k \underset{k\to+\infty}{\longrightarrow} 0$ alors, $\delta^k \underset{k\to+\infty}{=} o(\Delta^k)$. Ce qui implique que $|F^k| \underset{k\to+\infty}{\longrightarrow} +\infty$. De plus, Audet et *al.* [14] montrent qu'il est également possible de conserver les propriétés de convergence de l'algorithme en considérant non plus Δ^k et δ^k comme des scalaires, mais comme des vecteurs, pour effectuer des augmentations ou des réductions des paramètres de sorte à prendre en compte les différentes variables granulaires.

1. Il est possible de construire des exemples avec τ irrationnel pour lesquels la preuve de convergence de `MADS` [9] n'est plus valide.

2.3 Utilisation de ressources parallèles dans l'optimisation de boite noire

Dennis et Torczon [21] décrivent une classe de méthodes de recherche directe sur une machine multiprocesseurs, dans un cas d'optimisation sans contraintes. La stratégie développée est itérative, et met à jour la géométrie d'un simplexe selon les valeurs de la fonction objectif obtenue en chacun des sommets. D'après les auteurs, cette méthode induit que, pour un nombre de processeurs p plus grand que la dimension du problème n, certains de ces processeurs deviennent inutilisés. Pour profiter d'un grand nombre de processeurs devant la dimension du problème traité, les auteurs proposent de générer les points de plusieurs itération successives, pour ensuite y évaluer f, et ainsi occuper tous les processeurs disponibles. (thèse dont vient l'article : [41]). Torczon [42] décrit un ensemble de librairies en fortran qui implémentent, entre autres, l'algorithme décrit précédemment par Dennis et Torczon [21].

Le parallélisme synchrone et asynchrone de `GPS` [28, 31–33] permet d'occuper de manière optimale un nombre de processeur fixé. Comme l'algorithme est itératif, même si les évaluations sont effectuées en parallèle, il existe toujours une barrière de synchronisation qui entraîne l'inactivité de certains processeurs à chacune des itérations, car comme expliqué par Hough, Kolda et Torczon [28], les temps nécessaire à deux évaluations distinctes sont, en terme général, distincts. L'asynchronisme retire cette barrière et offre à l'algorithme un fonctionnement opportuniste : des décisions sont prises alors que certains processeurs n'ont pas fini certaines évaluations de f. Une exécution asynchrone est donc pertinente lorsque le nombre de processeurs est plus petit que le nombre d'évaluations à effectuer à chaque itération et que l'on cherche à minimiser le temps total d'inactivité des processeurs mais que l'algorithme est itératif par nature. Un fonctionnement asynchrone dans un environnement parallèle n'est pas déterministe.

Hough et Meza [29] décrivent une classe d'algorithme de type région de confiance (`TR`) dans lequel, un sous problème est généré à chacune des itérations, et est résolu à l'aide d'une méthode de recherche directe parallélisée. Chazan et Miranker [17] détaillent une méthode de Powell [38, 46] adaptée pour un fonctionnement parallèle. La méthode séquentielle utilise n minimisations dans une direction différente, chainées les unes après les autres. La méthode expliquée ici effectue les n minimisations en parallèle. Griffin et *al.* [26] présentent une version asynchrone de la méthode de gnénération d'ensemble de recherche (`GSS`). La mécanique asynchrone exposée est semblable à celle donnée pour `GPS` plus haut. García-Palomares et Rodríguez [24] proposent une classe d'algorithmes de recherche directe pour l'optimisation sans contrainte, et exposent deux variantes d'implémentation suivant si les communications entre les processeurs sont négligeable devant le temps d'évaluation de f ou non. Une version

asynchrone est également proposée.

Les travaux qui ont été effectués sur `MADS` portant sur l'utilisation de ressources parallèles sont les suivants. L'algorithme `p-MADS` [3] effectue les évaluations de points candidats en parallèle, de manière synchrone ou asynchrone suivant l'option spécifiée. Il revient alors à l'utilisateur de choisir le nombre de processeurs à utiliser en parallèle. L'algorithme `COOP-MADS` [3] exécute plusieurs instances de `MADS` en parallèle, chacune des instances étant pourvu d'une graine aléatoire différente. L'algorithme `PSD-MADS` [5] est applicable aux problèmes de grande dimension devant le nombre de processeurs. Cette version de l'algorithme `MADS` effectue une décomposition de l'espace et utilise plusieurs processeurs pour résoudre chacun des sous problèmes.

La figure 2.5 met en évidence le phénomène de saturation observé lorsque le nombre de processeurs excède le nombre de points généré lors d'une itération d'un algorithme de recherche directe. Lorsqu'il y a suffisamment de points générés à chacune des itérations, tous les processeurs peuvent être occupés. Cependant, si la dimension du problème est petite devant le nombre de ressources parallèle, la dernière configuration de la figure 2.5 est plus susceptible de survenir. Lors du fonctionnement de l'algorithme 1, il est pertinent d'utiliser des ressources parallèles uniquement pour déterminer si l'assertion de l'existence d'un point de `SEARCH` $t \in \mathbb{S}^k$ ou d'un point de sonde locale $t \in \mathbb{P}^k$ tel que $f_\Omega(t) < f_\Omega(x^k)$ est vraie ou fausse. Ainsi, lorsque l'algorithme est rendu à l'une de ces deux étapes, les évaluations sont distribuées sur les processeurs comme représenté sur la figure 2.5. Dans un fonctionnement non opportuniste, le nombre d'évaluations à effectuer est $|\mathbb{S}^k|$ ou $|\mathbb{P}^k|$ suivant l'étape à laquelle l'algorithme est rendu.

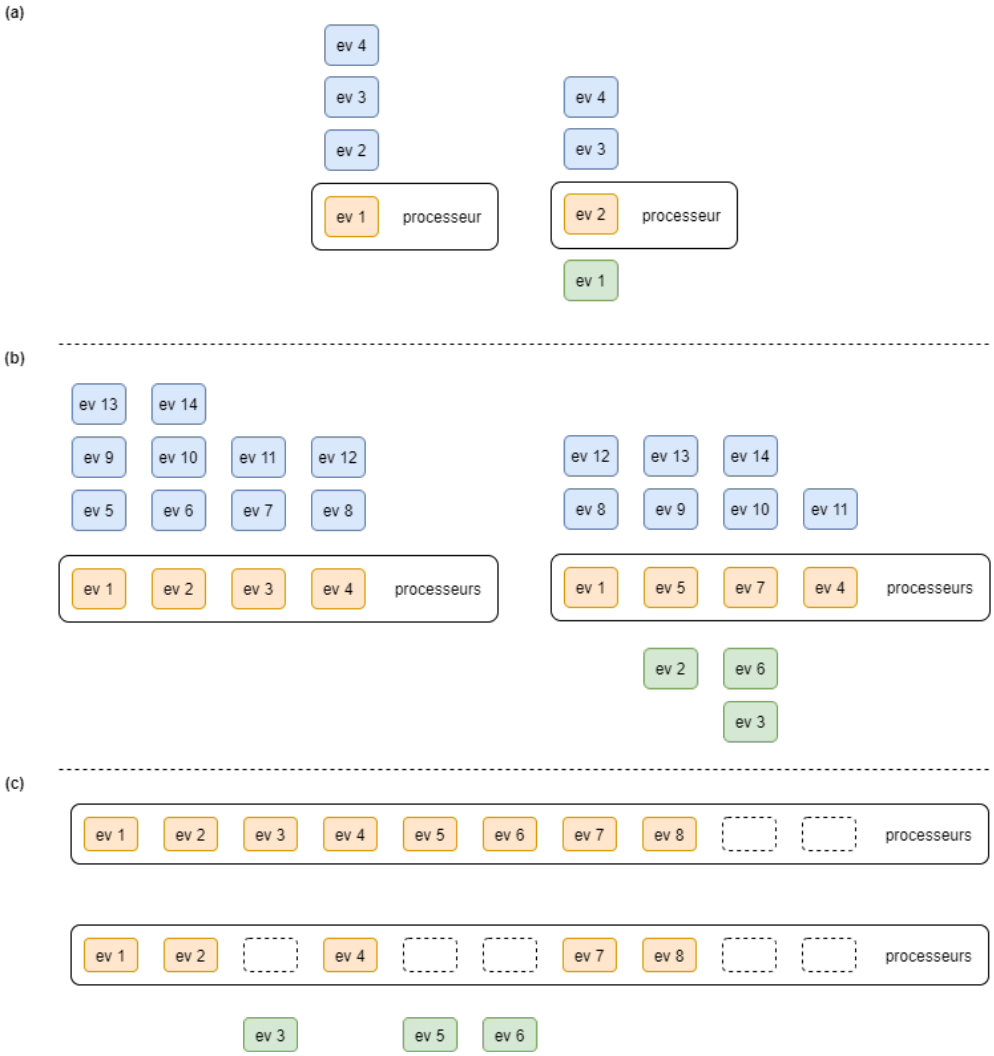

Figure 2.5 (a) une suite d'évaluations en séquentielle, (b) une suite d'évaluations parallèle avec moins de processeurs que d'évaluations à effectuer, (c) une suite d'évaluations avec plus de processeurs que d'évaluation à effectuer.

2.4 Approche informatique de la programmation sur des ressources parallèles

Dans cette sous-section, des notions propres à l'informatique parallèle sont présentées. On résume ici les propos des notes de cours sur la programmation parallèles donné à l'UQAM par Tremblay [44].

Processus vs. fil d'execution

Définition 5 (Processus [44])**.** Un processus représente un programme en cours d'exécution.

D'après Tremblay, un processus possède un espace mémoire privé, indépendant des autres processus. Deux processus qui veulent communiquer doivent donc utiliser un mécanisme d'échanges de messages. Le contexte associé au fonctionnement d'un processus est lourd. Plusieurs objets sont sollicités(stack pointer, instruction pointer,...) et créer un processus nécessite la création et la gestion de l'ensemble de ces objets.

Définition 6 (Fil d'exécution [44])**.** Un fil d'exécution représente une fonction en cours d'exécution.

Toujours d'après Tremblay [44], un processus peut posséder plusieurs fil d'exécutions, qui partagent entre autre l'espace mémoire du processus. Deux fil d'exécutions qui veulent communiquer peuvent donc le faire au travers d'une variable partagée, stockée dans la mémoire du processus parent. Le contexte associé au fonctionnement d'un fil d'exécution est léger en comparaison à celui d'un processus.

Souvent, une boite noire est matérialisé un code compilé, et une valeur de f_Ω obtenue lors de l'exécution de `NOMAD` 4 est en fait le résultat de l'exécution de ce code. Par exemple pour évaluer la boite noire représenté par le programme `bb.exe` au point $x = (1, 2, 3)$, il est d'usage de configurer `NOMAD` 4 pour appeler une commande du type :

```
./bb.exe 1 2 3
```

qui signifie que le programme `bb.exe` est exécuté avec pour paramètres d'entrées $1, 2$ et 3. Ainsi, l'appel à la boite noire pour effectuer une évaluation signifie la création et l'exécution d'un processus informatique. Les paramètres de `NOMAD` 4 ne modifient pas le fonctionnement de la boite noire, et donc, ne modifient pas l'exécution du processus créé. En particulier, dans `NOMAD` 4, il est possible de régler le nombre de fils d'exécution à l'aide du paramètre `NB_THREADS_OPENMP`. Ainsi fixer `NB_THREADS_OPENMP`$= 3$ signifie qu'on autorise au plus 3 instances de `bb.exe` à s'exécuter simultanément : on s'autorise à exécuter une commande du type :

```
./bb.exe 1 2 3 & ./bb.exe 4 5 6 & ./bb.exe 7 8 9 &
```

Programmation séquentielle et concurrente

Tremblay [44] définit un programme séquentiel comme ne possédant qu'un seul fil d'exécution : "Un seul doigt suffit pour indiquer l'instruction en cours d'exécution.". Inversement, un programme concurrent possède plusieurs fils d'exécutions qui coopèrent : "plusieurs doigts sont nécessaire pour indiquer les instruction en cours d'exécution." La coopération implique l'échange d'information, au travers de variables partagées dans le cas des fils d'exécution. Dans le contexte de NOMAD 4, l'une des principales variables partagées entre les fils d'exécution est la cache des évaluations. Cette dernière est mise à jour par chaque fil d'exécution lorsqu'une évaluation est complétée.

Tremblay [44] classifie les programmes concurrents en trois familles :

- «Application **multi-contextes** (multi-threaded) = contient deux ou plusieurs threads, qui peuvent ou non s'exécuter en même temps, et qui sont utilisés pour mieux organiser et structurer l'application (meilleure modularité). Exemples : Système d'exploitation multi-tâches, fureteurs multi-tâches, inter-face personne-machine vs. traitement de la logique d'affaire, serveur Web.
- Application **parallèle** = chaque thread s'exécute sur son propre CPU(ou coeur), dans le but de résoudre plus rapidement un problème — ou pour résoudre un problème plus gros. Exemples : Prévisions météorologiques, modélisation du climat, simulations physiques, bio-informatique, traitement graphique, etc.
- Application **distribuée** = contient deux ou plusieurs processus, qui communiquent par l'intermédiaire d'un réseau ($\Longrightarrow$ délais plus longs), et ce pour répartir, géographiquement, des données et des traitements. Exemples : Serveurs de fichiers, accès à distance à des banques de données. »

La figure 2.6 illustre les notions d'exécution parallèles et concurrentes.

NOMAD 4 est donc à considérer comme une application parallèle. La figure 2.6 illustre la différence entre exécution concurrente non parallèle et exécution parallèle. Les rectangles gris de la figure 2.6 sont assimilables aux évaluations de f_Ω.

Grannularité lors de la conception d'un programme parallèle

Lorsqu'il est possible de paralléliser une charge de travail, deux grandeurs se distinguent : la grannularité (unité parallèle) qui est une séquence d'instructions fixée comme insécable, et les fils d'exécutions capable de compléter les séquences d'instructions demandées.

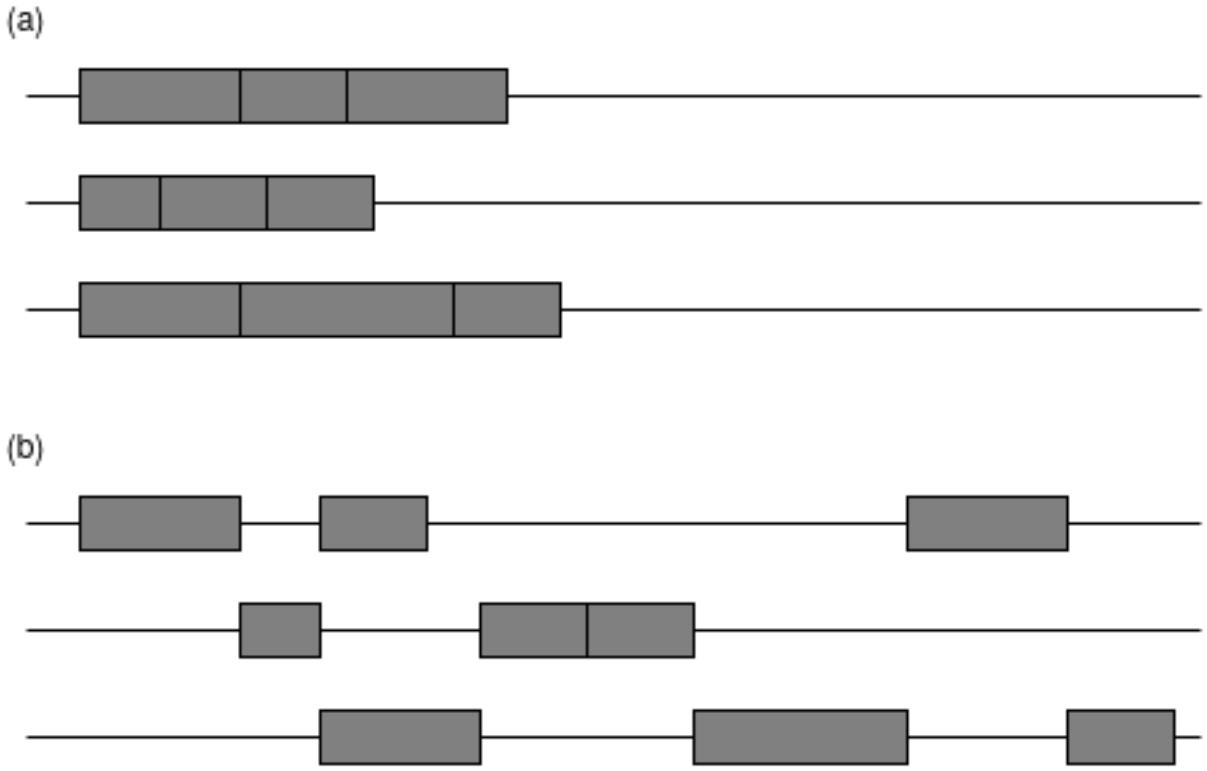

Figure 2.6 (a)-Exécution concurrente et parallèle de trois fils d'exécution vs. (b)-Exécution concurrente non parallèle. Inspiré de Tremblay [44]

Soit le calcul d'un produit scalaire entre deux vecteurs $u, v \in \mathbb{R}^n$ dont le résultat doit être stocké dans la variable r. Une unité parallèle peut être définie comme :

1. : Copier u_i et v_i dans $\tilde{u_i}$ et $\tilde{v_i}$.
2. : effacer u_i et v_i de u et v.
3. : calculer le produit $\tilde{u_i}$ $\tilde{v_i}$ et le stocker dans $\tilde{r}$.
4. : calculer la somme $r + \tilde{r}$ et placer le résultat dans r.

Les instructions 1,2 et 4 font intervenir des variables globales. Lorsqu'un fil d'exécution exécute cette séquence d'instruction, il doit mettre à jour les variables globales. Des communications sont engendrées pour verrouiller et déverrouiller les espaces mémoire associés aux variables en jeu. Ces opérations de verrouillage et déverrouillage des espaces mémoires sont relativement coûteuses comparé à une simple addition. Ainsi, dans cet exemple, il n'est pas pertinent de fixer le nombre de fils d'exécution comme égal au nombre d'unités parallèles présentes dans la tâche à effectuer. En effet, pour que le découpage en unités parallèles soit bénéfique, il faut que le temps accordé à l'instruction 3 soit dominant devant le temps accordé aux instructions 1,2 et 4. Cependant, si l'instruction 3 demande trop de ressources pour être complétée, alors les performances sont aussi dégradées. La figure 2.7 illustre le compromis à réaliser lorsqu'un programme est parallélisé.

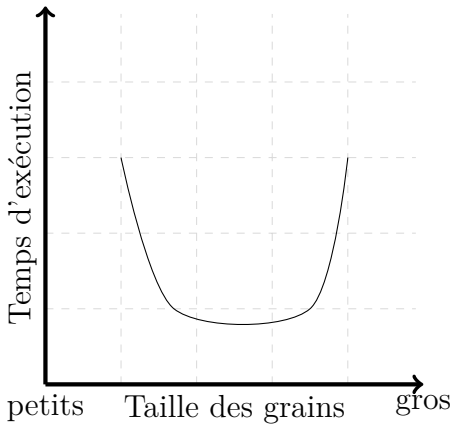

Figure 2.7 Courbe illustrant l'effet général de la taille des grains (des tâches) sur le temps d'exécution. Inspiré de Tremblay [44]

2.5 Métriques de performance pour algorithmes et programmes parallèles

Tremblay [44] introduit également un cadre théorique pour l'étude des performances des algorithmes parallèles.

Coût, travail et temps d'exécution

Définition 7 (Temps d'exécution parallèle [44])**.** Le temps d'exécution $T_P(p, n)$ d'un algorithme A sur un problème de taille n est le temps requis par l'algorithme pour résoudre le problème à l'aide de p processeurs.

Définition 8 (Temps d'exécution parallèle absolu [44])**.** le temps d'exécution $T_P(n)$ d'un algorithme A sur un problème de taille n est le temps requis par l'algorithme pour résoudre le problème en supposant que le nombre de processeurs est illimité, c'est-à-dire que toutes les opérations qui peuvent être effectuées de manière parallèle le sont effectivement.

Par ce qui précède, il vient :

$$T_P(n) = T_P(+\infty, n)$$

Définition 9 (Coût d'un algorithme parallèle [44])**.** Soit A un algorithme qui résout un problème de taille n en un temps d'exécution parallèle absolu $T_P(n)$ et, qui pour cela, utilise effectivement $p(n)$ processeurs. Le coût de cet algorithme est $C(n) = p(n)T_P(n)$

L'algorithme A n'utilise pas nécessairement tous les $p(n)$ processeurs à chaque instant de son fonctionnement, mais dans la plupart des cas, ces processeurs restent disponible. En effet, la plupart des ordinateurs ne coupent pas l'alimentation des processeurs non utilisés. Le coût permet donc de majorer la quantité des ressources utilisées par l'algorithme.

Définition 10 (Travail d'un algorithme parallèle [44])**.** Le travail $W(n)$ d'un algorithme A instancié sur un problème de taille n est le nombre d'opérations nécessaires à la résolution du problème.

Le travail donne une quantification plus exacte des ressources utilisées par l'algorithme que le coût.

Dans le contexte actuel, il est nécessaire de discuter de la définition d'opération. Si l'on considère une opération comme une évaluation de f_Ω, alors le fonctionnement interne de `NOMAD` 4 est négligeable en terme de ressources et n'intervient pas dans le calcul totale du coût. Si l'on considère une opération comme une opération arithmétique, ou une suite d'opérations

arithmétique, il est impossible de connaître le nombre d'opération arithmétiques effectuées dans la boite noire. Ainsi, il devient difficile de quantifier le travail effectué lors de l'évaluation de f_Ω de manière cohérente.

Tremblay [44] définit également le concept d'optimalité en travail ou en coût, à l'aide de $T_S^*(n)$, le temps mis par le meilleur algorithme séquentiel pour résoudre le problème de taille n.

Accélération et efficacité

L'utilisation d'un algorithme parallèle est souvent faite dans le but de réduire le temps de résolution du problème donné. Pour mesurer le gain obtenu lors de l'utilisation d'un algorithme parallèle à la place d'une version séquentielle, il est d'usage d'introduire l'accélération. Tremblay [44] définit deux types d'accélération : l'accélération relative et l'accélération absolue. Le deuxième type fait également appel à $T_S^*(n)$.

Définition 11 (Accélération relative [44])**.** Avec les notation précédentes, l'accélération relative $A_p^r(p,n)$ est définie par le rapport :

$$A^r(p,n) = \frac{T_P(1,n)}{T_P(p,n)}$$

Définition 12 (Accélération absolue [44])**.** Avec les notation précédentes, l'accélération absolue $A_p^a(p,n)$ est définie par le rapport :

$$A^a(p,n) = \frac{T_s^*(n)}{T_P(p,n)}$$

Toujours d'après Tremblay [44], l'accélération relative est optimiste, et s'écarte parfois de la réalité. l'accélération absolue étant plus exacte, est obtenue en remplaçant $T_P(1,n)$ par $T_S^*(n)$ dans la définition ci dessus. Á partir de l'accélération absolue, il est possible de définir la notion d'efficacité. Cette notion permet de mesurer l'occupation de processeurs fonctionnant en parallèle à l'exécution de l'algorithme A.

Définition 13 (Efficacité [44])**.** L'efficacité d'un algorithme (programme) est le rapport entre le temps d'exécution séquentiel et le coût d'exécution sur une machine à p processeurs :

$$E_p = \frac{T_s^*(n)}{C(n)} = \frac{T_s^*(n)}{p \times T_P(p,n)} = \frac{A^a(p,n)}{p}$$

En pratique, on fait varier le nombre de fils d'exécution utilisés dans le programme pour me-

surer l'accélération et l'efficacité : p compte le nombre de fils d'exécution, et un fil d'exécution désigne un processeur au sens de la définition 1.

Loi d'Amdahl

En notant :

- σ : temps d'exécution de la partie intrinsèquement séquentielle du programme,
- ϕ : temps d'exécution de la partie intrinsèquement parallèle du programme,
- ψ : l'accélération observée,

il vient :

$$T_P(1,n) = \sigma + \phi \tag{2.1}$$

$$T_P(p,n) = \sigma + \frac{\phi}{p} \tag{2.2}$$

$$\psi \leq \frac{T_P(1,n)}{T_P(p,n)} = \frac{\sigma + \phi}{\sigma + \frac{\phi}{p}} \tag{2.3}$$

ici f est définie comme "la fraction des opération d'un calcul qui doivent être exécutées de façon séquentielle" [44] :

$$f = \frac{\sigma}{\sigma + \phi}$$

L'inégalité précédente se met sous la forme suivante (loi d'Amdahl [44]) :

$$\psi \leq \frac{1}{f + (1-f)/p}$$

Cette loi révèle que peu importe de combien la partie parallélisable du programme est accélérée en ajoutant des processeurs, l'accélération globale est majorée du fait du fonctionnement intrinsèquement séquentiel du programme.

Loi de Gustafson-Barsis

Tremblay [44] affirme que la loi d'Amdahl approche la notion de parallélisme avec pour objectif d'exécuter le programme d'intérêt le plus rapidement possible sur un problème de taille fixe. La loi de Gustafson-Barsis se construit en supposant que le temps d'exécution est fixé, et que la taille du problème traité et le nombre de processeurs varient [27]. Dans un contexte réaliste, cette approche est pertinente dans la mesure où un problème industriel fixé n'est pas résolu plusieurs fois avec un nombre de processeurs qui varie, mais la grannularité

du problème pour une exécution parallèle ainsi que le nombre de processeurs utilisé peut être ajusté par l'opérateur de sorte que le temps d'exécution soit constant.

Avec les notations précédemment utilisées pour la définition de la loi d'Amdhal, s est définie comme "la fraction du temps d'exécution parallèle consacré aux opérations séquentielles" [44] :

$$s = \frac{\sigma}{\sigma + \phi/p}$$

Définition 14 (Loi de Gustafson–Barsis [44])**.** Pour un problème de taille n traité avec p processeurs, soit s la fraction du temps d'exécution parallèle consacrée à la partie intrinsèquement séquentielle : $0 \leq s \leq 1$. Alors, l'accélération maximale ψ pouvant être obtenue est la suivante :

$$\psi \leq p + (1 - p)s$$

Cette équation s'obtient par un jeu de substitution dans l'équation 2.3 après avoir transformé le numérateur en $(\sigma + \phi/p)(s + (1 - s)p)$.

2.6 Parallélisme dans le logiciel NOMAD 4

L'algorithme MADS est implémenté dans le logiciel NOMAD 4 [13] écrit en C++. Le logiciel utilise OpenMP pour effectuer les évaluations en parallèle. NOMAD 4 dispose d'un paramètre NB_THREAD_OPENMP permettant de spécifier le nombre maximum de fils d'exécutions autorisés à être créés. c'est à dire, le nombre maximum d'instances de la boite noire qui sont autorisées à fonctionner simultanément. C'est ensuite le système d'exploitation qui planifie les charges de travail à fournir aux CPUs. OpenMP restreint le parallélisme à une seule machine : les ressources disponibles sur le réseau auquel est connecté la machine ne sont pas rendu accessible via la librairie.

Lorsqu'un programme parallélisable est voué à fonctionner sur p CPUs d'une même machine, il est d'usage d'attribuer $2p$ fils d'exécution. Cela permet que lorsque les fils d'exécutions sont en attente, d'autres puissent s'exécuter sans que la queue d'attente soit trop importante. Cela rejoint l'idée exposée par la figure 2.7 dans la section précédente. Cette règle est empirique, mais conduite à de bons résultats en pratique.

Donc, si l'utilisateur souhaite utiliser p CPUs lors de l'optimisation, fixer NB_THREAD_OPENMP $= 2p$ est un bon moyen empirique d'occuper au mieux les CPUs disponibles. Cependant, si les évaluations de la boite noire sont suffisamment peu coûteuses, dans le cas où f_Ω est une fonction analytique par exemple ,il peut être plus efficace d'utiliser un seul fil d'exécution, même si le nombre d'évaluations dépasse le nombre de CPUs.

NOMAD 4 dispose aussi d'un paramètre BB_MAX_BLOCK_SIZE qui est utile lorsque c'est la boite noire qui est capable de gérer les évaluations en parallèle. Ce paramètre spécifie le nombre maximum de points envoyés d'un seul tenant à la boite noire.

BB_MAX_BLOCK_SIZE et NB_THREAD_OPENMP sont en fait équivalents. La différence est la suivante : dans le cas de NB_THREAD_OPENMP, le processus que modélise la boite noire est instancié plusieurs fois sur la même machine physique que celle où est exécuté NOMAD 4. Chaque fil d'exécution est créé et géré par NOMAD 4 via OpenMP et ne peuvent pas être créés sur des machines distantes. Tandis que dans le cas de BB_MAX_BLOCK_SIZE, c'est directement la boîte noire qui gère la répartition parallèle du travail. Ainsi, dans le second cas, l'utilisateur peut bénéficier de la totalité des ressources d'une infrastructure de calcul sans avoir à interférer avec le fonctionnement de NOMAD 4, car un travail effectué en parallèle à l'aide de OpenMP est tout de même confiné sur une seule machine, et ne peut pas profiter des communications entre les différents noeuds d'une infrastructure de calcul.

La figure 2.8 montre les nuances entre les différents objets d'un environnement numérique parallèle. Si la boite noire ne gère pas les évaluations en parallèle, c'est NOMAD 4 qui crée et

gère les fils d'exécution logiciel. Dans le cas contraire, c'est la boite noire qui s'en charge. La boite noire peut aussi faire appel à d'autres types de mécanismes pour la gestion des évaluations en parllèle.

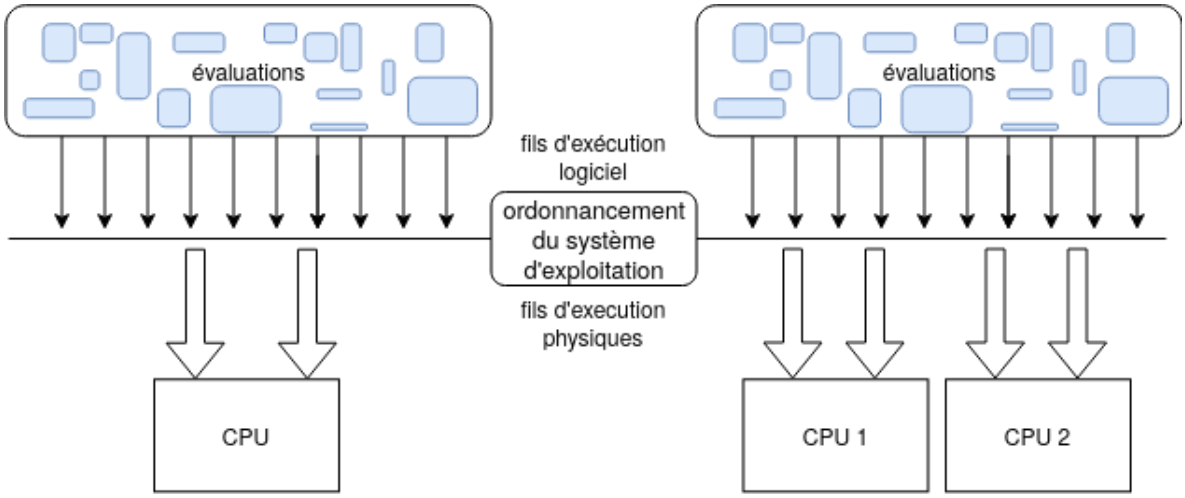

Figure 2.8 Nuances entre fils d'exécution physiques, fils d'exécution logiciel et pile d'évaluations.

Les fils d'exécution logiciel sont créés par les processus, tandis que le nombre de fils d'exécution physique dépend directement de l'architecture physique des CPUs (hyperthreading). A droite, l'exécution se fait sur une machine mono-CPU avec hyperthreading, à gauche, l'exécution se fait sur une machine multi-CPU avec hyperthreading

2.7 Occupation d'un grand nombre de processeurs lors de l'étape de recherche globale

Les travaux exposés jusque là supposent implicitement la configuration (b) de la figure 2.5. S'en suit une présentation des travaux de B. Talgorn, M. Kokkolaras [30] et S. Alarie [4] concernant la génération de points lors de l'étape de `SEARCH` à l'aide d'une optimisation sur un modèle $(\mathcal{S})$ (2.4) du problème $(\mathcal{P})$. Ces travaux concernent le cas (c) de la figure 2.5 et visent à obtenir une configuration semblable au cas (b) de 2.5, donc à construire $\mathbb{S}^k$ de manière pertinente de sorte que $|\mathbb{S}^k| > p$. Comme vu dans la section 2.1, il est possible de construire un modèle $(\mathcal{S})$ du problème $(\mathcal{P})$, moins coûteux à évaluer et qui partage néanmoins des similarités avec le problème initial :

$$\mathcal{S} : \begin{cases} \min\limits_{x \in \mathcal{X}} & \tilde{f}(x) \\ \text{s.c} & \tilde{c_j}(x) \leq 0, \quad j \in [\![1, m]\!]. \end{cases} \tag{2.4}$$

Où $\tilde{f}$ et les $\tilde{c_j}$ partagent des similarités avec f et les c_j et sont moins coûteux à évaluer. Il existe de nombreux types de modèles dynamiques [18, 25, 40]. Il est aussi possible que l'utilisateur ait à disposition un modèle statique obtenu par une simplification du fonctionnement interne de la boîte noire. Les explications qui suivent sont inchangées par la nature du modèle. L'algorithme 2 est adaptable aux modèles statiques.

Pour x et $y \in \mathbb{R}^n$, la distance entre deux points est définie comme suit :

$$d(x, y) = \|x - y\| . \tag{2.5}$$

Comme norme, il est possible d'utiliser $\|.\|_2$ ou $\|.\|_\infty$. Il en découle la distance d'un point à un ensemble donné par :

$$d(x, Y) = \min_{y \in Y} d(x, y) = \min_{y \in Y} \|x - y\| . \tag{2.6}$$

L'agrégation des contraintes [10] est définie par :

$$\tilde{h}(x) = \sum_{j=1}^{m} \max(\tilde{c_j}(x), 0)^2. \tag{2.7}$$

Où m et les c_j sont les grandeurs relatives à $(\mathcal{P})$ définie en introduction.

Enfin, la relation d'ordre $\prec$ sur $\mathbb{R}^n$ pour des points évalués sur $(\mathcal{S})$ est définie par :

$$x \prec y \Longleftrightarrow \tilde{h}(x) < \tilde{h}(y) \text{ ou } (\tilde{h}(x) = \tilde{h}(y) \text{ et } \tilde{f}(x) < \tilde{f}(y)). \tag{2.8}$$

La génération de points de SEARCH se fait à l'aide de la cache des évaluations produite lors de la résolution du problème $(\mathcal{S})$. L'algorithme 2 détaille la mise à jour des différents paramètres.

Algorithm 2 Génération de points pour la SEARCH

[0]. Initialisation :
- M^k : Maillage à l'itération k de MADS
- $\mathbb{S}^k$: Ensemble de points de SEARCH
- $T \leftarrow \varnothing$.
- $\tilde{V} \leftarrow \varnothing$: Cache des évaluations de $\tilde{f}$ et des $\tilde{c_j}$
- $\mathcal{F}_i : i \in [\![1, 6]\!]$: stratégies de filtrage
- $i \leftarrow 1$.
- Choisir $t_{max} \in \mathbb{N}^*$: nb. max. de points à générer.
- Choisir V un ensemble de points déjà évalués par f et les c_j.

[1]. Génération de $\tilde{V}$:
- Construire $(\mathcal{S})$ avec l'information donnée par V.
- Résoudre $(\mathcal{S})$ et stocker l'information obtenue dans $\tilde{V}$.
- Aller à [2].

[2]. Filtrage des points de $\tilde{V}$
- **Tant que** $\mathcal{F}_i(\tilde{V})$ retourne un point $\tilde{s}$ **faire** :
  - **Si** $|T| < t_{max}$ **faire** :
    - $T \leftarrow T \cup \{\tilde{s}\}$.
  - **Sinon** :
    - Go to [3].
- **Sinon** :
  - $i \leftarrow i + 1$.
  - Aller à [2].

[3]. Projection sur le maillage
- **Pour** $\tilde{s} \in T$ **faire** :
  - Projeter $\tilde{s}$ sur M^k pour obtenir s.
  - $\mathbb{S}^k \leftarrow \mathbb{S}^k \cup \{s\}$

L'étape 3 de l'algorithme 2 aboutit à un ensemble de points candidats pour la SEARCH. f et les c_j sont évalués en parallèle en ces points. Les six stratégies de sélection de points applicables lors de l'étape de SEARCH définies par B. Talgorn et M. Kokkolaras [30] sont décrites ci dessous.

Meilleur point De tous les points qui ont été évalués lors de l'optimisation de $(\mathcal{S})$, on souhaite garder le meilleur, en espérant qu'il fournisse une aussi bonne solution à $(\mathcal{P})$ qu'à $\mathcal{S}$.

Dans cette méthode, les points de $\tilde{V} \setminus V \cup T$ sont triés suivant la relation d'ordre $\prec$, et le meilleur point est sélectionné pour être ajouté à T.

Point le plus distant de V et T Itéré plusieurs fois, la stratégie précédente peut facilement retourner des points dans la même région. Dans cette deuxième stratégie, les régions ciblées sont celles qui n'ont pas encore été échantillonnées. C'est donc le point de S le plus loin de $V \cup T$ (au sens de d) qui est ajouté à T.

Meilleur point à distance minimale de V et T Un autre moyen d'appliquer la stratégie 2.7 sans risquer d'obtenir beaucoup de points dans la même région, est d'imposer une distance minimale entre les points déjà choisis et le nouveau point à ajouter à T

Le meilleur point de $\tilde{V}$ (au sens de $\prec$) est sélectionné à une distance $d_{\min}$ (au sens de d) de $V \cup T$. Avec cette stratégie, si $d_{\min}$ est trop grand, la stratégie peut ne pas retourner de point. En pratique, la stratégie est initialisée avec $d_{\min} = 0$, puis $d_{\min} = \Delta_k$

Meilleur point réalisable Le meilleur point réalisable, s^*, est sélectionné de sorte qu'il vérifie :

$$\forall s \in \tilde{V},\ c_{\max}(s^*) = \max_{j=1..m} \tilde{c}_j(s^*) \leq c_{\text{margin}} \text{ et } \tilde{f}(s^*) < \tilde{f}(s)$$

Avec :

$$c_{\text{margin}} = \max_{\substack{s \in S \\ c_{\max}(s)<0}} c_{\max}(s)$$

au début, comme première valeur, puis :

$$c_{\text{margin}} = 2(c_{\max}(s^*))_{k-1}$$

La sélection retourne un point qui satisfait de plus en plus les contraintes. Si la suite des $(c_{\text{margin}})_k$ prend des valeurs trop basse, il est possible de ne plus pouvoir retourner de point satisfaisant.

Point le plus isolé Cette stratégie repose sur une idée semblable à la stratégie Point le plus distant, qui consiste à chercher un point loin des régions déjà échantillonnées. Un indicateur d'isolement, d_{isolated}, est appliqué à chacun de points. Cette mesure, appliqué à un point $s \in \tilde{V}$ permet de connaître la distance qui sépare s de son voisin le plus proche, étant inférieur à s au sens de $\prec$. Cet indicateur repose sur la notion d'isolation topographique d'un sommet d'une chaîne de montagne. Ici, chaque point est vu comme un sommet, et le plus

"haut" sommet s^* est recherché [2]. Ce plus haut sommet, s^*, est caractérisé comme ayant le plus grand nombre de points dans un rayon de $d_{\text{isolated}}(s^*)$.

Pour $s \in \tilde{V}$:

1. $d_{\text{isolated}}(s)$ représente la distance (au sens de d) à l'ensemble des points $u \in \tilde{V}$ inférieurs à s (au sens de $\prec$), donc meilleur que s relativement à f et h :

$$d_{\text{isolated}}(s) = \min_{\substack{u \in S \\ u \prec s}} d(s,u) = d(s, \{u \in S, u \prec s\})$$

2. $n_{\text{isolated}}(s)$ représente le nombre de points $t \in S$ dans un rayon de $d_{\text{isolated}}(s)$ par rapport à s :

$$n_{\text{isolated}}(s) = |\{t \in \tilde{V}, d(s,t) < d_{\text{isolated}}(s)\}| = |\mathcal{B}_{d_{\text{isolated}}(s)}(s) \cap \tilde{V}|$$

Cette stratégie cherche un point s^* à ajouter à T, qui maximise la quantité $n_{\text{isolated}}(s)$

$$\forall s \in \tilde{V},\ n_{\text{isolated}}(s^*) > n_{\text{isolated}}(s)$$

Point dans la zone la plus peuplée Cette stratégie recherche le point s^* dans la zone la plus peuplée. Le but de cette stratégie est de sélectionner les points de $\tilde{V}$ ayant beaucoup de voisins dans $\tilde{V}$, mais peu de voisins dans V. L'existence d'un tel point signifie l'existence d'une zone de l'espace que la résolution de $(\mathcal{S})$ a déterminée comme prometteuse mais qui n'a pas beaucoup été explorée lors de la résolution de $(\mathcal{P})$.

1. $d(s, V \cup T)$ représente la distance entre s et l'ensemble des points déjà évalués.
2. $n_{\text{density}}(s)$ compte le nombre de points de $\tilde{V}$ dans un rayon de $d(s, V \cup T)$ autour de s

$$n_{\text{density}}(s) = |\{t \in \tilde{V}, d(s,t) < d(s, V \cup T)\}| = |\mathcal{B}_{d(s,V \cup T)}(s) \cap \tilde{V}|$$

s^* doit alors vérifier :

$$\forall s \in \tilde{V},\ n_{\text{density}}(s^*) > n_{\text{density}}(s)$$

2. dans le cas présent, on minimise, on devrait parler de plus profonde vallée

CHAPITRE 3 Stratégies de POLL intensives

Ce chapitre commence par rappeler le mécanisme de génération de points de POLL implémenté dans NOMAD 4. La section 3.1 présente ce mécanisme. La section 3.2 expose trois nouvelles stratégies de génération de points dans le but d'occuper au mieux un ensemble de processeurs à l'aide d'évaluations lors de l'étape de POLL. En effet, comme la recherche globale, l'étape de POLL reste générique concernant les conditions que doivent satisfaire les points évalués. Les trois stratégies ont pour nom : la Multi POLL, la POLL en Oignon et la POLL Enrichie. Les deux premières sont construites sur une idée semblable aux propos de Denis et Torczon [21] : La Multi POLL est bâtie en anticipant que l'on trouve des succès dans tout ou partie des voisins de POLL. Inversement, la POLL en Oignon est construite en imaginant que les prochaines itérations seront des échecs, et génère donc les points correspondant à des échecs successifs. Ainsi, ces stratégies rendent possible une mise à l'échelle pour que l'optimisation se déroule en utilisant la totalité des ressources parallèles à disposition. Elles permettent de passer d'une configuration semblable à 2.5(c) vers une configuration de type 2.5(b) en augmentant le nombre de points candidats, et donc le nombre d'évaluations à effectuer. La POLL Enrichie a pour but d'explorer l'intérieur du cadre de sonde. Les trois stratégies Multi, Oignon et Enrichie établissent une structure à donner aux points générés dans $\mathbb{R}^n$ autour de x^k, mais, pour Oignon et Enrichie, le choix du nombre de points à générer à chaque itération est libre. La section (3.3) propose un cadre algorithmique pour la gestion du nombre de points de sonde à chaque itérations. Enfin, la section 3.4 expose le problème d'ordonnancement des évaluations rencontré lors de l'exécution de l'algorithme MADS. Quelques pistes sont développées dans le cas où l'utilisateur dispose de fonctions oracles permettant d'estimer le temps ou le coût de chaque évaluation.

3.1 Cadre de travail pour l'étape de POLL

Comme vu dans la section 2.2, l'étape de POLL n'est pas beaucoup plus contraignante que l'étape de SEARCH pour la génération de points. Un choix qui a été fait dans [2] est d'utiliser des directions orthogonales ajoutées au centre de POLL courant x^k et de projeter le résultat sur le maillage M^k. Le mécanisme de POLL classique est exposé dans l'algorithme 3 :
h_j est la j-ième colonne de H et ROUND(.) arrondit les coordonées d'un vecteur à l'entier le plus proche. $\mathbb{D}$ forme une base positive de l'espace et chacune des coordonnées de ses éléments est un multiple du pas de maillage δ^k. Le mécanisme de génération de points de

Algorithm 3 `POLL` classique - Génération de points à l'itération k de `MADS`

[0]. Initialisation :

δ^k : Pas de maillage.
Δ^k : Pas de cadre.
x^k : Itéré courant.
$\mathbb{D} \leftarrow \varnothing$: Ensemble de directions de `POLL`.
$\mathbb{P} \leftarrow \varnothing$: Ensemble de points de `POLL`.

[1]. Génération de directions de `POLL` :

Sélectionner $u \in \mathbb{R}^n$ non nulle.
Poser $v = \frac{u}{\|u\|}$.
Poser $H = I - 2vv^\top$.

$$\mathbb{D} \leftarrow \left\{ \pm \delta^k \text{ROUND}\left(\frac{\Delta^k}{\delta^k} \frac{h_j}{\|h_j\|_\infty} \right) \ : \ j \in [\![1, n]\!] \right\} \subset \delta^k \mathbb{Z}^n.$$

[2]. Génération de points de `POLL` :

$\mathbb{P} \leftarrow \{x^k + d \ : \ d \in \mathbb{D}\}$.
RETOURNER $\mathbb{P}$.

`POLL` classique peut être considéré comme la fonction :

$$\begin{aligned} \text{CLASSICALPOLL} : \mathbb{R}^*_+ \times \mathbb{R}^*_+ \times \mathbb{R}^n &\to \mathbb{R}^n \\ (\delta^k, \Delta^k, x^k) &\mapsto \mathbb{P} \subset F^k \end{aligned}$$

3.2 Génération de points lors de l'étape de POLL

Les stratégies Multi, Oignon et Enrichie permettent de générer des points de POLL en plus grande quantité que ce qui est proposé dans le fonctionnement de NOMAD 4. Avec plusieurs structures de construction, on espère obtenir des différences de performances dans la résolution de $(\mathcal{P})$ suivant la stratégie utilisée.

3.2.1 Multi-POLL

La Multi POLL fournit une charge de travail aux processeurs disponibles en générant des voisins aux voisins de POLL. Sur le centre de POLL x_k, q directions de POLL primaires $d^1, .., d^q$ sont générées. Ces directions produisent q centres de POLL secondaires $y^1, .., y^q$. Pour chaque centre de POLL secondaire $y^i = x^k + d^i$, q_i directions de POLL secondaires sont générées, dans un cadre de taille $\Delta^{k,i}$, centré en y^i. les directions de POLL secondaires permettent de construire les points $y^{i,j},\ j \in [\![1, q_i]\!]$. À une étape de Multi POLL, $q + \sum_{i=1}^{q} q^i$ candidats sont produits. Conserver des directions de sonde et des tailles de cadre identiques entre la sonde principale et la sonde secondaire engendre de nombreux doublons. La version implémentée dans NOMAD 4, est fixée avec $\Delta^{k,i} = \Delta^k$ et $q = q_i = 2n$. Tous les paramètres de cadre de sondes secondaires sont fixés égaux à Δ^k, ainsi que le nombre de points de sonde échantillonnés dans chaque cadre à $2n$. Ce choix peut sembler arbitraire, mais sans plus d'informations sur le problème, il ne semble pas pertinent de s'attarder plus sur ces paramètres. L'algorithme 4 fournit la procédure à suivre pour pouvoir générer des points de Multi-POLL.

Algorithm 4 Multi-POLL - Génération de points à l'itération k de MADS

[0]. Initialisation :
- δ^k : Paramètre de maillage.
- Δ^k : Pramètre de cadre.
- x^k : Meilleur point candidat.
- $\mathbb{P} \leftarrow \varnothing$: Ensemble de points de sonde.

[1]. Génération des points de sonde primaire :
- $\mathbb{P} \leftarrow \text{CLASSICALPOLL}(\delta^k, \Delta^k, x^k)$

[2]. Génération des points de sonde secondaire :
- $T \leftarrow \varnothing$.
- **Pour** $y \in \mathbb{P}$ **Faire** :
  - $T \leftarrow T \cup \text{CLASSICALPOLL}(\delta^k, \Delta^k, y)$.
- **Fin**
- $\mathbb{P} \leftarrow \mathbb{P} \cup T$.
- **RETOURNER** $\mathbb{P}$.

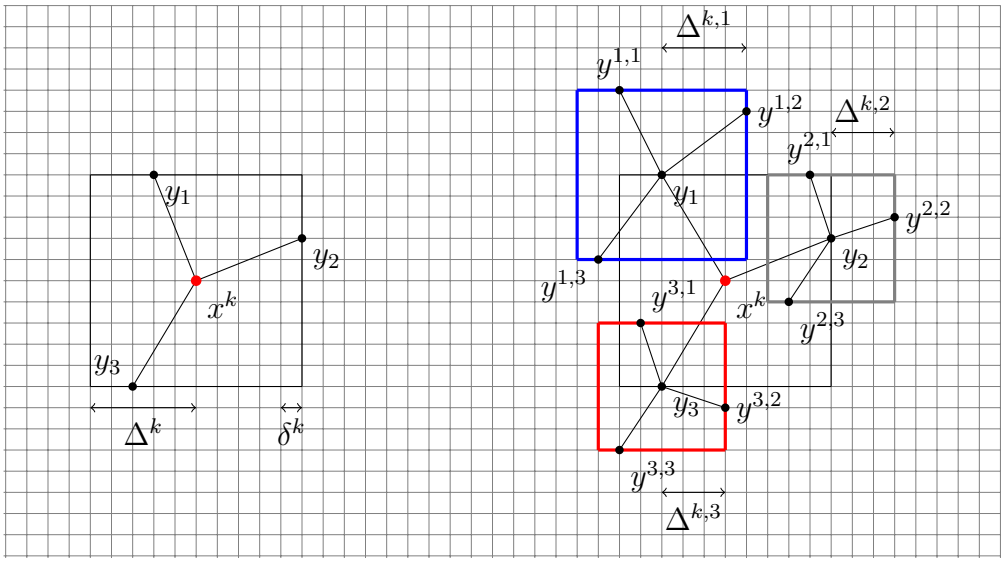

Figure 3.1 Comparaison entre POLL classique et Multi POLL avec le même nombre de directions dans chacune des sondes secondaire : $q = q_1 = q_2 = q_3 = 3$, et $\Delta^{k,1} = 4\delta^k, \Delta^{k,2} = \Delta^{k,3} = 3\delta^k$

3.2.2 Sonde locale en Oignon

Cette stratégie imite le comportement de MADS lorsque x^k est un minimum local, c'est-à-dire quand les échecs de POLL se succèdent. Il est supposé que le succès obtenu à l'étape k conduit à une suite d'échecs des étapes $k+1, ..., k+c-1$ d'une étape de POLL classique. Par conséquent, des points sont sélectionnés sur les cadres de POLL $F^k, ..., F^{k+c-1}$. Ces cadres, dont les paramètres respectifs sont $\Delta^k, \Delta^{k+1}, ..., \Delta^{k+c-1}$), sont tous centrés en x^k sur le maillage M^k. Ainsi, en supposant que les ressources nécessaires soient disponibles, si chaque étape de POLL génère n_p points, cette méthode permet de générer $c \times n_p$ points. Sous l'hypothèse que le succès de l'étape k conduit effectivement à $c-1$ raffinements successifs, et que Δ^k et δ^k sont mis à jour de manière adéquate, le ratio entre le nombre d'évaluations effectuées aux étapes de POLL classiques équivalentes menées séquentiellement et le nombre d'évaluations effectuées de la POLL en oignon est de c. La figure 3.2 donne un aperçu d'une POLL en oignon pour $c = 3$. L'algorithme 5 donne une description détaillée de la stratégie de POLL en oignon. La version implémentée dans NOMAD 4 utilise l'intervalle $[1, \frac{\Delta^k}{\delta^k}] \subset \mathbb{R}$. Cet intervalle est découpé en $c-1$ sous intervalles de longueur identiques, et leur bornes sont projetées sur $\mathbb{N}$, donnant lieu à c valeurs distinctes lorsque $c \leq \frac{\Delta^k}{\delta^k}$

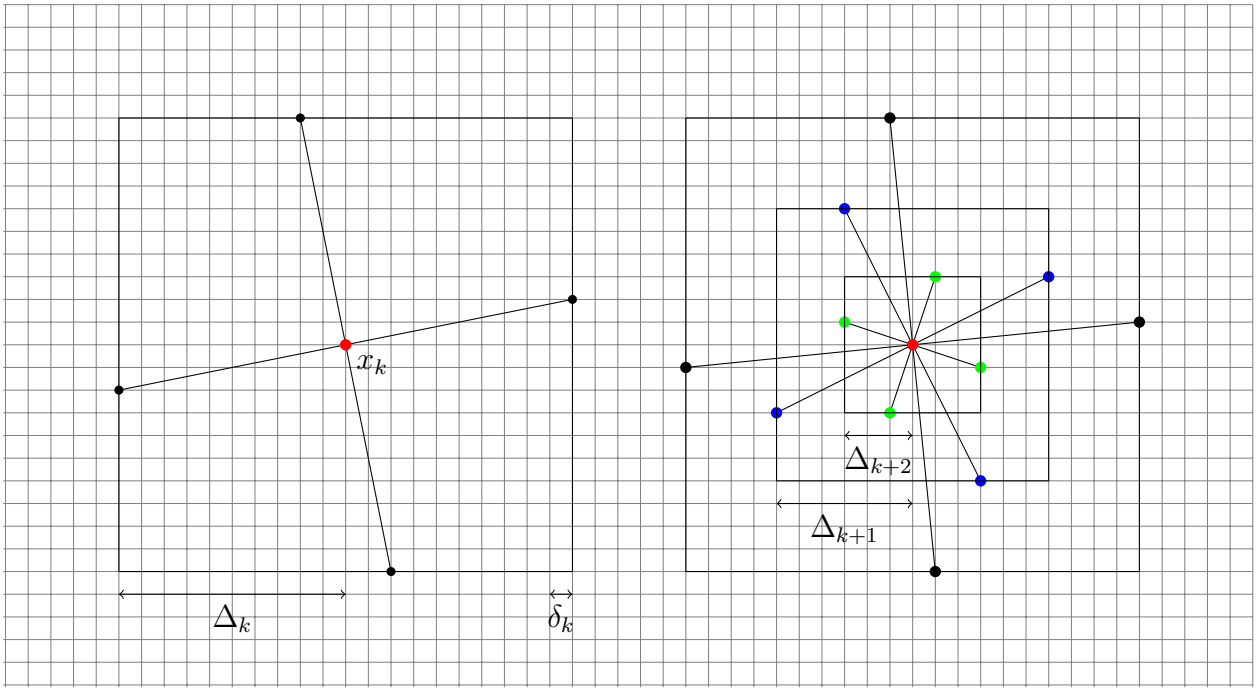

Figure 3.2 Des directions sont construites sur les cadres de sondes correspondant à deux réductions successives de Δ^k.

Algorithm 5 `POLL` en oignon - Génération de points à l'itération k de `MADS`

[0]. Initialisation :

δ^k	: Paramètre de maillage.
Δ^k	: Paramètre de cadre.
x^k	: Itéré courant.
c	: Nombre de couches ($c \leq \frac{\Delta^k}{\delta^k}$).
$\mathbb{P} \leftarrow \varnothing$	: Ensemble de points de sonde.

[1]. Generation des sous cadres :

Sélectionner $\Gamma \subseteq [\![1, \frac{\Delta^k}{\delta^k}]\!]$, $|\Gamma| = c$.

[2]. Generation des points de sonde :

Pour $\gamma \in \Gamma$ **aire** :

$\mathbb{P} \leftarrow \mathbb{P} \cup \text{CLASSICALPOLL}(\delta^k, \gamma\delta^k, x^k)$.

Fin

RETOURNER $\mathbb{P}$.

3.2.3 Sonde locale Enrichie

Dans MADS, le choix des directions de POLL est flexible, les suivants sont implémentés dans NOMAD 3 :

— $n+1$-MADS : Sonde l'espace suivant $n+1$ directions sur le cadre de POLL,

— $2n$-MADS : Sonde l'espace suivant $2n$ directions orthogonales sur le cadre de POLL.

Cependant, toutes ces directions ont au moins une coordonnée dont la valeur est Δ^k. Cela provient de la construction de $\mathbb{D}^k$ exposée en section 2.2 :

$$\mathbb{D}^k = \left\{ \pm \delta^k \text{round}\left(\frac{\Delta^k}{\delta^k} \frac{h_i}{\|h_i\|_\infty} \right) \ : \ i \in [\![1, n]\!] \right\} \tag{3.1}$$

En effet, dans cette définition, $\left\| \frac{h_i}{\|h_i\|_\infty} \right\|_\infty = 1$ donc, $\frac{h_i}{\|h_i\|_\infty}$ a au moins une de ses composantes égale à $+1$ ou -1. Une solution pour obtenir des points de sonde aléatoirement distribués dans F^k est de multiplier chacune des directions $\frac{h_i}{\|h_i\|_\infty}$ par un scalaire ξ_i choisi de manière aléatoire dans $[0, 1]$. Cette stratégie propose de construire des points de sonde en plus grande quantité en générant plusieurs directions aléatoires v^k, afin de construire plusieurs matrices de Householder. Une fois ces directions construites, il est possible de créer des points de sonde comme sur le deuxième dessin de la figure 3.3, en conservant les colonnes de H de norme unitaire. Il est aussi possible de multiplier chacune de ces directions par un coefficient aléatoire, comme décrit plus haut, et représenté sur le troisième dessin de la figure 3.3. Sur la figure 3.3, chaque couleur représente un ensemble de directions générées à l'aide d'une direction v^k et de l'opérateur de Householder. L'algorithme 6 détaille le fonctionnement de la POLL Enrichie. La distribution de probabilité qui permet de générer ξ influe sur la répartition des points. En effet, si les ξ_i sont échantillonnés suivant une loi uniforme sur $[0, 1]$ alors les points de sonde sont uniformément répartis dans F^k. Si au contraire, les ξ_i sont échantillonnés suivant une loi non uniforme, certains zones de F^k seront privilégiées. Enfin, changer les bornes d'échantillonage des ξ_i agit sur la répartition des points. Si $a > 1$ cela donne accès à des points hors du cadre de POLL F^k défini dans la section 2.2. Si $b > 0$ cela interdit l'accès à des points trop proche de x^k. Fixer $a = b = 1$ conduit à obtenir plus de points, tous sur la frontière du cadre.

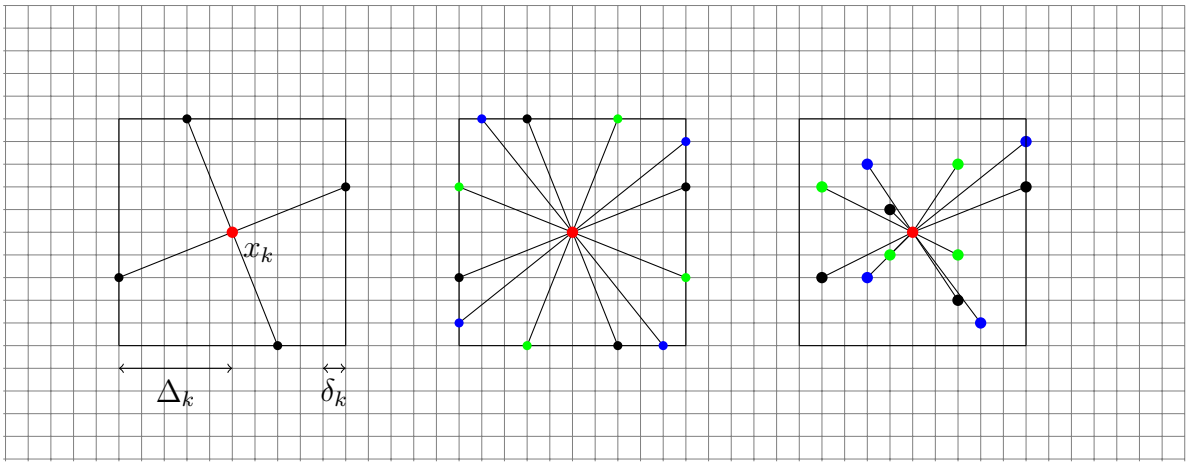

Figure 3.3 De gauche à droite : POLL classique, POLL enrichie sur le cadre, poll enrichie sur et à l'intérieur du cadre.

Algorithm 6 POLL Enrichie - Génération de points à l'itération k de MADS

[0]. Initialisation :

Δ^k : Pas de cadre de POLL.
δ^k : Pas de maillage.
q : Nombre de bases positives à générer.
x^k : Itéré courant.
a, b :bornes d'échantillonage.
$\mathbb{D} \leftarrow \varnothing$: Ensemble de directions de POLL.
$\mathbb{P} \leftarrow \varnothing$: Ensemble de points de POLL.

[1]. Génération de direction de POLL :

Pour $i \in [\![1, q]\!]$ **faire** :

Sélectionner $u \in \mathbb{R}^n$ non nulle.
Poser $v = \frac{u}{\|u\|_2}$.
Poser $H = I - 2vv^\top$.
Sélectionner $\xi = (\xi_1, ..., \xi_{2n}) \in [a, b]^{2n}$ de manière aléatoire.

$$\mathbb{D} \leftarrow \mathbb{D} \cup \left\{ \delta^k \text{round}\left(\frac{\Delta^k}{\delta^k} \frac{h_j}{\|h_j\|_\infty} \xi_j \right) \ : \ j \in [\![1, n]\!] \right\}$$
$$\cup \left\{ -\delta^k \text{round}\left(\frac{\Delta^k}{\delta^k} \frac{h_j}{\|h_j\|_\infty} \xi_{n+j} \right) \ : \ j \in [\![1, n]\!] \right\}.$$

Fin

[2]. Génération de points de POLL :

$\mathbb{P} \leftarrow \{x^k + d \ : \ d \in \mathbb{D}\}$.
RETOURNER $\mathbb{P}$.

3.3 Intensification dynamique lors de l'étape de POLL

Concernant les stratégies dont le nombre de points est modifiable à chaque itération (Oignon et Enrichie), considérons un historique des échecs successifs (2.4) défini comme suit :

$$e^0 = \begin{cases} 1 & \text{Si l'étape de POLL à l'itération 0 est un échec,} \\ 0 & \text{sinon.} \end{cases}$$

$$e^k = \begin{cases} e^{k-1} + 1 & \text{Si l'étape de POLL à l'itération } k \text{ est un échec,} \\ 0 & \text{sinon.} \end{cases}$$

Ainsi, lorsque l'étape de POLL échoue à améliorer la fonction objectif, «on regarde plus près» : réduction du cadre de sonde, et «plus en détail» : augmentation du nombre de points de sonde. Soit la fonction :

$$g : ((e^\ell)_{1\leq\ell\leq k-1}, n_p^{max}) \mapsto n_p^k$$

qui permet de décider du nombre de points à générer n_p^k à l'étape de POLL k connaissant le comportement lors des étapes de POLL précédentes $(e^\ell)_{1\leq\ell\leq k}$ et le nombre de points maximum à générer n_p^{max}.

Alors, pour les POLL en Oignon et Enrichie :

$$\textsc{DynamicPoll}(\delta^k, \Delta^k, x^k, n_p^k) = \mathbb{P}^k,$$

avec :

$$n_p^k = g((e^\ell)_{1\leq\ell\leq k-1}, n_p^{max}).$$

Par exemple :

$$g((e^\ell)_{1\leq\ell\leq k-1}, n_p^{max}) = \min(e^{k-1}, n_p^{max}) : \text{intensification linéaire,}$$
$$g((e^\ell)_{1\leq\ell\leq k-1}, n_p^{max}) = \min(2^{e^{k-1}}, n_p^{max}) : \text{intensification exponentielle.}$$

Aussi, avec g définie comme ci dessus, si l'expression de e^k est changée en :

$$e^k = \begin{cases} e^{k-1} + 1 & \text{Si l'étape de POLL à l'itération } k \text{ est un échec,} \\ \max\{e^{k-1} - 1, 0\} & \text{Sinon,} \end{cases}$$

n_p^k est obtenu sans oublier l'intensification qui a été nécessaire aux itérations $k-1, k-2, ...$ pour obtenir un succès à l'itération k. En d'autres mots, si le nombre de points de sonde a été augmenté à plusieurs reprises avant d'arriver à un succès, il n'est pas directement réinitialisé à $n_p^k = 2n$ au premier succès, car rien ne permet d'affirmer que les succès futurs seront plus faciles à obtenir. En quelque sorte, les efforts consacrés à la sonde locale sont relâchés, mais de manière modérée. Par la suite, on parlera de POLL dynamique sans mémoire pour désigner une POLL dynamique réalisée avec la première définition de e^k, et de POLL dynamique avec mémoire pour désigner une POLL dynamique réalisée à l'aide de la seconde définition de e^k.

Ce qui a été exposé ci dessus avec n_p^k peut aussi s'appliquer au nombre de points de l'étape de SEARCH : n_s^k. D'après la figure 2.2, l'étape de POLL n'est effectuée que lorsque l'étape de SEARCH est un échec. Ainsi, il semble pertinent de déterminer n_s^k et n_p^k de manière complémentaire : lorsque n_p^k est augmenté, n_s^k est diminué, et inversement. Cela permet de consacrer des évaluations à la diversification de la SEARCH ou à l'intensification de la POLL selon que les itérations précédentes sont majoritairement des échecs ou des succès. La stratégie de SEARCH exposée en section 2.7 se prête bien à un tel fonctionnement.

Il semble aussi pertinent de décider de n_p^k en fonction de la quantité de ressources disponibles.

3.4 Ordonnancement optimal de la pile d'évaluations

Soit $(\mathcal{P})$ un problème d'optimisation défini en introduction (1) et $x \in \mathbb{R}^n$. On note respectivement $\sigma(x)$ et $\tau(x)$ le coût et le temps nécessaires à un processeur pour effectuer une évaluation en x de f et des c_j. Comme vu en introduction, une boite noire est un processus dont le fonctionnement nécessite certaines ressources disponibles en quantité limitée, ou tout simplement qui sont coûteuses.

Le développement qui suit est pertinent dans la mesure où l'utilisateur a accès aux fonctions oracles $\sigma(x)$ et $\tau(x)$. Cela peut être le cas lorsque l'utilisateur sait ce que représente les x_i, et qu'il est capable d'anticiper facilement le fonctionnement de la boite noire qui en découle.

Le coût dépend de la nature de la boite noire et de l'intérêt de l'utilisateur. Par exemple, $\sigma(x)$ peut être le nombre d'opérations effectuées, l'énergie consommée, la quantité d'un produit chimique utilisé, le nombre de CPUs nécessaires, etc. Tandis que le temps est le temps écoulé,

celui que pourrait mesurer l'utilisateur avec sa montre, entre le début et la fin de l'exécution du processus par un processeur. Sans information sur le fonctionnement interne de la boite noire, rien ne permet à priori de corréler coût et temps. Deux évaluations peuvent avoir le même coût, et des temps différents, et inversement. Cependant, dans le cas où la boite noire est une simulation numérique exécutée sur une machine multi-CPUs, il semble cohérent de supposer qu'augmenter le nombre de CPUs diminue le temps $\tau(x)$. Cela dépend de l'implémentation de la boite noire. Si la boite noire est implémentée en séquentiel, augmenter le nombre de CPUs sur la machine n'aura aucun effet sur $\tau(x)$. De plus, même si la boite noire est implémentée en parallèle, il existe une limite au degré de parallélisme de cette implémentation (voir (2.4)), marquant le changement de sens de variation de $\tau(x)$: décroissant avant la limite, croissant après.

Comme spécifié à la définition (1), dans contexte présent, un processeur peut être un téléphone cellulaire, un banc d'essais en laboratoire, le simulateur de réseau électrique d'Hydro-Québec, une machine munie de 12 CPUs, etc. Nous possédons un nombre fixé $\rho \in \mathbb{N}^*$ de ces processeurs pour évaluer les points de $\mathbb{S}^k$ et de $\mathbb{P}^k$ avec f et les c_j. Rappelons les notations utilisées précédemment :

$$n_s^k = |\mathbb{S}^k|,\ n_p^k = |\mathbb{P}^k|.$$

La figure 2.5 montre que si $n_p^k < \rho$ et $n_s^k < \rho$ alors, l'ensemble des processeurs disponibles est utilisé en dessous de ses pleines capacités à l'itération k. Les stratégies proposées à la section 3.2 font en sorte que $n_s^k, n_p^k \geq \rho$. Pour une itération, le temps d'exécution se décompose comme suit :

$$T^k = \tau_{\text{maj}}^k + \tau_s^k + \tau_p^k.$$

Avec τ_{maj}^k le temps nécessaire au fonctionnement interne de `NOMAD`. Dans le cas séquentiel,

$$\tau_s^k = \sum_{x \in \mathbb{S}^k} \tau(x),$$
$$\tau_p^k = \sum_{x \in \mathbb{P}^k} \tau(x).$$

Dans le cas parallèle, avec $\rho > \max\{n_p^k, n_s^k\}$,

$$\tau_s^k = \max_{x \in \mathbb{S}^k} \tau(x),$$
$$\tau_p^k = \max_{x \in \mathbb{P}^k} \tau(x).$$

Souvent, lors de la résolution d'un problème réalise, l'hypothèse suivante est faite : $\tau^k_{\text{maj}} \ll \tau(x)$. Formalisons d'avantage la figure 2.5. A chaque itération, il est nécessaire de décider quels points doivent être évalués sur quel processeur. A chaque point x de $\mathbb{S}^k$ ou de $\mathbb{P}^k$, est associé un unique processeur i.Soit Π^k_s et Π^k_p les piles d'évaluations obtenues à l'itération k de `MADS` pour la `SEARCH` et la `POLL`. Les objets Π^k_s, Π^k_p et $\mathbb{S}^k$, $\mathbb{P}^k$ sont semblables : à chaque point x de $\mathbb{P}^k$ correspond une unique évaluation $e = (x, f(x), c(x))$. De ce fait, il est possible d'étendre les fonctions σ et τ sur ces quatre ensembles. On cherche alors à partitionner les piles d'évaluations et à assigner à chaque processeur une partition sur laquelle travailler, de sorte que la durée ou le coût total cumulé des évaluations soit minimal. Pour une pile d'évaluations Π, une p-partition de cette pile est une famille $\pi = (\pi^i)_{1\leq i\leq \rho}$ de sous ensembles de Π, telle que :

$$\pi^i \cap \pi^j = \varnothing, \quad \forall i \neq j \in [\![1, \rho]\!].$$
$$\bigcup_{i=1}^{\rho} \pi^i = \Pi.$$

Aux étapes de `SEARCH` et de `POLL`, les p-partitions $(\pi^i_s)_{1\leq i\leq \rho}$ et $(\pi^i_p)_{1\leq i\leq \rho}$ les construites à partir des piles d'évaluations Π^k_s et Π^k_p. Avec ces notations, dans le cas parallèle avec $\rho \leq \max(n_p, n_s)$, on a :

$$\tau^k_s = \max_{i\in[\![1,\rho]\!]} \sum_{e\in\pi^i_s} \tau(e),$$
$$\tau^k_p = \max_{i\in[\![1,\rho]\!]} \sum_{e\in\pi^i_p} \tau(e).$$

En d'autres mots, la durée des étapes de `SEARCH` et de `POLL` est le temps nécessaire au processeur qui va fonctionner le plus longtemps. Avec ces notations, certains des π^i_s ou des π^i_p peuvent être vide. Cela dépend des valeurs relatives de n_s, n_p et ρ.

Soit $\phi : \mathcal{P}(\Pi) \to \mathbb{R}^+$ [1], vérifiant $\phi(\pi_i) = 0$ si et seulement si $\pi_i = \varnothing$. La fonction ϕ mesure le temps ou le coût nécessaire à un processeur pour écouler la pile d'évaluation qui lui est attribué. Par exemple la fonction ϕ peut être : $\phi(\pi^i) = \sum_{e\in\pi^i} \tau(e)$ ou $\phi(\pi^i) = \sum_{e\in\pi^i} \sigma(e)$ ou encore $\phi(\pi^i) = |\pi^i|$. En notant $\coprod$ l'ensemble des ρ-partitions de Π, une ρ-partition $\pi^* =$

1. Ici, $\mathcal{P}(E)$ représente l'ensemble des parties de l'ensemble E

$(\pi^{i*})_{1\leq i\leq \rho}$ ϕ-optimale de Π est une ρ-partition vérifiant :

$$\max_{i\in[\![1,\rho]\!]}\phi(\pi^{i*}) = \min_{\pi\in\coprod}\left\{\max_{i\in[\![1,\rho]\!]}\phi(\pi^{i})\right\}. \tag{3.2}$$

Une telle ρ-partition existe. En effet, Π est fini. Donc, l'ensemble de ses ρ-partitions, $\coprod$, est fini. Donc l'ensemble $\left\{\max_{i\in[\![1,\rho]\!]}\phi(\pi^{i}) : \pi \in \coprod\right\} \subset \mathbb{R}^+$ est fini, il admet donc un plus petit élément, image de π^* par ϕ. Cependant, sans hypothèse supplémentaire, rien ne garantit l'unicité d'une ρ-partition ϕ-optimale. Un contre exemple trivial peut être obtenu lorsque toutes les évaluations ont la même durée et que le nombre d'évaluations est un multiple de ρ. Partitionner Π selon π^* et, pour tout $i \in [\![1, \rho]\!]$, effectuer les évaluations de π^i sur le processeur i, permet d'écouler la pile d'évaluations en un temps minimal.

Lorsque les fonctions oracles τ et σ sont accessibles, il est possible de construire π_s et π_p et ainsi d'accélérer l'optimisation tout en donnant de la priorité aux évaluations qui sont le moins coûteuses.

Le temps est une mesure qui n'est pas cumulative lorsqu'il s'agit d'effectuer des tâches de manière parallèle : Pour calculer la durée d'une exécution parallèle, il ne suffit pas d'additionner les durées de chacune des tâches. La durée d'une itération est affectée selon que les évaluations sont effectuées de manière parallèle ou non, et du degré de parallélisme utilisé. Le coût, au contraire, conserve son caractère additif. Par exemple, considérons que σ mesure l'énergie électrique nécessaire pour faire fonctionner un processeur lors d'une évaluation en x. L'énergie totale consommée est la somme des énergies consommées pour l'exécution de chacune des tâches. La métrique du temps permet de regrouper les évaluations par processeur, c'est à dire, attribuer un paquet d'évaluations π^i sur lequel le processeur i doit travailler, mais cette métrique ne donne pas l'ordre dans lequel le processeur i doit traiter les évaluations appartenant à π^i. En effet, chaque processeur travaille de manière séquentielle sur π^i, et, dans ce cas là, la métrique du temps est additive.

Supposons alors que l'utilisateur a à disposition Θ_0 unités de coût et Γ_0 unités de temps, et et qu'il souhaite mener la résolution de $(\mathcal{P})$ en respectant cette limite sur la consommation de ressources. Comment mener l'optimisation ? Faut-il s'arrêter lorsque l'une des deux ressources est épuisée ? Ou est-il possible d'élaborer une stratégie qui utilise l'information retournée par σ et τ ? Serait-il pertinent par exemple d'ordonner chacun des π^i à l'aide de σ ? Dans l'ordre croissant : pour prioriser les évaluations peu coûteuses. Dans l'ordre décroissant : pour prioriser les évaluations coûteuses.

Ce qui précède doit être reformulé dans les cas suivant. (1) Lorsque σ et τ ne fournissent pas

une information exacte, mais avec une erreur aléatoire e_σ et e_t. (2) Lorsque le contexte est asynchrone : à chaque pas de temps, de nouvelles évaluations sont ajoutées à la pile Π, est-il nécessaire de résoudre le problème posé ci dessus à chaque pas de temps ? (3) Lorsque les fonctions σ et τ dépendent du processeur utilisé ou de l'ordre dans lequel sont effectuées les évaluations : évaluer x_1 avant x_2 ou x_2 avant x_1 sur le même processeur aboutit à des valeurs de $\tau(x_1)$ ou $\sigma(x_1)$ différentes.

REMARQUE : Se référer au problème de bin-packing, avec un nombre de boite fixé à ρ et comme objectif de remplir le moins possible la plus pleine des boites. boite <=> processeur, volume remplit <=> temps/coût

CHAPITRE 4 Résultats théoriques et numériques

Les méthodes proposées en section 3, donnent des règles pour sélectionner des points à l'étape de `POLL` afin d'occuper les processeurs disponibles. Dans ce chapitre, l'objectif est de mesurer l'apport de ces règles par rapport à une version de `MADS` qui ne les utilisent pas, tant sur la meilleure valeur de f obtenue que sur l'évolution de la valeur de f pendant l'exécution de `NOMAD` 4. La section 4.1 présente les travaux de Moré et Wild [36] sur la comparaison d'algorithmes d'optimisation de boite noire. La section 4.2 présente une banque de 24 problèmes tests analytiques sans contraintes utilisées pour appliquer le stravaux esposées dans la section 4.1. Un problème réaliste, `STYRENE` [8], est également présenté, en vue d'effectuer des essais pour évaluer les performances des stratégies proposées dans un contexte réaliste. Dans la section 4.3, les résultats obtenus sur les problèmes analytiques sont présentés. Ces résultats comparent uniquement les stratégies de `POLL` : l'exécution de `NOMAD` 4 se fait uniquement à l'aide de la `POLL`. La `SEARCH` et les autres mécanismes d'aide à l'optimisation sont désactivés. La section 4.4 présente les résultats obtenus lors de la résolution de `STYRENE`, à l'aide de la `POLL` seule mais aussi à l'aide des autres mécanismes d'aide à la résolution. L'objectif est de mesurer l'influence des mécanismes autres que les stratégies de `POLL`, lorsque ces mêmes stratégies sont utilisées. La section 4.5 fait la synthèse des résultats obtenus. L'implémentation des stratégies est disponible à l'adresse suivante : `https://github.com/Guillaume5255/nomad.git` L'implémentation du programme d'obtention et de traitement des données est disponible à l'adresse suivante : `https://github.com/Guillaume5255/Benchmark_NOMAD.git` L'ensemble des données obtenues et des graphes s'y trouvent également. TODO : Mettre l'URL vers bbopt et non pas vers mon profil perso

4.1 Métriques de performance

Moré et Wild [36] introduisent des outils d'analyse comparative pour étudier un ensemble d'algorithmes d'optimisation sur un ensemble de problèmes. Dans cette sous-section, $\mathfrak{P}$ désigne un ensemble de problèmes, et $\mathfrak{A}$ désigne un ensemble d'algorithmes d'optimisation sans dérivées. Pour un algorithme $a \in \mathfrak{A}$, un problème $p \in \mathfrak{P}$, et une précision $\tau \in [0, 1]$, f^p désigne la fonction objectif du problème p, et $f_L^p := \min\{f^p(x_a^*) : a \in \mathfrak{A}\}$ la meilleure valeur de la fonction objectif obtenue, tout algorithme $a \in \mathfrak{A}$ confondu, lors de la résolution du problème

p. $N_{p,a}$ désigne la valeur d'un critère d'intérêt lorsque l'assertion

$$\frac{f^p(x^0) - f^p(x^k)}{f^p(x^0) - f_L^p} \geq (1 - \tau), \tag{4.1}$$

est satisfaite pour la première fois lors de l'exécution de a sur p. Par convention, si cette assertion n'est jamais satisfaite lors l'exécution de a sur p, $N_{p,a} = +\infty$. $N_{p,a}$ peut désigner un nombre d'évaluations, une durée, un nombre d'itérations, etc.

A partir de $N_{p,a}$, Moré et Wild [36] définissent la grandeur

$$r_{p,a} := \frac{N_{p,a}}{\min\{N_{p,\tilde{a}} : \tilde{a} \in \mathfrak{A}\}},$$

qui permet de rapporter la valeur $N_{p,a}$ à N_{p,a^*}, où a^* est l'algorithme qui est le plus performant sur p, c'est à dire, qui a vérifié l'assertion 4.1 pour une valeur du budget la plus petite.

Définition 15 (Profil de performance [36])**.** Le profil de performance de l'algorithme a est le graphe de la fonction :

$$\rho_a : \alpha \mapsto \frac{|\{p \in \mathfrak{P} : r_{p,a} \leq \alpha\}|}{|\mathfrak{P}|},$$

qui représente la proportion de problèmes résolus par a à l'aide de au plus α fois le budget du meilleur algorithme sur chaque problème.

Définition 16 (Profil de donnée [36])**.** Le profil de donnée de l'algorithme a est le graphe de la fonction :

$$d_a : \kappa \mapsto \frac{|\{p \in \mathfrak{P} : \frac{N_{p,a}}{n_p+1} \leq \kappa\}|}{|\mathfrak{P}|}, \tag{4.2}$$

qui représente le nombre de problèmes résolus par a en un budget d'au plus κ groupes de $n_p + 1$ évaluations.

Le terme $n_p + 1$ désigne le nombre de directions nécessaire à la construction d'un simplexe gradient (Audet et Hare [11], définition 9.5) en dimension n_p du problème p. Ce terme est présent pour prendre en compte le fait que plus la dimension du problème est grande, plus le budget nécessaire à l'algorithme a pour satisfaire (4.1) est grand. Ainsi, $\mathfrak{P}$ peut être constitué de problèmes de dimension différentes.

Soit le cas où $N_{p,a}$ désigne le nombre d'évaluations de f^p effectuées par a pour que (4.1) soit satisfait. Les méthodes d'analyse comparative proposées par Moré et Wild [36] ne permettent pas de démarquer deux algorithmes de recherche directe sans stratégie opportuniste,

dont le second est une version parallèle synchrone du premier. En effet, ces méthodes d'analyse comparative permettent d'évaluer l'efficacité avec laquelle un algorithme fait usage de l'information qu'il extrait du problème à l'aide du budget disponible.

Les performances d'une exécution parallèle sont plus subtiles à mesurer que les performances d'une exécution séquentielle. Il est nécessaire de distinguer la performance en terme d'amélioration de la fonction objectif, propre à l'algorithme, et la performance en terme d'utilisation de ressources, propre au programme qui implémente l'algorithme. La première est en quelque sorte une métrique de l'efficacité de la stratégie décrite par l'algorithme, la seconde est relative aux versions parallèles et séquentielles du programme qui implémentent le même algorithme : langage, paradigme de programmation utilisé, etc. Par exemple, si deux résolutions de $\mathcal{P}$ sont menées avec `MADS`, identiques, sans stratégie opportuniste, à l'exception que pour la première résolution, les évaluations sont effectuées en série, pour la deuxième résolution, les évaluations sont effectuées en parallèle. Le gain observable ne se compte pas en nombre d'évaluations mais en temps nécessaire à passer au travers des étapes de `SEARCH` et de `POLL`. De plus, par hypothèse, l'opportunisme n'est pas activé. Donc, toutes les évaluations correspondant à une étape de `MADS` sont effectuées avant de passer à l'étape suivante. La parallélisation de `MADS` n'influe pas les performances de l'amélioration de la fonction objectif pendant la résolution, mais influe sur la manière dont sont utilisées les ressources et donc sur le temps nécessaire à minimiser convenablement la fonction. En effet, à chaque étape, dans les deux cas (parallèle ou séquentiel) l'algorithme termine toutes les évaluations pour prendre une décision. Cette décision ne dépend que des valeurs retournées par la boite noire, et non pas de leur ordre d'obtention. Ce n'est plus le cas si l'opportunisme ou l'asynchronisme sont utilisés.

Souvent, lors de la résolution d'un problème industriel, la proportion de ressources informatique consommées par `NOMAD` 4 lors de la résolution est négligeable devant la proportion de ressources consommées par les évaluations.

De plus, il ne semble pas judicieux de vouloir mesurer l'occupation des CPUs, car cela dépend du processus que modélise la boite noire. Par exemple, une boite noire qui fait appel à des communications réseau (par exemple pour obtenir la météo dans une région du monde) introduit inévitablement des délais non déterministes lors de chacune des évaluations. `NOMAD` 4 peut être exécuté sur un seul CPU, car les opérations propre à son fonctionnement sont souvent peu coûteuses en comparaison avec le fonctionnement d'une boite noire.

En résumé, pour comparer des algorithmes itératifs sujets à une exécution parallèle, il semble nécessaire de mener l'analyse comparative suivant deux axes :

— Amélioration de la fonction objectif.
— Occupation des ressources parallèles.

Le premier point est l'objet des sections 4.3 et 4.4. Les profils de données comme définit précédemment comptent la proportion de problèmes résolus après un certain nombre d'évaluations de la fonction objectif. Dans un contexte parallèle, il est pertinent de parler en terme d'itération, car dans le cas d'une exécution synchrone, une itération correspond à un cycle d'occupation des ressources parallèles. Ainsi, il semble pertinent de revoir la définition de profils de données comme suit :

Définition 17 (Profil de donnée en itération (inspiré de Moré et Wild [36]))**.** Le profil de donnée en itération de l'algorithme a est le graphe de la fonction :

$$d_a : \kappa \mapsto \frac{|\{p \in \mathfrak{P} : N_{p,a} \leq \kappa\}|}{|\mathfrak{P}|}, \tag{4.3}$$

qui représente le nombre de problèmes résolus par a en un budget d'au plus κ itérations.

Dans cette définition, $N_{p,a}$ représente le nombre d'itérations de l'algorithme a nécessaires pour que l'équation (4.1) soit satisfaite pour le problème p.

4.2 Description de l'ensemble de problèmes

Pour une exécution, f^* désigne la meilleure valeur de f obtenue et x^* est tel que $f^* = f(x^*)$. Deux types de problèmes sont utilisés pour l'obtention des résultats numériques qui suivent :

— Des problèmes sans contraintes dont l'objectif est une fonction analytique.

— Des problèmes dont l'objectif et les contraintes sont issus d'une boite noire.

Les problèmes de la première catégorie sont appelés des problèmes analytiques, et les problèmes de la seconde catégorie, des problèmes réalistes.

4.2.1 Problèmes analytiques

Cette catégorie se constitue de 24 problèmes analytiques de minimisation sans contraintes [23]. Pour chaque problème, $f^* = 0$. L'expression de la fonction de Weierstrass (problème n°16) a été modifiée pour permettre cela en remplaçant le cube par un carré dans l'expression de f. $x^* \in [-5, 5]^n$. Les contrainte de borne sont : $\forall i \in [\![1, n]\!],\ -5 < x_i < 5 : \mathcal{X} = [-5, 5]^n$. Chaque problème est généré en dimension $n \in \{2, 4, 8, 16, 32, 64\}$. Il est possible de générer autant d'instances que souhaité de chaque problème, en fournissant une graine aléatoire différente lors de la construction. Chaque problème est instancié 5 fois pour un total de $5 \times 24 = 120$ problèmes analytiques par dimension. Les problèmes sont déterministe. Modifier la graine aléatoire revient à modifier les grandeurs définissant le problème, telle que le point de départ, x^0. L'avantage des problèmes analytique est leur faible demande en ressources

de calcul pour effectuer une évaluation, et donc pour l'obtention de résultats relativement complets.

4.2.2 Problèmes réalistes

Nous avons à notre disposition une simulation informatique modélisant une production de styrène [8] dont le code source est disponible à `https://github.com/bbopt/styrene`. Cette simulation prend 8 paramètres en entrée, et retourne un objectif ainsi que 11 contraintes dont 4 binaires et 7 relaxables ([34], [11], chapitre 12). Pour ce problème, $\mathcal{X} = [0, 100]^8$ et $f^* = -3.37101 \times 10^7$, obtenue de manière numérique par S. Kojtych.

Pour obtenir plusieurs instances du problème `STYRENE`, la valeur du point initial est changée. Pour cela, des points réalisables sont échantillonnés dans $\mathcal{X}$ avec pour contraintes que pour deux points différents, toutes les coordonnées soient différentes pour créer suffisamment de diversité. Ainsi 10 instances du problème sont générées.

4.3 Résultats numériques obtenus lors de l'exécution sur les problèmes analytique

Dans cette section, les résultats numériques en terme de performance d'amélioration de la fonction objectif sont exposés. L'exploitation des données collectées est réalisée grâce à la méthodologie de Moré et Wild [36] présentée en section 4.1.

4.3.1 Détails des valeurs des paramètres algorithmiques et de l'environnement numérique de tests

Les résultats de cette section sont obtenus dans le contexte suivant. Quatre stratégies de `POLL` sont testées : la `POLL` classique, la Multi `POLL`, la `POLL` en Oignon et la `POLL` Enrichie. Tous les pas de cadre de sonde secondaires de la Multi `POLL` sont égaux au pas de cadre de sonde principale ($\forall i \in [\![1, n]\!],\ \Delta^{k,i} = \Delta^k$), et chaque `POLL` principale et secondaire comporte $2n$ directions orthogonales. Pour la `POLL` en Oignon, le nombre de couches est fixé à $c \in \{2, 4, 8, 16, 32, 64\}$. Pour la `POLL` Enrichie, le nombre bases positives est fixé à $q \in \{2, 4, 8, 16, 32, 64\}$. Lors de l'exécution de `MADS`, seule l'étape de `POLL` est effectuée. Lors de chaque étape de `POLL`, tous les points sont évalués : la stratégie opportuniste n'est pas utilisée. `NOMAD` 4 effectue 500 itérations au plus (donc 500 étapes de `POLL`). Il y a également un critère d'arrêt sur la taille δ^k du maillage : $\delta^k \geq 10^{-13}$, où 10^{-13} représente la précision machine par défaut utilisée dans `NOMAD` 4(paramètre `DEFAULT_EPSILON`). Il n'y a pas de critère d'arrêt imposé sur le nombre d'évaluations. Le maillage anisotropique n'est pas utilisé.

L'environnement numérique pour l'exécution des problèmes analytiques est le suivant. Une évaluation de l'objectif d'un problème analytique prend entre 0.001 et 0.00001 secondes en dimension 1000, et des essais sont réalisés jusqu'en dimension 32. `NOMAD` 4 est utilisé sur un ordinateur muni de 8 Go de mémoire vive et d'un CPU `Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz`. Pour chaque optimisation, le nombre de fils d'exécution `OpenMP` alloués à `NOMAD` 4 est réglé à 1, pour des raisons semblables à celles exposées en section 2.4 : ici, le but n'est pas de mesurer l'influence de la programmation parallèle sur le temps d'exécution, mais plutôt de mesurer l'influence des stratégies de `POLL` sur l'amélioration de la fonction objectif.

4.3.2 Intérêt de l'exploration de l'intérieur du cadre

Comme vu dans la section 3.2.3, la disposition des points générés par la `POLL` Enrichie diffère selon les valeurs minimales et maximales, a et b, imposées sur la valeur des ξ_i. Poser $a = b = 1$ produit des points de `POLL` sur la frontière du cadre. Au contraire, $a = 1$ et $b = 0$ conduit à des points situés strictement à l'intérieur du cadre. Des profils de données en évaluation sont tracés en dimension $2, 4, 8, 16$ et 32 dans les deux configurations précédentes. La figure 4.1 représente les profils de données de la `POLL` Enrichie pour des points générés sur, et à l'intérieur du cadre de sonde, toutes dimensions confondues. La figure 4.1 montre qu'explorer l'intérieur du cadre de sonde dégrade les performances.

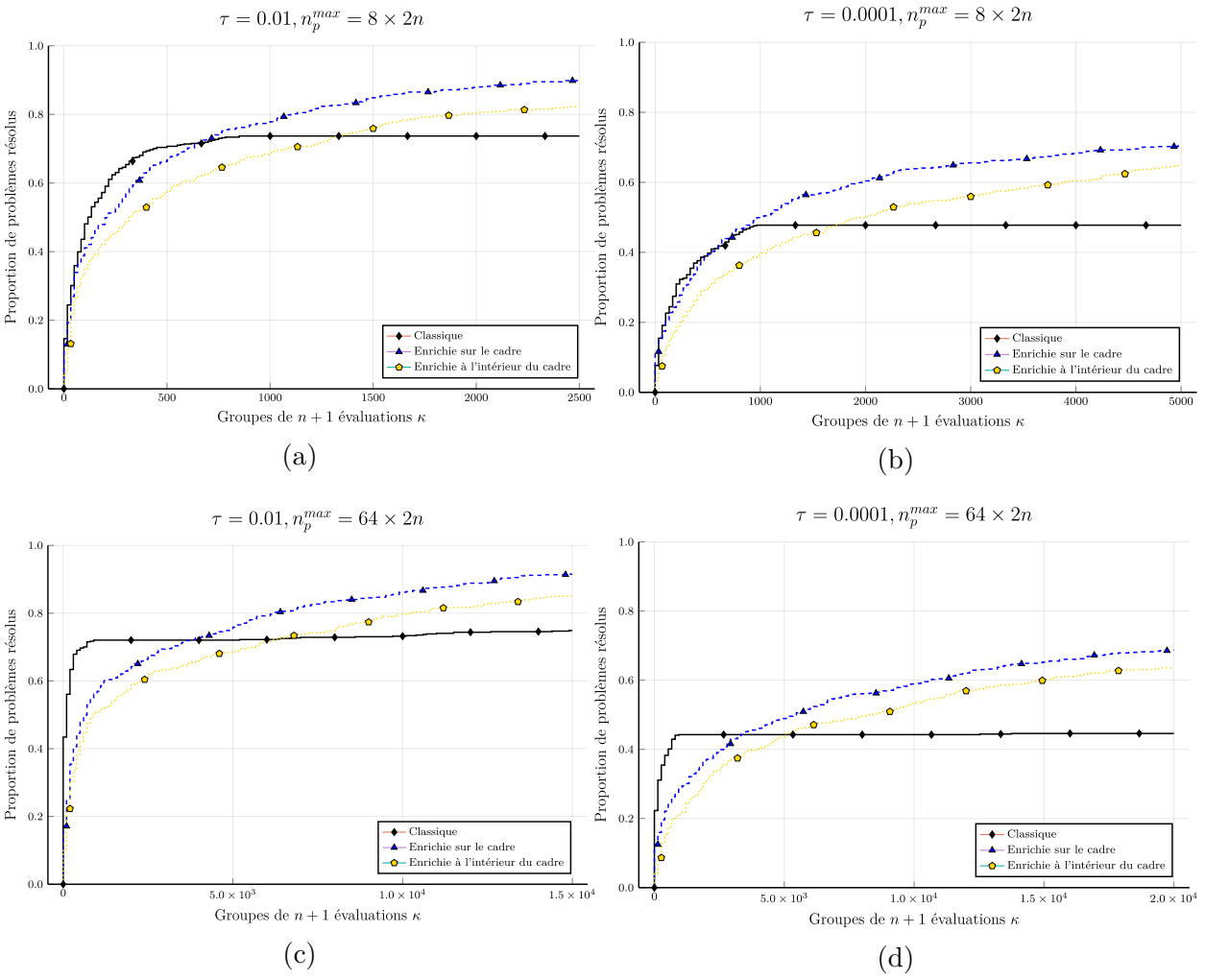

Figure 4.1 Profils de donnée en évaluation de la stratégie Enrichie lorsque les points de sonde sont uniquement sur le cadre ou aussi à l'intérieur du cadre, toutes dimensions confondues.

Cependant, les profils tracés pour des problèmes de même dimension montrent que les performances peuvent être améliorées par l'exploration de l'intérieur du cadre. C'est le cas en dimension 8. La figure 4.2 représente les profils de données de la `POLL` Enrichie pour des points générés sur, et à l'intérieur du cadre de sonde en dimension 8. De plus, l'augmentation de la dimension semble améliorer les performances de l'exploration de l'intérieur du cadre, et cet effet semble d'autant plus marqué que la précision τ est élevée, et que le nombre de points générés n_p^{max} est grand. Dans ce qui suit, les paramètres de bornes de la `POLL` enrichie sont fixés à $a = 1$ et $b = 0$.

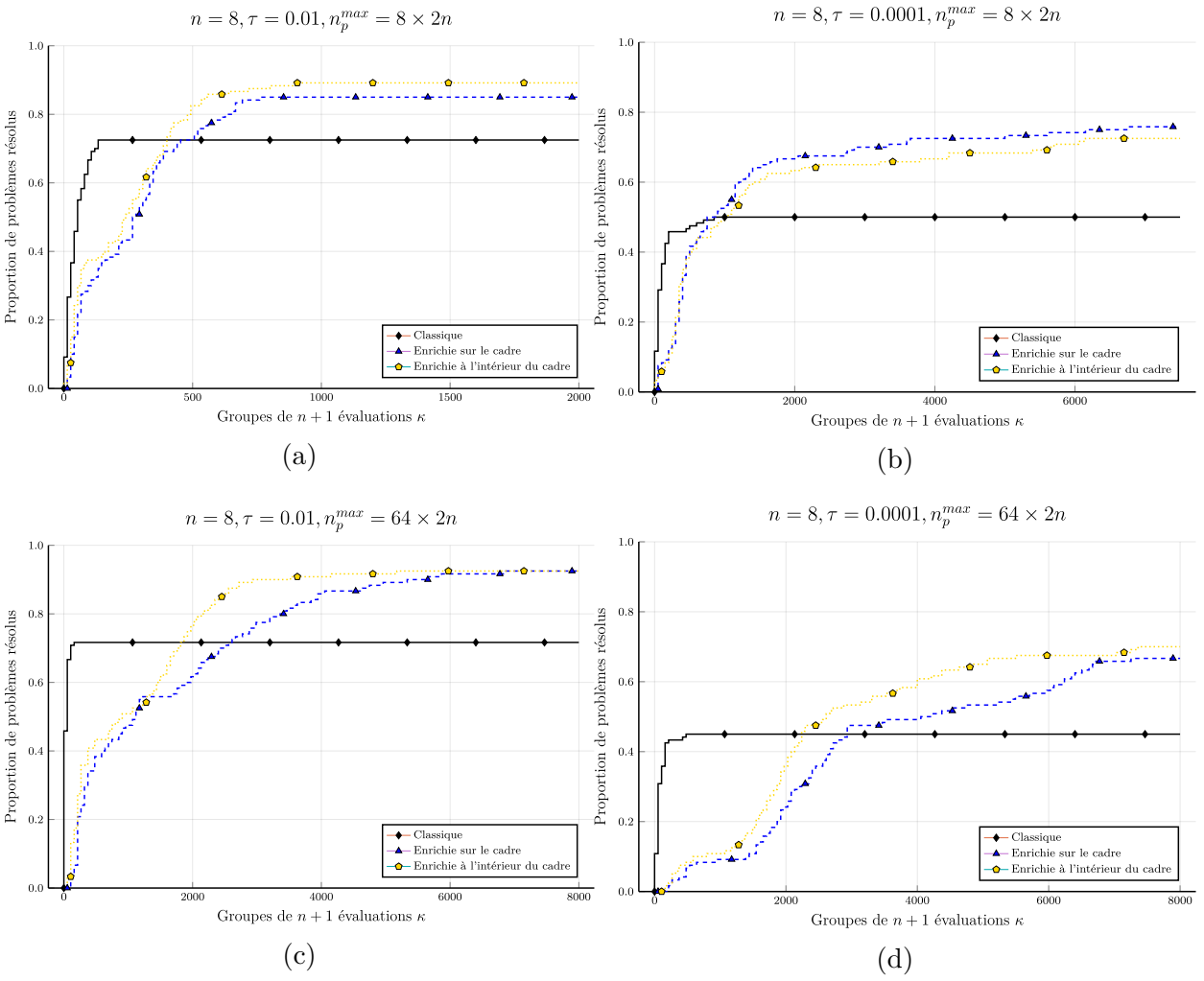

Figure 4.2 Profils de donnée en évaluation de la stratégie Enrichie en dimension 8 lorsque les points de sonde sont uniquement sur le cadre ou aussi à l'intérieur du cadre.

4.3.3 Influence du nombre de points et de la géométrie sur la valeur optimale

Comme mentionné dans la section 4.2, tous les problèmes analytiques ont pour valeur optimale $f^* = 0$. La figure 4.3 représente la valeur optimale moyenne obtenue en fonction de la stratégie de `POLL` utilisée et de n_p^{max}. A titre de comparaison, les résultats d'une recherche par hypercube latin sur $\mathcal{X}$ avec autant de points que ce qui serait généré dans le cas de la sonde classique pendant 500 itérations, $2n \times 500$ points, est également représenté. Comme la Multi `POLL` produit $(2n+1) \times 2n$ points à chaque itérations, les `POLL` en Oignon et Enrichie sont également exécutées pour ce nombre de points. Les `POLL` classique et Multi n'apparaissent qu'une fois car le nombre de points qu'elles génèrent à chaque itération est fixé par la dimension du problème. Les graphes de la figure 4.3 réunissent moyenne (cercle) et maximum

(triangle pointant vers le haut). La figure 4.3 ne tient compte que de la valeur finale, et ne donne aucune information sur la manière dont la valeur de la fonction objectif a diminué au cours de chacune des exécutions.

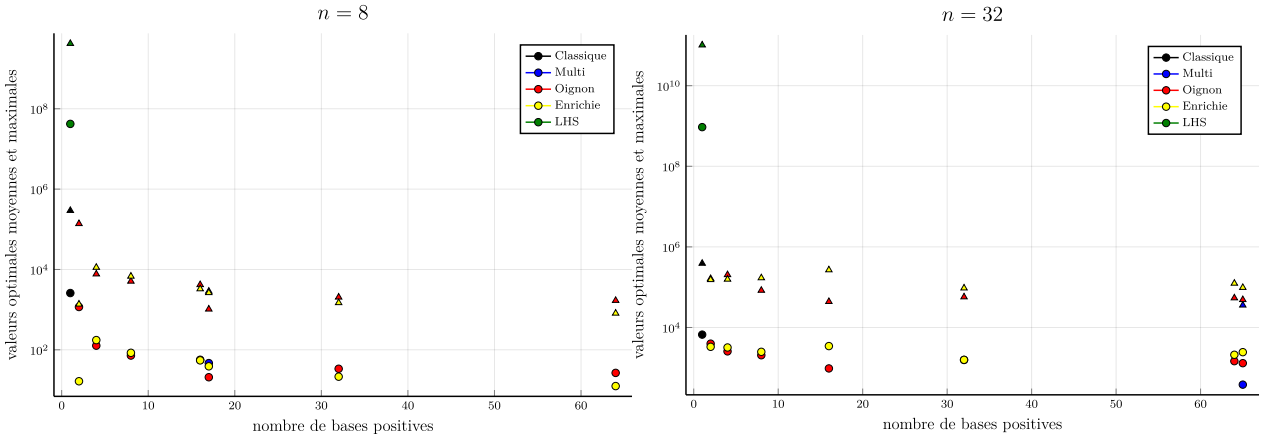

Figure 4.3 Influence du nombre de points sur la moyenne (cercle) et le maximum (triangle) des valeurs finales f^*. Stratégies statiques.

De la figure 4.3, il est possible de tirer deux conclusion :

1. La structure donnée aux points générés n'a pas n'impacte significatif sur l'amélioration de la fonction objectif.
2. Une meilleure valeur finale que dans le cas classique est obtenue avec une augmentation relativement «grande» du nombre de points.

Détaillons davantage le premier point. Pour cela, des profils de données pour les dimensions $2, 4, 8, 16, 32$ sont produit, pour comparer les stratégies classique, Multi, Oignon et Enrichie. Étant donné le caractère de la Multi-`POLL` à générer $(2n + 1) \times 2n$ points en dimension n, nous réglons le nombre de couches de la `POLL` en Oignon et le nombre de bases positives de la `POLL` Enrichie sur $2n + 1$: $c = 2n + 1$ et $q = 2n + 1$. La figure 4.4 confirme la conclusion donnée au point (1) ci-dessus.

La `POLL` classique génère $2n$ points là où les autres stratégies en génèrent $(2n + 1) \times 2n$, ce qui explique la distinction entre la `POLL` classique et les trois autres stratégies. L'augmentation de la précision a pour effet d'adoucir la transition entre les deux parties de chacune des courbes. Ici les graphes (a) et (b) de la figure 4.4 n'ont pas la même échelle en abscisse, et cette conclusion est plus difficile à voir. Il semble également pertinent de tracer des profils

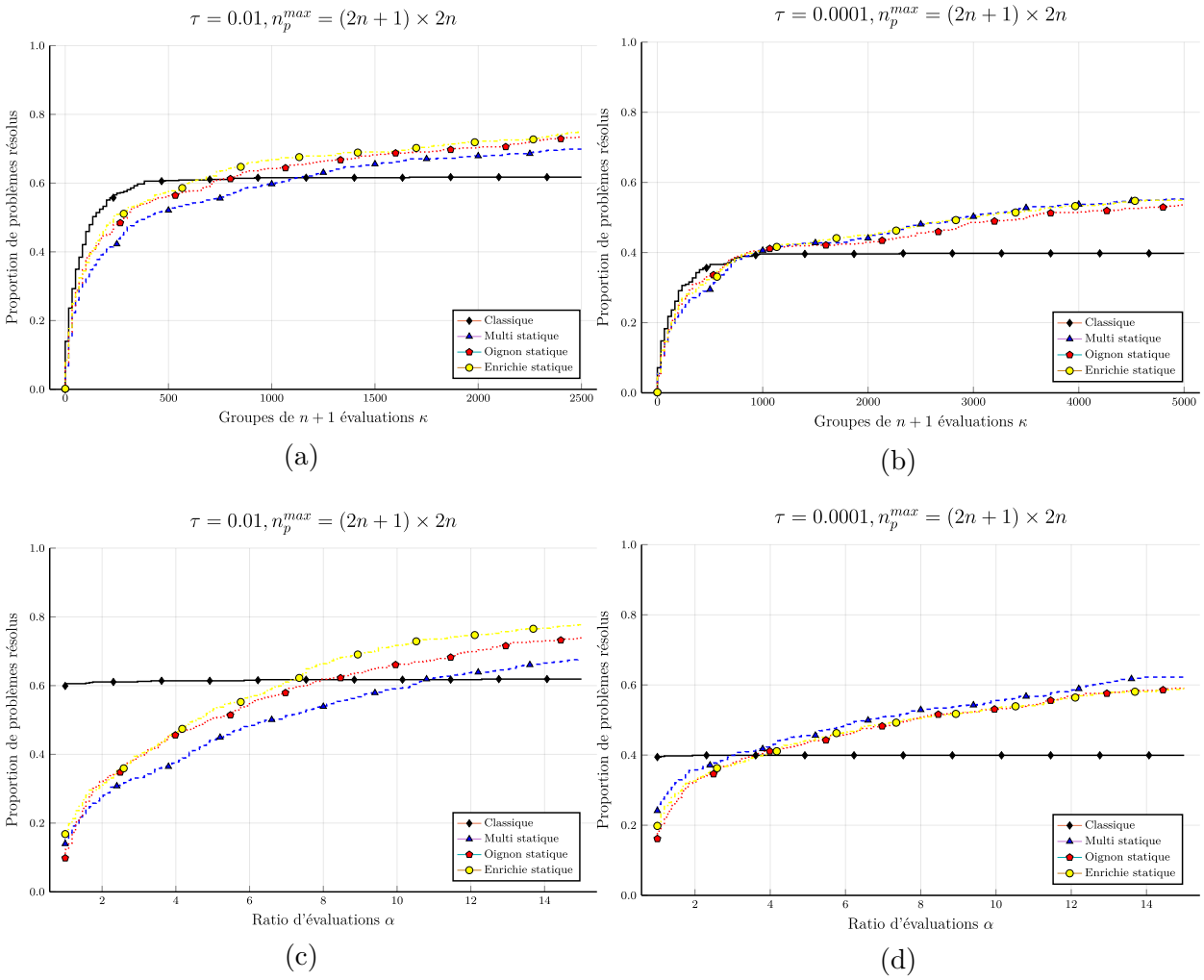

Figure 4.4 Profils de données (a,b) et profils de performances (c,d) en évaluation toutes dimensions confondues : Influence de la géométrie de génération de points.

de données en itérations (17) plutôt qu'en groupes d'évaluations. En effet, dans un environnement parallèle, si l'exécution est synchrone (ce qui est le cas ici), une itération constitue un cycle d'occupation de l'ensemble des processeurs. Cette occupation peut être complète ou non, selon le nombre de points généré à l'itération courante, n_p^k, et du nombre de processeurs ρ.

La figure 4.5 montre qu'à un nombre d'itérations fixé, les trois stratégies résolvent un plus grand nombre de problèmes que la `POLL` classique, mais comme précédemment, les règles utilisées pour générer les points ne semblent pas changer de manière significative le comportement des stratégies. La Multi `POLL` a des performances légèrement meilleures que la `POLL` en Oignon et la `POLL` Enrichie. Une explication possible à cette différence est que les points

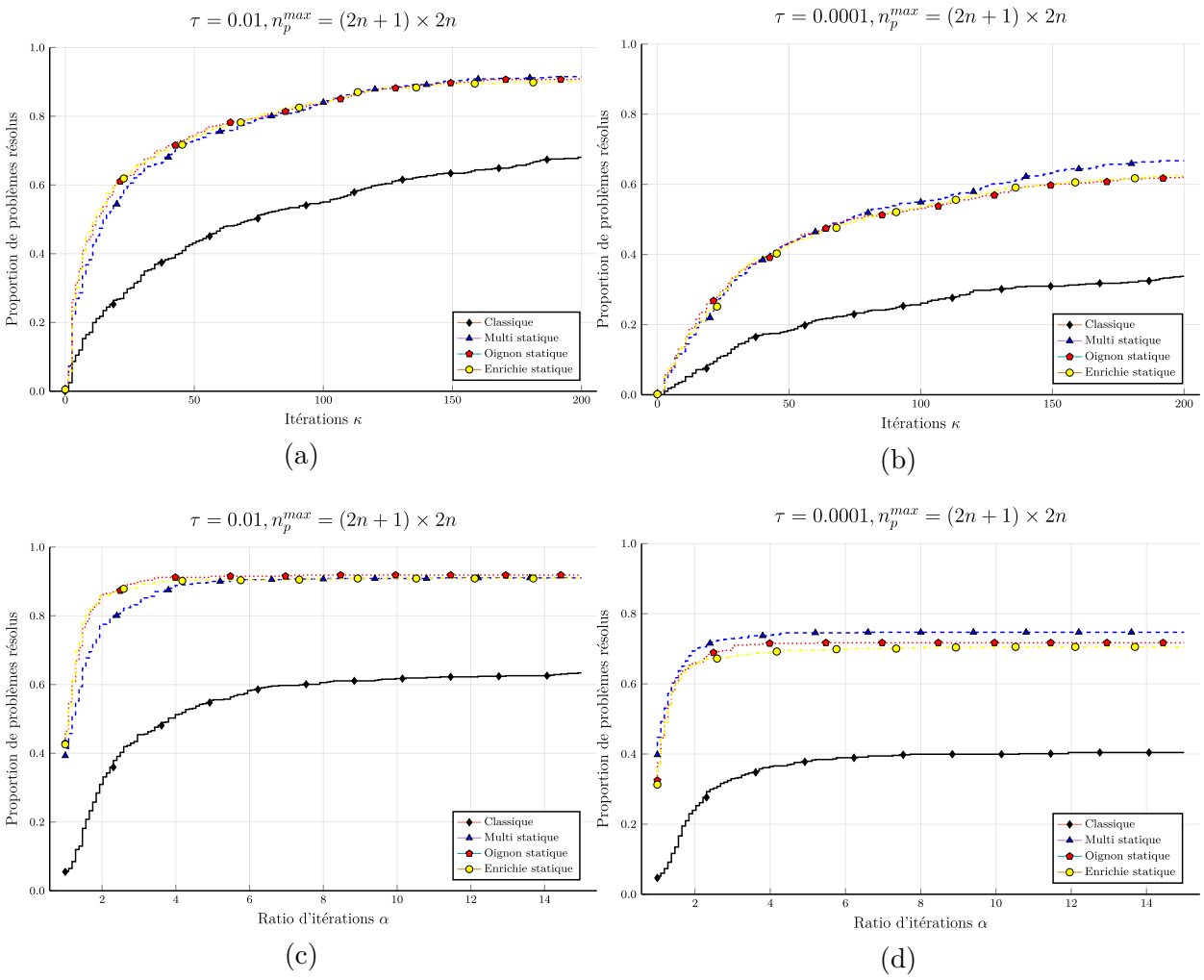

Figure 4.5 Profils de données (a,b) et profils de performance (c,d) en itérations toutes dimensions confondues : Influence de la géométrie de génération de points.

générés par la Multi `POLL` couvrent un rayon de $2\Delta^k$, alors que les points générés par la `POLL` en Oignon et la `POLL` Enrichie couvrent un rayon de Δ^k.

4.3.4 Influence de l'intensification

Le deuxième point énoncé plus haut et la figure 4.4 soulèvent une question : Est-il nécessaire de générer beaucoup de points de `POLL` à chaque itération ? Qu'en est-il si le nombre de direction de `POLL` est augmenté uniquement en cas d'échecs successifs ? Cela n'est possible que pour les stratégies Oignon et Enrichie. Le cadre algorithmique d'une augmentation du nombre de points quand cela est nécessaire a été donné en section 3.3. Il y a cinq contextes à comparer :

1. POLL statique,
2. POLL dynamique sans mémoire, à intensification linéaire :
   $n_p^k = \min\{n_p^{k-1}+2n, n_p^{max}\}$ si l'itération k n'améliore pas la valeur de l'objectif, $n_p^k = 2n$ sinon.
3. POLL dynamique sans mémoire, à intensification exponentielle :
   $n_p^k = \min\{2n_p^{k-1}, n_p^{max}\}$ si l'itération k n'améliore pas la valeur de l'objectif, $n_p^k = 2n$ sinon.
4. POLL dynamique avec mémoire, à intensification linéaire :
   $n_p^k = \min\{n_p^{k-1} + 2n, n_p^{max}\}$ si l'itération k n'améliore pas la valeur de l'objectif, $n_p^k = \max\{n_p^{k-1} - 2n, 2n\}$ sinon.
5. POLL dynamique avec mémoire, à intensification exponentielle :
   $n_p^k = \min\{2n_p^{k-1}, n_p^{max}\}$ si l'itération k n'améliore pas la valeur de l'objectif, $n_p^k = \max\{n_p^{k-1}/2, 2n\}$ sinon.

et ce, en terme d'évaluations et d'itérations. Les exécutions de la POLL classique sont également ajoutées à la comparaison, pour servir de référence. Les exécutions sont produites pour une dimension $n \in \{2, 4, 8, 16, 32\}$ et un nombre de points par étape de POLL maximum $n_p^{max} \in \{2 \times 2n, 4 \times 2n, 8 \times 2n, 16 \times 2n, 32 \times 2n, 64 \times 2n\}$, et ce, pour les quatre types d'intensification. Les axes de comparaison sont les suivants :

Différence entre les deux stratégies Oignon et Enrichie : Les profils de données sont semblables d'une stratégie à l'autre. Le mode statique résout la plus grande proportion de problèmes au prix d'un plus grand nombre d'évaluations effectuées. La figure 4.6 montre le comportement des deux stratégies, en évaluations et en itérations. La figure 4.6 montre également que l'ajout de la mémoire a plus d'influence que le type d'intensification linéaire ou exponentiel.

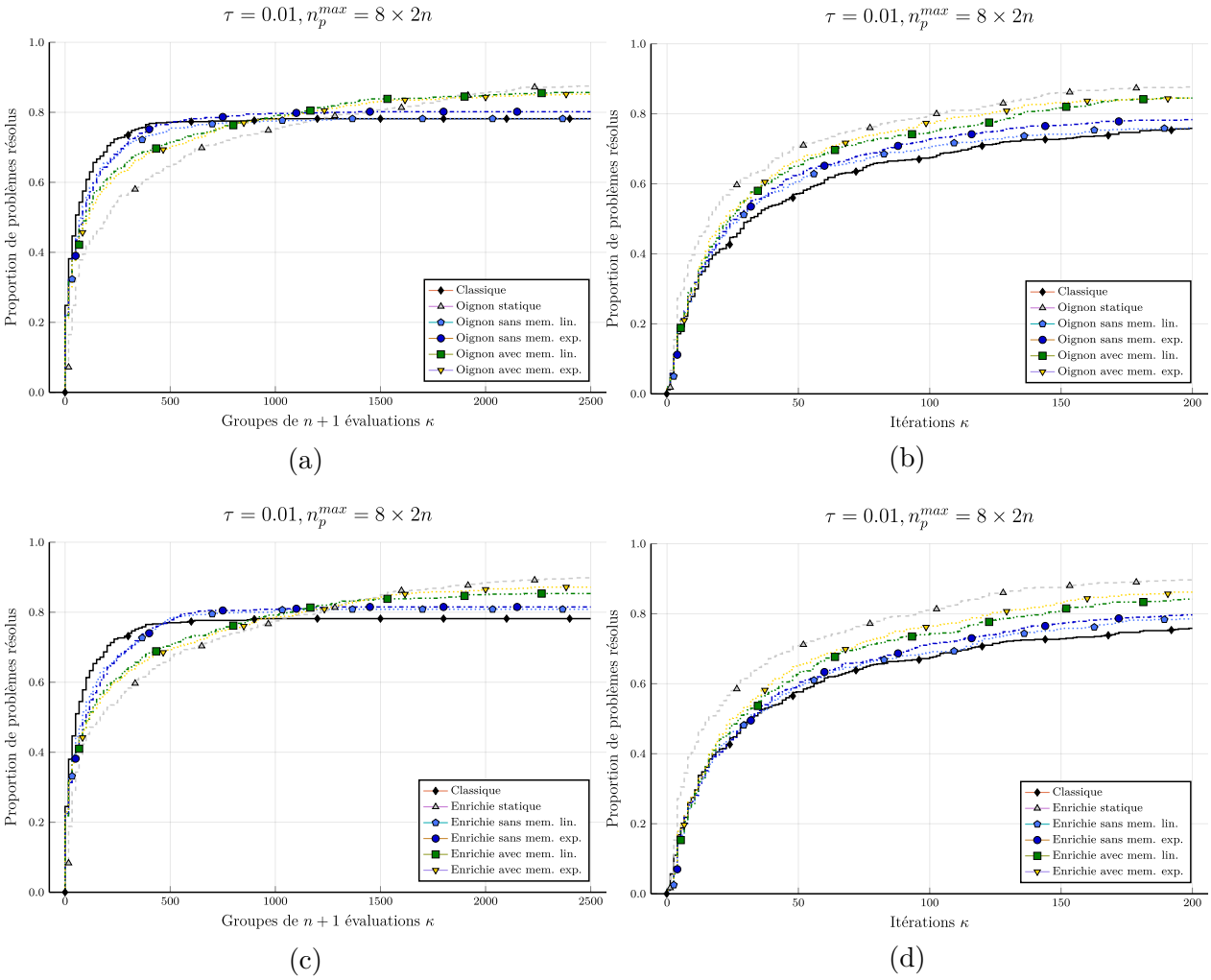

Figure 4.6 Profils de donnée en évaluation et en itération des stratégies Oignon (a, b) et Enrichie (c,d)

toutes dimensions confondues.

Différence entre deux dimensions données : La figure 4.7 montre que l'aspect général des courbes s'aplatit avec l'augmentation de la dimension : il est plus difficile de résoudre un problème de grande taille, cela nécessite plus d'évaluations. L'augmentation de la dimension semble favoriser l'intensification exponentielle avec mémoire, qui en dimension 16 et 32 est capable d'égaler le mode statique. L'intensification exponentielle avec mémoire permet aussi de résoudre un plus grand nombre de problèmes que le mode statique à l'aide d'un faible nombre d'évaluations lorsque la précision demandée est élevée. La figure 4.7 montre aussi que selon la dimension des problèmes, les types d'intensification sans mémoire peuvent aboutir à des résultats moins bons que la `POLL` classique. Il ne semble pas possible d'établir de règle de comportement plus précise des profils en fonction de la dimension du problème.

La figure 4.8 utilise les mêmes exécutions que la figure 4.7, mais représente des profils de données en itérations. Les conclusions données précédemment sont toujours vérifiées, à l'exception que cette fois ci, l'intensification statique domine toujours les autres stratégies : chaque itération du mode statique est plus performante car génère toujours le maximum de points.

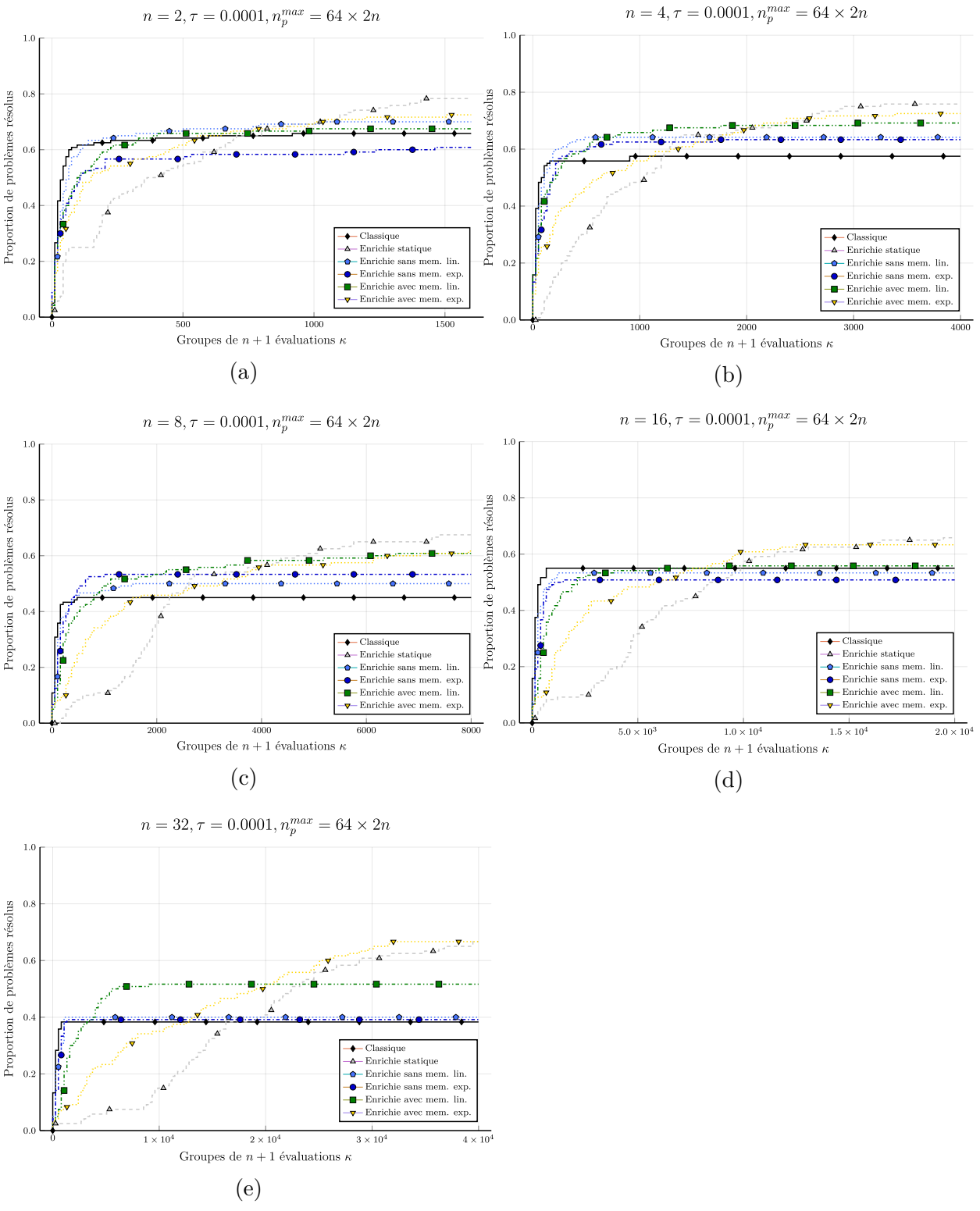

Figure 4.7 Profils de données en évaluations de la stratégie Enrichie pour différentes dimensions.

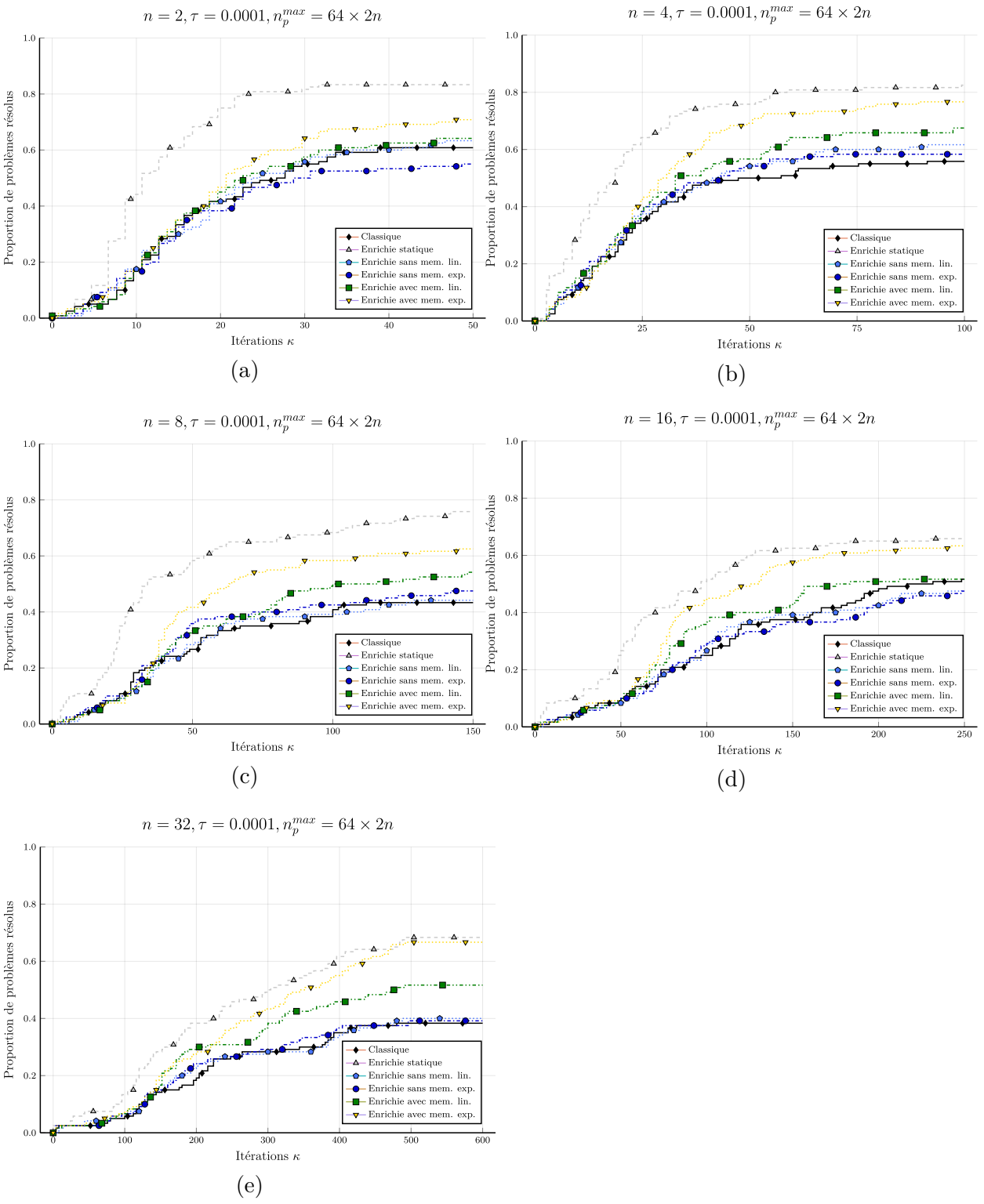

Figure 4.8 Profils de données en itérations de la stratégie Enrichie pour différentes dimensions.

Différence engendrée par l'augmentation de n_p^{max} : La figure 4.9 montre que l'augmentation de n_p^{max} influe le plus sur le mode statique, laissant quasiment inchangées les performances des autres stratégies à chaque itération.

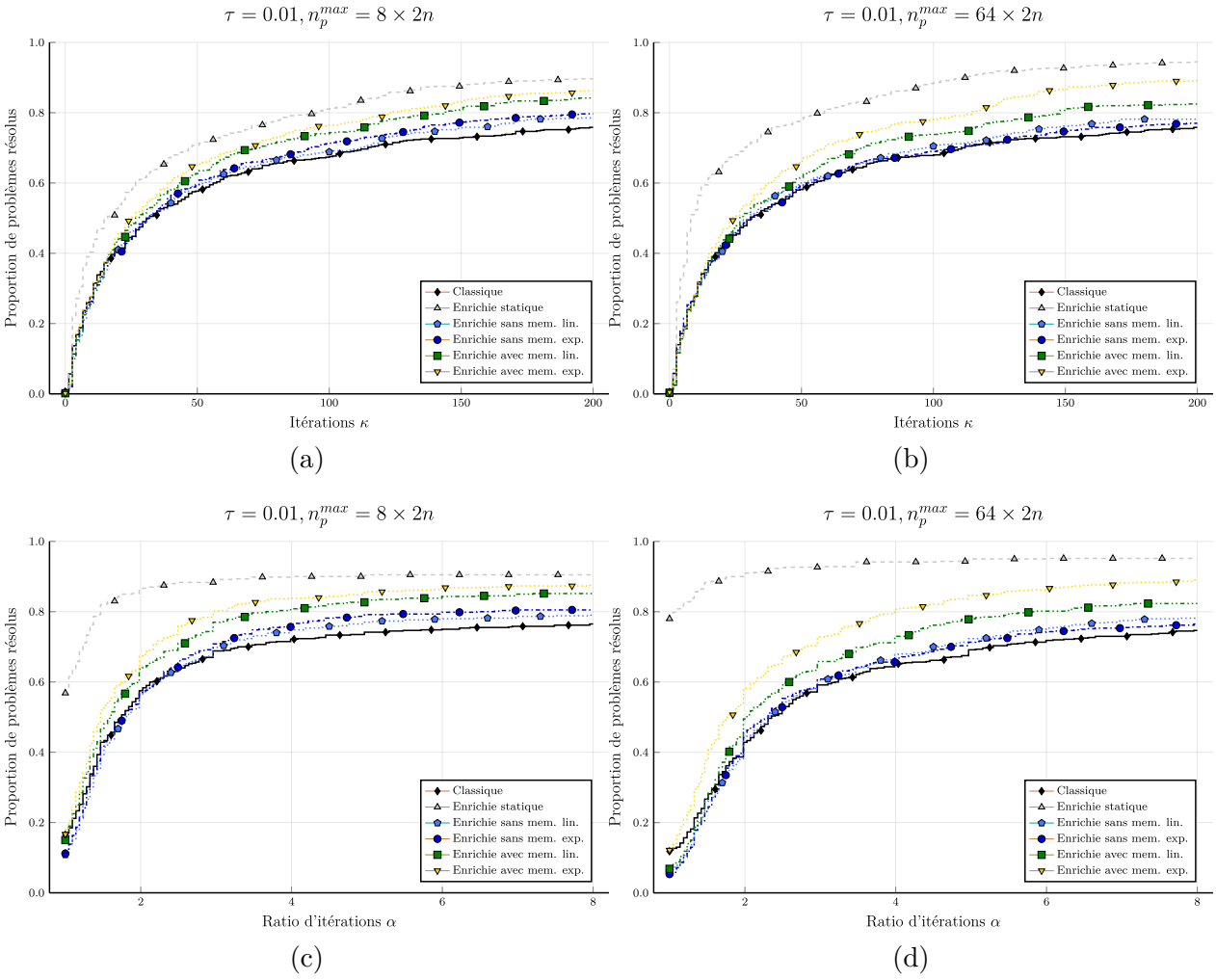

Figure 4.9 Profils de données (a,b) et de performance (c,d) en itérations de la `POLL` Enrichie avec $n_p^{max} \in \{8 \times 2n, 64 \times 2n\}$, et différents types d'intensification, toutes dimensions confondues.

4.4 Résultats numériques obtenus lors de la résolution d'un cas réaliste

Dans la section 4.3, NOMAD 4 est utilisé en désactivant les fonctionnalités de recherche globale, et les mécanismes additionnels. Dans cette section, l'étude tente de quantifier l'influence des stratégies de sonde décrites plus haut lorsqu'elles sont utilisées dans un contexte réaliste. Les exécutions sont réalisées dans deux contextes algorithmiques. Le premier contexte consiste à n'utiliser uniquement les stratégies de POLL sans qu'aucune autre option ne soit activée. Les résultats obtenus dans ce contexte sont présentés dans la section 4.4.2. Le second contexte consiste à activer les différentes options de NOMAD 4 telle que la SEARCH mais aussi l'opportunisme, le maillage anisotropique, etc. Les stratégies proposées en section 3.2 sont utilisées pour résoudre le problème STYRENE décrit en section 4.2. Les résultats obtenus dans ce contexte sont présentés à la section 4.4.3.

4.4.1 Détails des valeurs des paramètres algorithmiques et de l'environnement numérique de tests

Les résultats de cette section sont obtenus dans le contexte algorithmique suivant. Étant donné les résultats obtenus dans la section précédente, on s'intéresse uniquement aux stratégies Oignon et Enrichie, et plus particulièrement au type d'intensification utilisé. Pour les deux stratégies utilisées, $n_p^{max} = 64 \times 2n$. Dans le cas de l'intensification statique, n_p^{max} correspond au nombre de points générés dès que le rapport $\frac{\Delta^k}{\delta^k}$ le permet. Dans le cas de l'intensification dynamique, n_p^{max} correspond au maximum du nombre de points générés à chaque étape de POLL. Des exécutions sont réalisées de manière statique et de manière dynamique, comme présenté dans la section précédente. NOMAD 4 effectue 500 itérations au plus. Il y a également un critère d'arrêt sur la taille δ^k du maillage : $\delta^k \geq 10^{-13}$, où 10^{-13} représente la précision machine par défaut utilisée dans NOMAD 4(paramètre DEFAULT_EPSILON). Il n'y a pas de critère d'arrêt imposé sur le nombre d'évaluations.

L'environnement numérique pour la résolution de STYRENE est le suivant. Une évaluation de f et des c_j prend environ une seconde. NOMAD 4 est utilisé sur un ordinateur muni de 32 Go de mémoire vive et d'un CPU Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz. Pour chaque résolution, le nombre de fils d'exécution OpenMP alloués à NOMAD 4 est réglé à 1, pour des raisons semblables à celles exposées à la section 2.4 : ici, le but n'est pas de mesurer l'influence de la programmation parallèle sur le temps d'exécution, mais de mesurer l'influence des stratégies de POLL sur l'amélioration de la fonction objectif. De plus cela permet la reproductibilité des résultats.

4.4.2 Comportement des stratégies d'intensification seules

La figure 4.10 est obtenue en exécutant la résolution de `STYRENE` à partir du point $x^0 = (54, 66, 86, 08, 29, 51, 32, 15)$ initialement proposé dans la description du problème (4.2) comme point réalisable. La figure 4.10 représente l'évolution de la fonction objectif du problème `STYRENE` pour les deux stratégies Oignon et Enrichie exécutées avec différentes intensifications. L'évolution de f lors de la résolution à l'aide de la `POLL` classique est également représenté, à titre de comparaison.

Les quatre modes de `POLL` dynamique produisent exactement les mêmes résultats lors des premières évaluations, avant de se distinguer les uns des autres. Il est probable que cela soit dû aux types d'intensifications, qui conduisent à une amélioration de f différente, mais qui nécessitent d'avancer dans la résolution pour se distinguer les uns des autres.

Entre les stratégies Oignon et Enrichie, la différence se fait sur l'intensification linéaire et exponentielle sans mémoire. Ces deux types d'intensification conduisent à de moins bonnes valeurs de f dans le cas de la stratégie Enrichie.

Enfin, pour une exécution dans un environnement parallèle, il semble judicieux de choisir la stratégie Oignon, car à un nombre d'itérations fixé, la valeur de f obtenue avec cette stratégie est inférieure à celle obtenue avec la stratégie classique. On ne retrouve pas ce comportement à l'aide de la stratégie Enrichie.

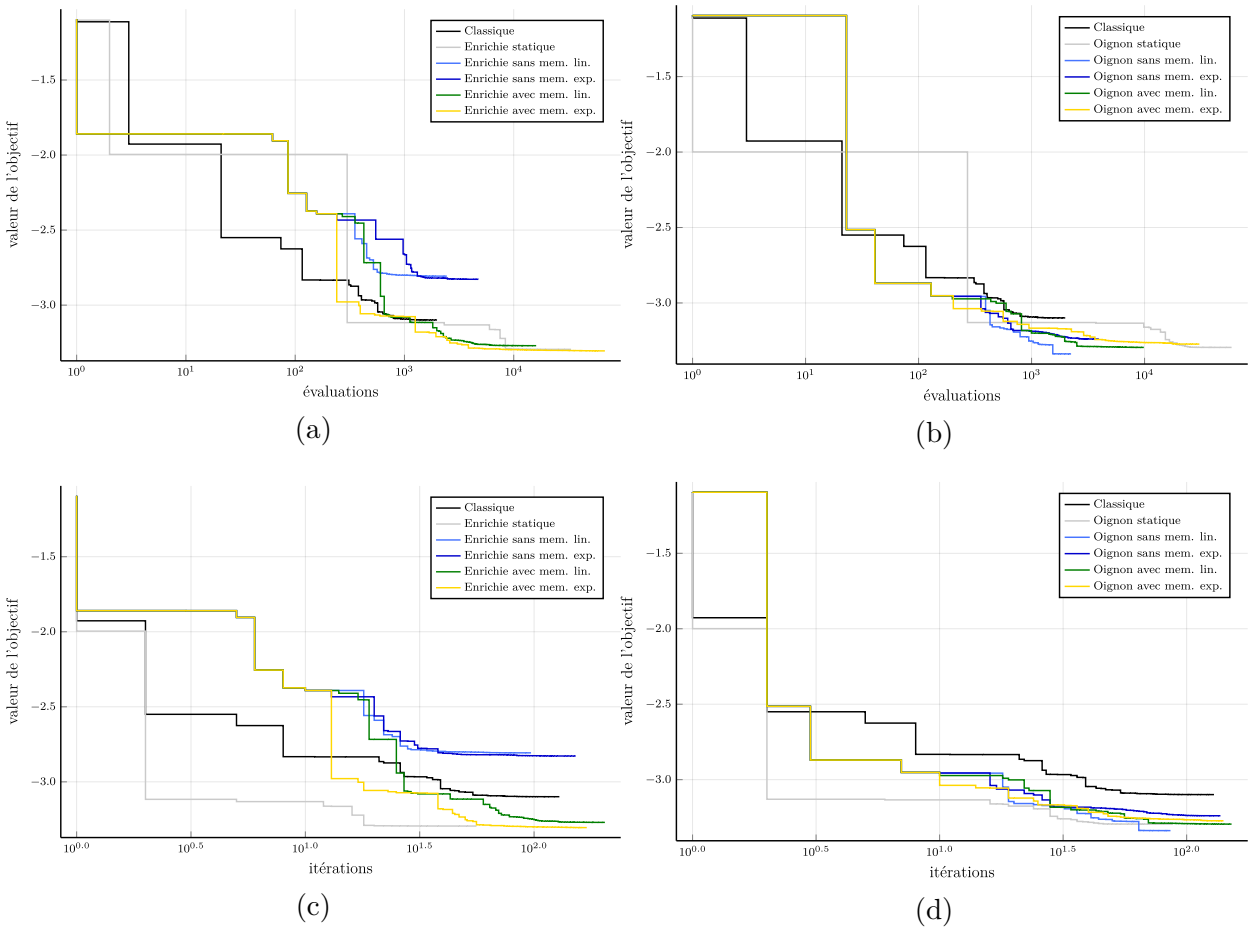

Figure 4.10 Graphe de convergence lors de la résolution de `STYRENE` pour les stratégies Oignon et Enrichie avec différents types d'intensification seules. Les graphes sont en évaluation (a,b), en itérations (c,d), $n = 8$, $n_p^{max} = 64 \times 2n$

4.4.3 Ajout de mécanismes d'aide à la résolution

Les mêmes exécutions que dans la section précédente sont menées, mais en activant la `SEARCH` par latin hypercube (`LH_SEARCH`) avec $n+1$ points échantillonnés sur $\mathcal{X}$ à chaque itération, et la recherche spéculative (`SPECULATIVE_SEARCH`), avec les paramètres par défaut. Le maillage anisotropique (`ANISOTROPIC_MESH`) et l'opportunisme (`OPPORTUNISM`) sont également autorisés, avec les paramètres par défaut. Le paramètre `FRAME_CENTER_USE_CACHE` est réglé à `true`. La figure 4.11 montre l'évolution de f au cours de la résolution selon la stratégie utilisée : Oignon ou Enrichie, ainsi que le type d'intensification. Dans chacun de ces cas, l'évolution de f est représentée en terme d'évaluations et d'itérations. L'exécution à l'aide de la `POLL` classique sans aucun des mécanismes cités ci dessus est également présentée pour permettre la comparaison avec les graphes de la figure 4.10. Les mécanismes supplémentaires appliquées

aux stratégies dynamiques dégradent les performances, au point de fournir de moins bons résultats que la POLL classique seule. Il n'y a pas de divergence de comportement entre les différents types d'intensification, contrairement à ce qui est observé sur la figure 4.10.

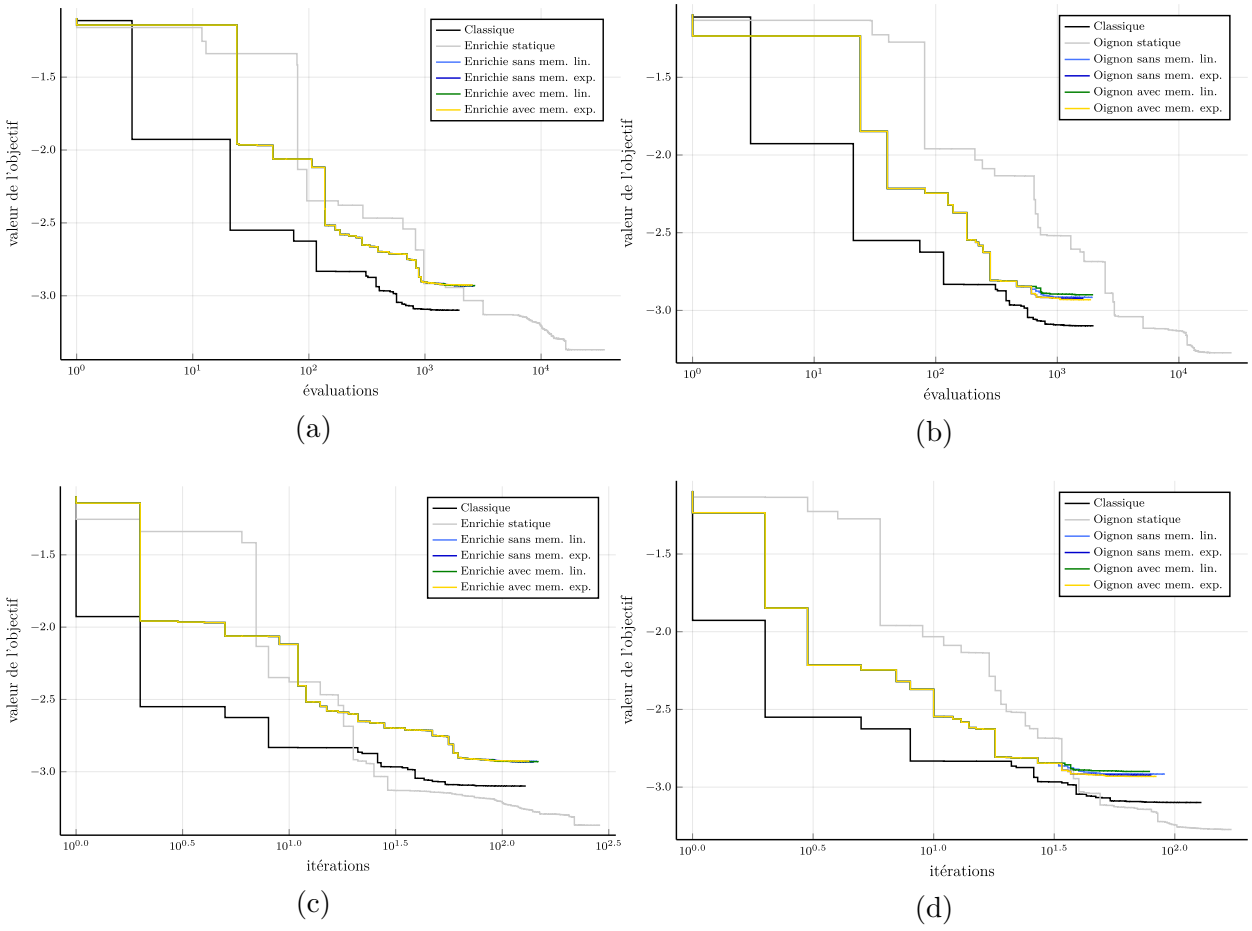

Figure 4.11 Graphe de convergence lors de la résolution de STYRENE pour les stratégies Oignon et Enrichie avec différents types d'intensification, munies des mécanismes d'aide à la résolution. Les graphes sont en évaluation (a,b), en itérations (c,d), $n = 8$, $n_p^{max} = 64 \times 2n$

La table 4.1 synthétise les résultats finaux obtenus lors de la résolution de STYRENE, avec et sans mécanismes supplémentaires supposés aider à la résolution du problème. La meilleure valeur de f^* est obtenue lorsque tous les mécanismes supplémentaires sont utilisés. 8 exécutions sur 10 faites sans mécanismes supplémentaires aboutissent à des valeurs de f^* inférieures à -3.20×10^7, tandis que seulement 2 exécutions sur 10 y parviennent lorsque les mécanismes supplémentaires sont activés.

intensification	stratégie	k	eval.	t (s)	$f^* \times 10^7$
POLL seule					
-	Classique	128	1987	743	-3.09896
statique	Oignon	87	58891	17222	-3.29458
	Enrichie	55	33447	9544	-3.2945
sans mem. lin.	Oignon	85	2237	1216	-3.33762
	Enrichie	96	2440	1074	-2.80671
sans mem. exp.	Oignon	136	3916	2071	-3.2391
	Enrichie	151	4744	1762	-2.82785
avec mem. lin.	Oignon	151	9847	3936	-3.2945
	Enrichie	203	16046	5673	-3.2701
avec mem. exp.	Oignon	140	30549	9919	-3.27251
	Enrichie	169	68274	29975	-3.30478
mécanismes supplémentaires					
statique	Oignon	171	27129	6900	-3.2727
	Enrichie	287	35439	9292	**-3.37052**
sans mem. lin.	Oignon	90	1956	523	-2.91599
	Enrichie	145	2589	686	-2.92779
sans mem. exp.	Oignon	79	1640	449	-2.92274
	Enrichie	140	2629	692	-2.93171
avec mem. lin.	Oignon	78	1971	545	-2.89931
	Enrichie	147	2701	707	-2.93027
avec mem. exp.	Oignon	83	1896	513	-2.93124
	Enrichie	133	2572	676	-2.92609

Tableau 4.1 Données finales obtenues pour les différentes stratégies utilisées sur STYRENE à partir de $x^0 = (54, 66, 86, 08, 29, 51, 32, 15)$.

4.4.4 Profils de données avec et sans mécanismes d'aide à l'optimisation

Pour approfondir les résultats obtenus dans les sections 4.4.2 et 4.4.3, des profils de données ont été tracés en amorçant la résolution de STYRENE en utilisant 10 points initiaux différents pour créer 10 instances du problème. La figure 4.12 montre que, à faible précision, l'ajout de tous les mécanismes d'aide à la résolution entraîne une dégradation des performances pour la plupart des stratégies. 4.13 montre que la perte de performance existe aussi en itérations.

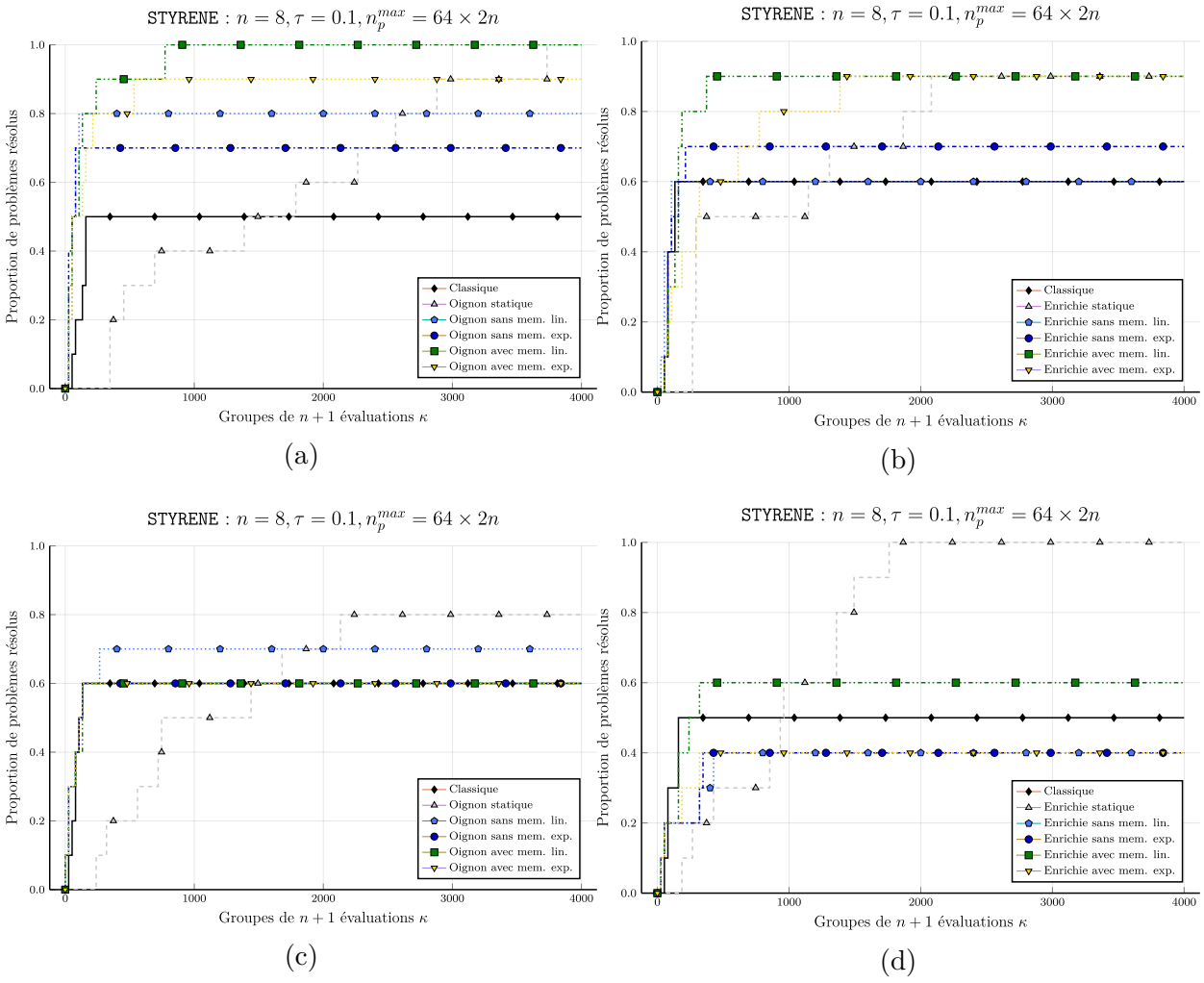

Figure 4.12 Profils de données en évaluation sur STYRENE pour les stratégies Oignon et Enrichie seules (a,b), et avec les mécanismes d'aide à la résolution supplémentaires (c,d).

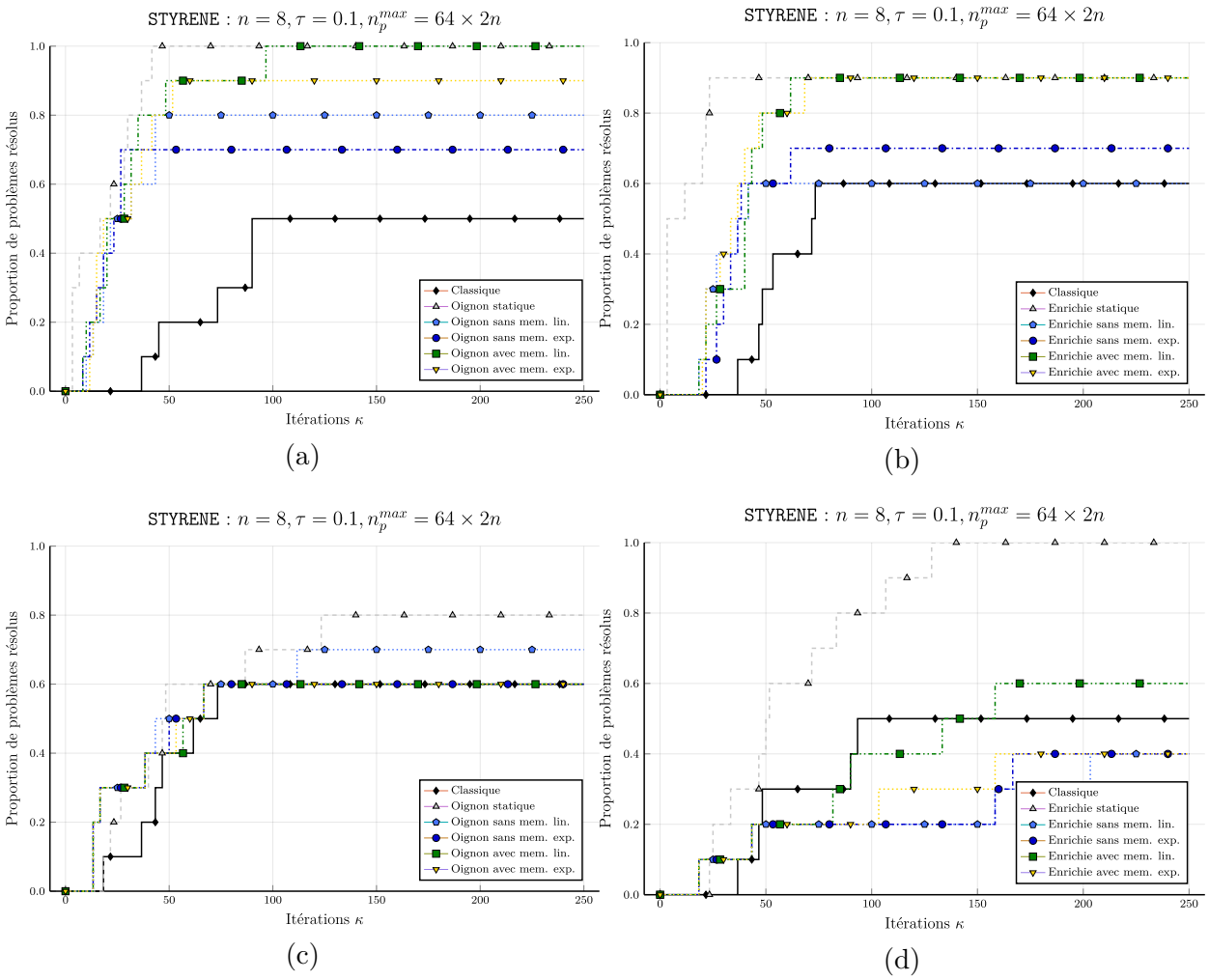

Figure 4.13 Profils de données en itérations sur STYRENE pour les stratégies Oignon et Enrichie seules (a,b), et avec les mécanismes d'aide à la résolution supplémentaires (c,d).

4.5 Discussion

L'ensemble de ces résultats numériques permettent de conclure que l'ajout de points de sonde est bénéfique dans la plupart des cas, mais la répartition de ces points n'influe que peu sur l'efficacité de la POLL. Le nombre de points de sonde n'a d'influence que lorsque sa valeur est drastiquement augmentée. Ceci pose le problème de l'évaluation de cette grande quantité de points. Les stratégies de POLL proposées, en particulier muni du mode d'intensification statique, ajoutent une quantité de travail non négligeable et peuvent s'avérer limitantes pour les boites noires coûteuses. Une économie sur le nombre d'évaluation peut être réalisé en utilisant les stratégies munies d'intensification dynamiques avec mémoire. Cette économie conduit à de moins bonnes performances que le mode statique dans la plupart des cas.

Les performances restent meilleures que celles obtenues à l'aide de la POLL classique. Les profils en itération permettent de rendre compte des performances dans un environnement parallèle idéal où toutes les évaluations d'une itération peuvent être effectuées simultanément. L'intensification dynamique permet d'économiser des évaluations mais les performances sont dégradées. Les résultats obtenus sur STYRENE montrent aussi que l'utilisation de tous les mécanismes d'aide à la résolution est contre productive.

REMARQUE : J'ai fais d'autres runs avec seulement opportunisme , LH search et speculative search, le résultat est le même : les performances sont moins bonnes qu'avec la POLL seule. j'ai alors relancé sans l'opportunisme, mais avec les deux search, j'attend les résutats, ça devrait être bon d'ici 1 ou 2 jours. à voir si je les mets dans le mémoire ou si je les présente que à la soutenance

CHAPITRE 5 Conclusion

5.1 Synthèse des travaux

Ce travail propose un cadre algorithmique pour la génération de points de sonde afin d'utiliser pleinement un grand ensemble de processeurs comparé à la dimension du problème lors de l'exécution de `MADS`. Trois stratégies sont proposées : la Multi `POLL`, la `POLL` en Oignon, et la `POLL` Enrichie. Ces stratégies diffèrent les unes des autres par la géométrie donnée à l'ensemble de points de sonde et sont implémentées dans `NOMAD` 4.

Le nombre de points générés à l'aide des stratégies Oignon et Enrichie peut être décidé à chaque itération. Un ensemble de règles de gestion du nombre de points de sonde générés à chaque itération est proposé pour les stratégies Oignon et Enrichie. Ces règles ont pour but de générer une grande quantité de points que lorsque cela s'avère nécessaire. L'idée sous-jacente est d'augmenter le nombre de points générés à l'itération courante lorsque les itérations précédentes sont des échecs, c'est à dire lorsqu'il devient difficile d'améliorer l'objectif. Quatre variantes de ces règles sont proposées : intensification avec ou sans mémoire et linéaire ou exponentielle.

Des essais numériques sont réalisés sur un ensemble de problèmes analytiques sans contraintes, ainsi que sur un cas réaliste : `STYRENE`. Les essais sur les problèmes analytiques montrent entre autre que la géométrie donnée aux points de sonde n'influe pas sur les performances de la résolution de $(\mathcal{P})$. Seul le nombre de points de sonde généré à chaque itération semble avoir un effet notable sur la résolution de $(\mathcal{P})$. Les essais sur `STYRENE` montrent que l'ajout de mécanismes d'aide à la résolution de $(\mathcal{P})$ est contre-productive.

5.2 Limites de la solution proposée

Le nombre de points à générer pour observer une amélioration des performances par rapport au cas classique est important. Cela pose problème dans le cas de boites noires coûteuses, car le nombre de points générés peut rapidement entraîner la saturation de l'infrastructure de calcul, même si celle ci possède un fort degré de parallélisme.

5.3 Améliorations et travaux futures

Des règles d'intensification analogue à celles données en section 3.3 pour la `POLL` peuvent également s'appliquer à la `SEARCH`, plus particulièrement à la stratégie de `SEARCH` exposée en

section 2.7. De ce fait, il est possible de concevoir un mode d'intensification complémentaire entre `SEARCH` et `POLL` : si à une itération donnée, la `SEARCH` conduit à un succès, alors le nombre de points générés lors de la `SEARCH` de l'itération suivante est augmenté jusqu'à une certaine limite fixée par l'utilisateur, et l'étape de `POLL` n'est pas réalisée. Si la `SEARCH` ne parvient pas à améliorer f, alors le nombre de points générés par la `SEARCH` est diminué, et l'étape de `POLL` est réalisée suivant le schéma d'intensification décrit en section 3.3

Les évaluations générées par les stratégies de sonde intensives forment une charge de travail qu'il est possible d'ordonner lorsqu'il est possible d'évaluer facilement la charge de travail ou le temps que requiert chaque évaluation. Ce cadre de travail est détaillé dans la section 3.4 mais n'est pas appliqué dans ce mémoire. Les performances de ce cadre de travail sont mesurables à l'aide des métriques détaillées dans la section 2.4.

RÉFÉRENCES

[1] M.A. Abramson, C. Audet, G. Couture, J.E. Dennis, Jr., S. Le Digabel, and C. Tribes. The NOMAD project. Software available at `https://www.gerad.ca/nomad/`.

[2] M.A. Abramson, C. Audet, J.E. Dennis, Jr., and S. Le Digabel. OrthoMADS : A Deterministic MADS Instance with Orthogonal Directions. *SIAM Journal on Optimization*, 20(2) :948–966, 2009.

[3] M.A. Abramson, C. Audet, J.E. Dennis, Jr., and S. Le Digabel. Parallel Versions of the MADS Algorithm for Black-Box Optimization. In *Optimization days*, Montreal, May 2010. GERAD. Slides available at `https://www.gerad.ca/Sebastien.Le.Digabel/talks/2010_JOPT_25mins.pdf`.

[4] S. Alarie. Use of surrogate-based model search for parallel blackbox optimization. Technical report, IREQ, 2019.

[5] S. Alarie, N. Amaioua, C. Audet, S. Le Digabel, and L.-A. Leclaire. Selection of variables in parallel space decomposition for the mesh adaptive direct search algorithm. Technical Report G-2018-38, Les cahiers du GERAD, 2018.

[6] C. Audet. A short proof on the cardinality of maximal positive bases. *Optimization Letters*, 5(1) :191–194, 2011.

[7] C. Audet. A survey on direct search methods for blackbox optimization and their applications. In P.M. Pardalos and T.M. Rassias, editors, *Mathematics without boundaries : Surveys in interdisciplinary research*, chapter 2, pages 31–56. Springer, 2014.

[8] C. Audet, V. Béchard, and S. Le Digabel. Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search. *Journal of Global Optimization*, 41(2) :299–318, 2008.

[9] C. Audet and J.E. Dennis, Jr. Mesh Adaptive Direct Search Algorithms for Constrained Optimization. *SIAM Journal on Optimization*, 17(1) :188–217, 2006.

[10] C. Audet and J.E. Dennis, Jr. A Progressive Barrier for Derivative-Free Nonlinear Programming. *SIAM Journal on Optimization*, 20(1) :445–472, 2009.

[11] C. Audet and W. Hare. *Derivative-Free and Blackbox Optimization*. Springer Series in Operations Research and Financial Engineering. Springer International Publishing, Cham, Switzerland, 2017.

[12] C. Audet, M. Kokkolaras, S. Le Digabel, and B. Talgorn. Order-based error for managing ensembles of surrogates in mesh adaptive direct search. *Journal of Global Optimization*, 70(3) :645–675, 2018.

[13] C. Audet, S. Le Digabel, and C. Tribes. NOMAD user guide. Technical Report G-2009-37, Les cahiers du GERAD, 2009.

[14] C. Audet, S. Le Digabel, and C. Tribes. The Mesh Adaptive Direct Search Algorithm for Granular and Discrete Variables. *SIAM Journal on Optimization*, 29(2) :1164–1189, 2019.

[15] C. Audet and C. Tribes. Mesh-based Nelder-Mead algorithm for inequality constrained optimization. *Computational Optimization and Applications*, 71(2) :331–352, 2018.

[16] I. Boussaïd, J. Lepagnot, and P. Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237 :82 – 117, 2013. Prediction, Control and Diagnosis using Advanced Neural Computations.

[17] D. Chazan and W.L. Miranker. A nongradient and parallel algorithm for unconstrained minimization. *SIAM Journal on Control*, 8(2) :207–217, 1970.

[18] L. Chen, H. Qiu, C. Jiang, X. Cai, and L. Gao. Ensemble of surrogates with hybrid method using global and local measures for engineering design. *Structural and Multidisciplinary Optimization*, 57(4) :1711–1729, 2018.

[19] A.R. Conn, N.I.M. Gould, and Ph.L. Toint. *Trust region methods*. SIAM, 2000.

[20] A.R. Conn and S. Le Digabel. Use of quadratic models with mesh-adaptive direct search for constrained black box optimization. *Optimization Methods and Software*, 28(1) :139–158, 2013.

[21] J.E. Dennis, Jr. and V. Torczon. Direct search methods on parallel machines. *SIAM Journal on Optimization*, 1(4) :448–474, 1991.

[22] E. Fermi and N. Metropolis. Numerical solution of a minimum problem. Los Alamos Unclassified Report LA–1492, Los Alamos National Laboratory, Los Alamos, USA, 1952.

[23] S. Finck, N. Hansen, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2010 : Presentation of the noiseless functions. Technical report, INRIA, 2010.

[24] U.M. García-Palomares and J.F. Rodríguez. New sequential and parallel derivative-free algorithms for unconstrained optimization. *SIAM Journal on Optimization*, 13(1) :79–96, 2002.

[25] T. Goel, R.T. Haftka, W. Shyy, and N.V. Queipo. Ensemble of surrogates. *Structural and Multidisciplinary Optimization*, 33(3) :199–216, 2007.

[26] J.D. Griffin, T.G. Kolda, and R.M. Lewis. Asynchronous parallel generating set search for linearly-constrained optimization. *SIAM Journal on Scientific Computing*, 30(4) :1892–1924, 2008.

[27] John L. Gustafson. Reevaluating amdahl's law. *Communications of the ACM*, 31 :532–533, 1988.

[28] P.D. Hough, T.G. Kolda, and V. Torczon. Asynchronous parallel pattern search for nonlinear optimization. *SIAM Journal on Scientific Computing*, 23(1) :134–156, 2001.

[29] P.D. Hough and J.C. Meza. A class of trust-region methods for parallel optimization. *SIAM Journal on Optimization*, 13(1) :264–282, 2002.

[30] M. Kokkolaras and B. Talgorn. EngagePlus technical Report. Technical report, IREQ and McGill University, 2017.

[31] T.G. Kolda. Revisiting asynchronous parallel pattern search for nonlinear optimization. *SIAM Journal on Optimization*, 16(2) :563–586, 2005.

[32] T.G. Kolda and V. Torczon. Understanding asynchronous parallel pattern search. In G. DiPillo and A. Murli, editors, *High Performance Algorithms and Software for Nonlinear Optimization*, pages 316–335. Kluwer Academic Publishers B.V., 2003.

[33] T.G. Kolda and V. Torczon. On the convergence of asynchronous parallel pattern search. *SIAM Journal on Optimization*, 14(4) :939–964, 2004.

[34] S. Le Digabel and S.M. Wild. A Taxonomy of Constraints in Simulation-Based Optimization. Technical Report G-2015-57, Les cahiers du GERAD, 2015.

[35] Ken IM McKinnon. Convergence of the nelder–mead simplex method to a nonstationary point. *SIAM Journal on Optimization*, 9(1) :148–158, 1998.

[36] J.J. Moré and S.M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization*, 20(1) :172–191, 2009.

[37] J.A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4) :308–313, 1965.

[38] M.J.D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2) :155–162, 1964.

[39] R. G. Regis. Evolutionary programming for high-dimensional constrained expensive black-box optimization using radial basis functions. *IEEE Transactions on Evolutionary Computation*, 18(3) :326–347, June 2014.

[40] B. Talgorn. `sgtelib` : Surrogate model library for Derivative-Free Optimization. `https://github.com/bastientalgorn/sgtelib`, 2019.

[41] V. Torczon. *Multi-Directional Search : A Direct Search Algorithm for Parallel Machines*. PhD thesis, Department of Mathematical Sciences, Rice University, Houston, Texas, 1989 ; available as Tech. Rep. 90-07, Department of Computational and Applied Mathematics, Rice University, Houston, Texas 77005-1892.

[42] V. Torczon. Pds : Direct search methods for unconstrained optimization on either sequential or parallel machines. Technical Report 92–09, Rice University, Department of Computational and Applied Mathematics, Mail Stop 134, 6100 Main Street, Houston, Texas 77005-1892, 1992.

[43] V. Torczon. On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7(1) :1–25, 1997.

[44] G. Tremblay. Programmation parallèle haute performance, 2017.

[45] K. Vu, C. D'Ambrosio, Y. Hamadi, and L. Liberti. Surrogate-based methods for black-box optimization. *International Transactions in Operational Research*, 24, 04 2016.

[46] W.I. Zangwill. Minimizing a function without calculating derivatives. *The Computer Journal*, 10(3) :293–296, 1967.