POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

Département de Mathématiques Appliquées et de Génie Industriel

Utilisation efficace de ressources parallèles lors de l'exécution de l'algorithme MADS

Sondes locales intensives lors de l'exécution de l'algorithme MADS à l'aide de ressources parallèles

Optimisations de boite noire à l'aide de l'algorithme MADS et de stratégies de sonde locale intensives

Comportement de l'algorithme MADS modifié à l'aide de stratégies de sonde locale intensives

Stratégies de génération de points lors de l'étape de sonde locale de l'algorithme MADS exécuté dans un environnement parallèle

Guillaume Lameynardie • 1970436

Table des matières

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. Suivant ce que modélise la quantité d'intérêt, les paramètres et les contraintes, les propriétés de ces objets mathématiques donnent lieu à des regroupement en famille, ou plutôt branches de l'optimisation. Par exemple, la branche de l'optimisation qui étudie 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 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 : on 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 on ne connaît pas le fonctionnement, 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 [10]. Ce processus est paramétré par $n \in \mathbb{N}^*$ variables $x = (x_1, x_2, \dots, x_i, \dots, x_n)$ et retourne (après des minutes voir des heures de fonctionnement) 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.

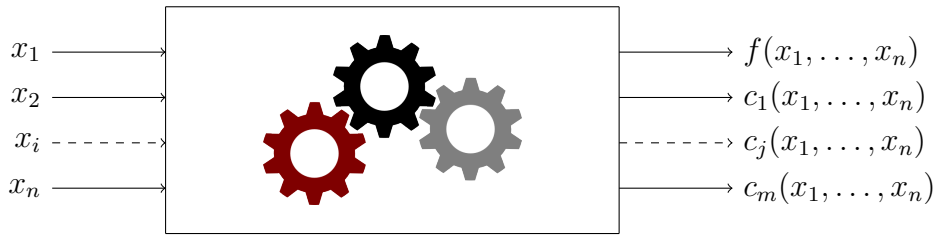


FIGURE 1 – représentation d'une boîte noire. Ce terme est utilisé pour signifier que nous ne connaissons pas de lien analytique entre les entrées et les sorties.

Dans la suite on se fixe comme convention que l'on veut minimiser la valeur d'intérêt, tout en ayant des valeurs de contraintes négatives ou nulles. Peu importe le contexte, on peut montrer que l'on peut toujours ramener un problème d'optimisation à la minimisation d'une certaine fonction, sous une ou plusieurs contraintes négatives ou nulles. Cela se formalise donc par :

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

Avec :

— $\mathcal{X}$ est l'ensemble des points réalisables pour f : On dit que $\mathcal{X}$ est formé par les contraintes de bornes. Par exemple, si x_1 représente l'épaisseur d'une aile d'avion, on a : $0 \leq x_1 \leq d$, où d est le diamètre de la cabine, si c'est notre 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 représente les contraintes, il se peut que cette liste ne soit pas exhaustive, et, que pour certains paramètres d'entrée, la valeur de l'objectif renvoyé par la boîte noire n'ait aucun sens. 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 intéressante.

— 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 l'estimation des dérivées de f et des c_j n'est pas considérée comme pertinente, en raison d'aucune garantie que les dérivées existent, et de contraintes cachées donnant lieu à des valeurs de sortie aberrantes.

— Une évaluation de f et des c_j est coûteuse en temps et en ressources informatiques, logistiques, etc.

— f et les c_j sont déterministes : fournir deux fois la même entrée fournira deux fois la même sortie. [1]

Nous pouvons également définir l'ensemble des points réalisables de la manière suivante :

$$\Omega = \{x \in \mathcal{X} : \forall j \in [\![1, m]\!], c_j(x) \leq 0\}. \tag{2}$$

Bien sûr, comme nous l'avons 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. 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 l'on utilise la méthode de Monte-Carlo pour obtenir une série de données sur laquelle on effectue ensuite des calculs.

1.2 Motivations et organisation

Jusqu'à maintenant, nous n'avons fait aucune hypothèse 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 évaluations de la boîte noire en parallèle, par exemple lorsque la boîte noire est une simulation numérique, que nous disposons des ressources informatiques suffisantes, ou lorsque la boîte noire est une expérience de laboratoire qui nécessite des ressources dont on dispose en plusieurs fois.

Ainsi nous pouvons 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.

Dans ce mémoire, nous nous intéressons au cas où le nombre de variables du problème est petit devant le nombre de processeurs. En effet, de nos jours il existe des infrastructures de calcul possédant des centaines, voir des milliers d'unités de calcul. Par exemple, l'Institut de recherche d'Hydro-Québec (IREQ) possède une grappe de calcul (CASIR) de 152 noeuds, chacun muni de 36 unités centrales de traitement. 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) [8] 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. En bref, on espère qu'avoir plus de processeurs à disposition va nous permettre de résoudre $\mathcal{P}$ plus rapidement ou d'obtenir une meilleure solution. On va donc chercher à fournir du travail judicieusement choisi à ces processeurs.

Par la suite, on considère que l'on 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, on expose une revue de littérature sur les méthodes d'optimisation de boite noire, avec un développent plus poussé pour les algorithmes de recherche directe. On présente le fonctionnement détaillé de `MADS`[8]. Puis nous nous intéressons aux travaux concernant l'utilisation de ressources parallèles dans les différentes méthodes d'optimisation de boite noire, en restant centré sur les méthodes de recherches directes. Enfin, on détaille les différents travaux qui allient `MADS` et environnement parallèle. Pour finir,on présente un bref aperçu des fonctionnalités de gestion de ressources parallèles, offertes par le logiciel `NOMAD` 4 [1, 12] sur lequel nous allons réaliser nos implémentations et nos tests numériques.

Comme expliqué dans la section 2, nous verrons qu'une itération de l'algorithme `MADS` se décompose en deux étapes : la recherche globale et la sonde locale. Aux vues de l'état de l'art présenté en section 2, nous discuterons donc dans la section 3 des différentes stratégies qui s'offre

à nous pour sélectionner des points candidats à l'évaluation lors de l'étape de sonde locale, dans le but d'utiliser un maximum de processeurs lors de cette étape.

La section 4 présente différentes métriques de performances de `MADS` muni des stratégies décrites en section 3.2, ainsi que des résultats numériques obtenu à l'aide de ces métriques. Le but de cette section est de montrer si oui ou non, il est pertinent de sonder plus intensément l'espace à chacune des étapes de sonde locale, et si oui, quel peut être le gain espéré. Le sondage plus intensif de l'espace s'accompagne d'une charge de travail plus grande à chaque étape de sonde locale. Cette charge de travail est effectuée en parallèle dans la mesure du possible, et occupe donc un maximum de processeurs.

2 Revue de littérature

Nous exposons ici l'état de l'art en optimisation de boite noire dans un environnement parallèle. 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, les infrastructures de calcul à l'aide de ressources parallèles se sont largement répandu, ouvrant la voie vers une nouvelle utilisation des algorithmes d'optimisation. Il est alors envisageable de résoudre des problèmes de plus grande dimension, mais les problèmes de petite dimension produisent des itérations qui sous occupent les ressources allouées.

2.1 Différentes méthodes d'optimisation de boite noire

Comme dans les autres branches de l'optimisation, des méthodes spécifiques aux caractéristiques du problème $\mathcal{P}$ ont été développées depuis les années 1950. La figure 2 donne une vue d'ensemble des types de méthodes existantes pour optimiser des problèmes de type boite noire.

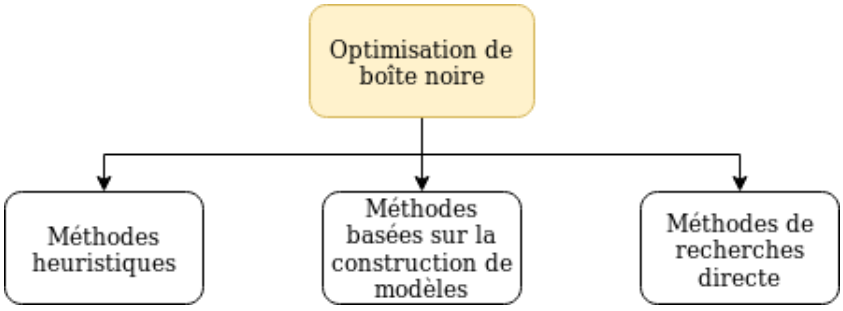


FIGURE 2 – Classification des méthodes existantes en optimisation de boîte noire

Détaillons un peu ces classes de méthodes :

Les méthodes heuristiques. Ce sont pour la plupart des algorithmes génétiques [15], 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 du vivant dans un environnement incertain. Cependant, pour être efficace, elles nécessitent un grand nombre d'évaluations de la boite noire, ce qui devient très vite limitant lorsqu'une évaluation est coûteuse. D'autres méthodes, telle que la méthode de Nelder et Mead (`NM`) [34] possèdent des propriétés de convergentes plus fortes et 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 : [32] 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. Ce modèle peut avoir des propriété de régularité exploitable telles que la différentiabilité, ou être beaucoup moins coûteux à évaluer que la boite noire. Ces classes de méthodes se montrent intéressantes lorsque le problème $\mathcal{P}$ est

coûteux à évaluer. Deux types de modèles se distinguent. On trouve les modèles dynamiques, construits à l'aide des évaluations déjà terminées de $\mathcal{P}$. Différentes classes de modèles existent [40]. On trouve aussi les modèles statiques, qui sont des simplifications du processus que modélise $\mathcal{P}$. Par exemple, si le calcul de f et des c_j fait appel à un processus itératif, un modèle $\mathcal{S}$ de $\mathcal{P}$ peut être le même problème avec moins d'itérations effectuées dans le fonctionnement interne. Ce type de modèle est généralement fourni par l'opérateur qui a construit $\mathcal{P}$. Toujours d'après [40], les manières d'utiliser un modèle pour construire une méthode d'optimisation sont très diverses. 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, on retrouve, entre autre les méthodes dite de région de confiance (`TR`) [18]. [36] expose une librairie de construction de modèles dynamiques de nature diverses. Cette librairie est utilisée dans `NOMAD` pour guider l'optimisation.

Les méthodes de recherche directe. Ce sont des méthodes itératives qui évaluent la boite noire sur une partie des noeuds d'un maillage de l'espace dans différentes directions autour de la meilleure solution connue. Suivant les résultats obtenus, on réduit ou on élargit le pas de ce maillage pour y effectuer de nouvelles évaluations. Une revue détaillée de ces méthodes est donnée dans [7]. Trois principales générations de méthodes de recherches directes ont vu le jour. La première d'entre elle, la recherche par coordonées (CS)[21] 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 on effectue à nouveau cette étape de sonde. En cas de succès on se déplace 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) [39] introduit trois améliorations d'après [7]. (1) L'ensemble des directions de sonde à chaque itération n'est plus restreint aux directions de coordonées, 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 coordonées. (2) Le pas de sonde peut être augmenté en cas de succès pour accélérer l'exploration de l'espace. (3) Enfin, 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 par treillis adaptatif (`MADS` [8]) 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 sous section 2.2.

Bien sûr, ces classes de méthodes ne sont pas cloisonnées, et on retrouve certains algorithmes qui empruntent des idées à plusieurs classes de méthodes, comme par exemple dans [19] qui expose 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, ou dans [35], qui 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. [11] propose une utilisation des modèles pour guider une optimisation à l'aide de `MADS`, et [14] qui expose une version de `NM` combiné à la mécanique de maillage utilisée dans `MADS`.

2.2 L'algorithme MADS

L'algorithme MADS [8] est un algorithme itératif de recherche directe qui comporte deux étapes : la recherche globale et la sonde locale. 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 3. Les étapes de recherche globale et de sonde locale font appel aux notions de maillage, de cadre de sonde locale 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 :

$$\forall x \in \mathbb{R}^n, \exists \lambda^1, ..., \lambda^N \in \mathbb{R}_+, \ x = \sum_{i=1}^{N} \lambda^i d^i \tag{3}$$

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 [10])**.** 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 :

$$M^k \ := \ \{ \ x^k + \delta^k D y \ : \ y \in \mathbb{N}^p \ \} \subset \mathbb{R}^n \tag{4}$$

Définition 4 (Cadre et pas de sonde locale [10])**.** 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 sonde locale de taille Δ^k généré par D centré en $x^k \in R^n$ est défini par :

$$F^k \ := \ \{ \ x \in M^k \ : \ \|x - x^k\|_\infty \leq \Delta^k b \ \} \subset \mathbb{R}^n \tag{5}$$

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

Notons qu'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, on pose $D = [I_n, -I_n]$ et donc $b = 1$. Afin d'alléger les explications, on considère le cas sans contrainte obtenu à l'aide de la barrière extrême [10] :

$$f_\Omega(x) = \begin{cases} f(x) & \text{si } x \in \Omega \\ +\infty & \text{sinon} \end{cases} \tag{6}$$

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 sonde locale Δ^0 et une tolérance ϵ positive ou nulle. L'algorithme 1 détaille la mise à jour des différents paramètres lors de l'exécution.

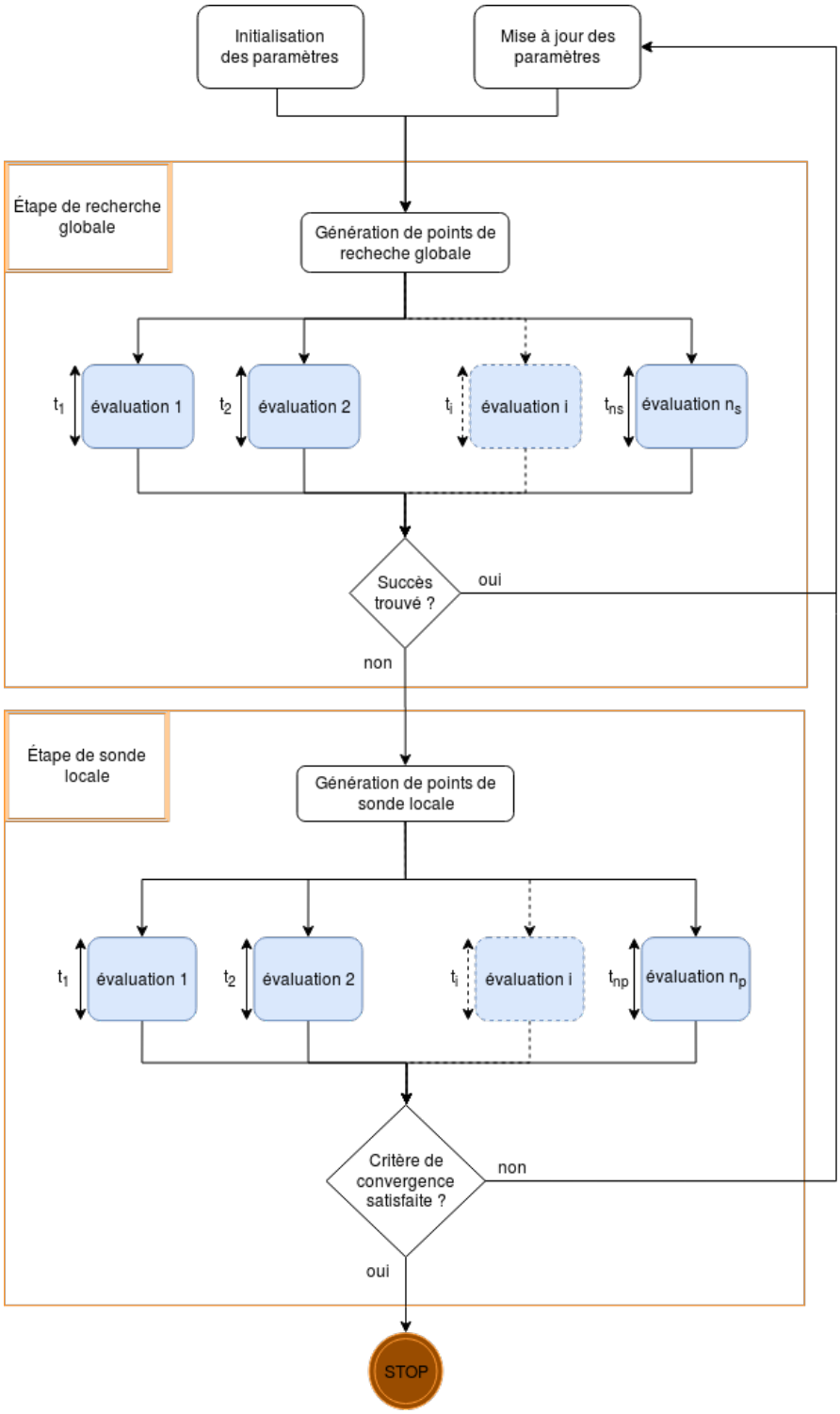


FIGURE 3 – fonctionnement de l'algorithme `MADS` en parallèle d'un point de vue logique.

L'étape de recherche globale 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 recherche globale 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 à l'itération k centré en x^k consiste en l'évaluation de f_Ω sur un ensemble de points $y^{k,1}, ..., y^{k,i}, ..., y^{k,n_p}$ appartenant au cadre de sonde F^k, de sorte que

$$\left\{ x^k - y^{k,1}, ..., x^k - y^{k,i}, ..., x^k - y^{k,n_p} \right\} \tag{7}$$

forme un ensemble générateur positif de $\mathbb{R}^n$. En pratique, on construit les points y_i en construisant un ensemble générateur positif de directions de sonde : $\mathbb{D}^k = \{d^{k,1}, ..., d^{k,n_p}\}$, et en additionnant chacune de ces directions au point x^k : $y^{k,i} = x^k + d^{k,i}$. La méthode utilisée [2] dans `NOMAD` 4 fait appel à l'opérateur de Householder pour générer une base orthogonale : pour $v^k \in \mathbb{R}^n$, $\|v^k\| = 1$, les colonnes de

$$H^k = I - 2(v^k)(v^k)^\top \tag{8}$$

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 $n_p = 2n$. On pose alors :

$$\mathbb{D}^k := \left\{ \pm \delta^k \text{round}\left(\frac{\Delta^k}{\delta^k} \frac{h_j^k}{\|h_j^k\|_\infty} \right) : j \in [\![1, n]\!] \right\} \tag{9}$$

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

La recherche globale nous autorise à évaluer des points en nombre fini n'importe où dans M_k (diversification), tandis que la sonde locale nous autorise uniquement à évaluer des points de F_k (intensification). La figure 4 montre un maillage sur lequel nous effectuons les étapes de recherche globale et de sonde locale. Le maillage est l'ensemble des points qui forment l'intersection des lignes fines. Notons qu'à chacune de ces deux étapes, il est possible d'obtenir des points candidats dans $\mathcal{X} \setminus \Omega$. C'est l'évaluation par f_Ω nous permet de déduire que ces points sont effectivement réalisable ou non.

Comme nous l'avons vu plus haut, les règles qui définissent les étapes de recherche globale et de sonde locale sont flexibles sur la manière de choisir des points candidats à l'évaluation. De plus, lorsque l'algorithme converge, à chaque étape de recherche globale et de sonde locale, de plus en plus de points sont candidats à être évalué.

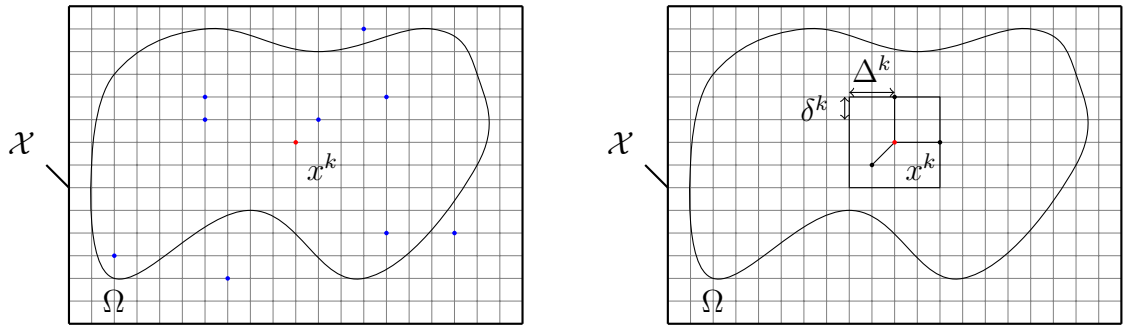


FIGURE 4 – Si l'étape de recherche globale (gauche) ne parvient pas à améliorer la valeur de la fonction objectif, on passe à l'étape de sonde locale (droite).

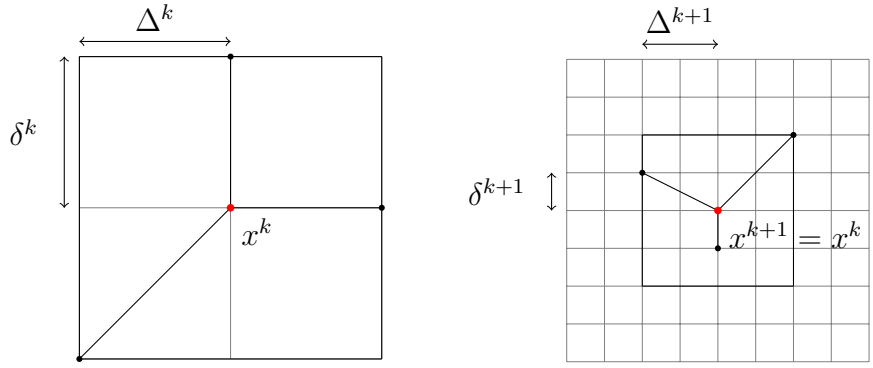


FIGURE 5 – Un échec de l'étape de sonde locale conduit à une réduction de δ^k et de Δ^k.

Afin d'éviter d'évaluer plusieurs fois f_Ω aux mêmes points lors de l'exécution de `MADS`, on maintient une cache des évaluations :

$$V := \left\{ \; (x^i, f_\Omega(x^i)) \;\; : \;\; i \in [\![1, q]\!] \; \right\} \tag{10}$$

où q est le nombre de points deux à deux distincts en lesquels on a évalué f_Ω. Ainsi, avant chaque évaluation de f_Ω on vérifie si le point candidat est présent en cache. 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 recherche globale à l'itération k
- $\mathbb{P}^k \leftarrow \varnothing$: Ensemble des points de sonde locale 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 $\mathbb{S}^k$.
Si $\exists t \in \mathbb{S}^k,\ f_\Omega(t) < f_\Omega(x^k)$:
  $(\Delta^{k+1}, \delta^{k+1}) \leftarrow \textsc{increase}(\Delta_k, \delta_k)$.
  $x^{k+1} \leftarrow t$.
  Aller à [3].
Sinon :
  Aller à [2].

[2]. Sonde locale :

Construire $\mathbb{P}^k$.
Si $\exists t \in \mathbb{P}^k,\ f_\Omega(t) < f_\Omega(x^k)$:
  $(\Delta^{k+1}, \delta^{k+1}) \leftarrow \textsc{increase}(\Delta_k, \delta_k)$.
  $x^{k+1} \leftarrow t$.
Sinon :
  $(\Delta^{k+1}, \delta^{k+1}) \leftarrow \textsc{decrease}(\Delta_k, \delta_k)$.
  $x^{k+1} \leftarrow x^k$.
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 $\textsc{increase}(\cdot,\cdot)$ et $\textsc{decrease}(\cdot,\cdot)$ peuvent être définies de plusieurs manières. Dans la version originale détaillée dans [8], on utilise la multiplication pour la diminution, ou

la division pour l'augmentation, par un rationnel [2] $\tau \in]0,1[$ (typiquement $\tau = \frac{1}{2}$) :

$$\text{DECREASE}(\Delta^k, \delta^k) = (\tau\Delta^k, \min\{\tau\Delta^k, (\tau\Delta^k)^2\}) \tag{11}$$

$$\text{INCREASE}(\Delta^k, \delta^k) = (\tau^{-1}\Delta^k, \min\{\tau^{-1}\Delta^k, (\tau^{-1}\Delta^k)^2\}) \tag{12}$$

tandis que dans [13], c'est un système utilisant une représentation compacte en base 10 qui sert à construire les opérateurs $\text{INCREASE}(\cdot,\cdot)$ et $\text{DECREASE}(\cdot,\cdot)$:

$$\text{INCREASE}((a \times (10)^b, (10)^{b-|b|}) = \begin{cases} 2 \times (10)^b, (10)^{b-|b|} & \text{si } a = 1 \\ 5 \times (10)^b, (10)^{b-|b|} & \text{si } a = 2 \\ 1 \times (10)^{b+1}, (10)^{(b+1)-|b+1|} & \text{si } a = 5 \end{cases} \tag{13}$$

$$\text{DECREASE}((a \times (10)^b, (10)^{b-|b|}) = \begin{cases} 5 \times (10)^{b-1}, (10)^{(b-1)-|b-1|} & \text{si } a = 1 \\ 1 \times (10)^b, (10)^{b-|b|} & \text{si } a = 2 \\ 2 \times (10)^b, (10)^{b-|b|} & \text{si } a = 5 \end{cases} \tag{14}$$

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, [13] montre qu'il est également possible de conserver les propriétes de comvergence 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.

2. Il est possible de construire des exemples avec τ irrationnel pour lesquels la preuve de convergence de MADS [8] n'est plus valide.

2.3 Parallélisme dans le logiciel NOMAD

L'algorithme MADS est implémenté dans le logiciel NOMAD [12] écrit en C++. La version 4, encore en développement, utilise OpenMP pour effectuer les évaluations des points des étapes de recherche globale et de sonde locale en parallèle. NOMAD 4 dispose d'un paramètre NB_THREAD_OPENMP permettant de spécifier le nombre de fils d'exécutions maximum que l'on autorise à être créés pour réaliser les évaluations parallèles. C'est ensuite le système d'exploitation qui planifie les charges de travail (attribution des fils d'exécution, changement de fil,...) à fournir aux processeurs. De ce fait, le parallélisme se restreint à une seule machine. La figure 6 montre les nuances entre les différents objets d'un environnement numérique parallèle. La diversité de forme des rectangles bleu ont pour but de faire comprendre que deux évaluations peuvent être très différentes tant sur la durée d'exécution, que sur la quantité de ressources nécessaire pour les compléter. En pratique, lorsqu'il est possible de paralléliser une charge de travail, deux grandeurs se distinguent : l'unité parallèle qui est une séquence d'instructions que l'on fixe comme insécable, dans notre cas, une évaluation ; et les fils d'exécution qui sont les «travailleurs» capable de compléter les séquences d'instructions demandées. Notons que le terme «processeur» définit en 1 correspond ici à un «travailleur».

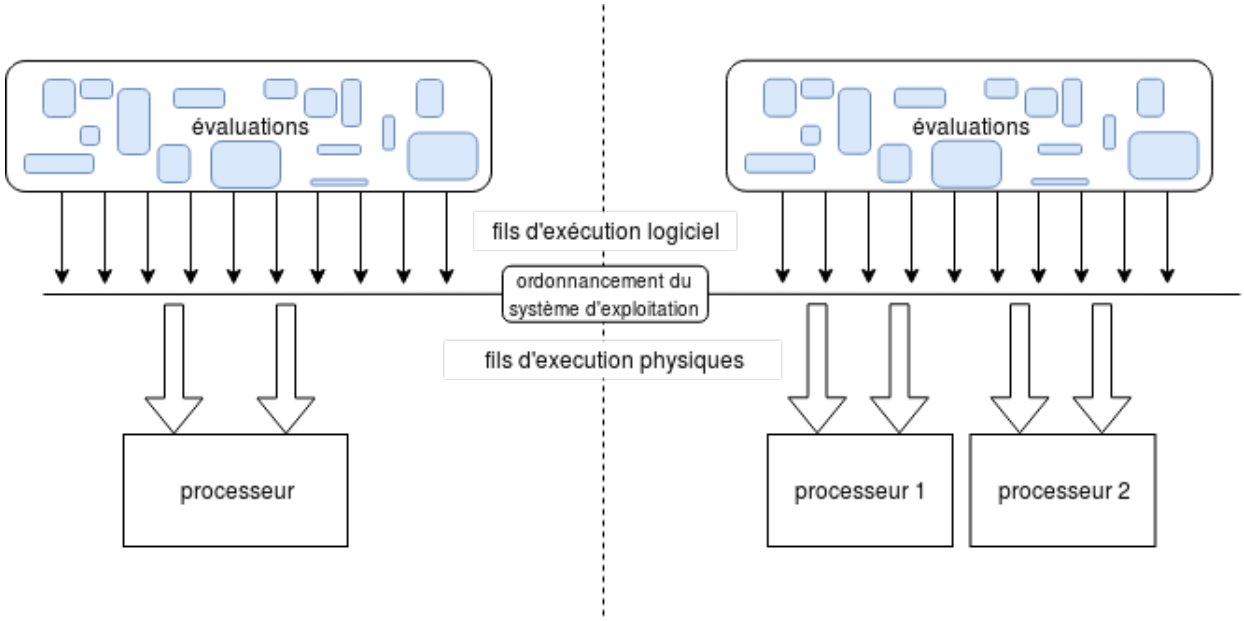


FIGURE 6 – Nuances entre fils d'exécution physiques, fils d'exécution logiciel et évaluations. Les fils d'exécution logiciel sont créés par les logiciels, tandis que le nombre de fils d'exécution physique dépend directement de l'architecture physique des processeurs. Ici, les processeurs sont à comprendre comme la puce que l'on trouve dans un ordinateur, et non pas comme au sens de la définition 1

Prenons l'exemple du calcul d'un produit scalaire entre deux vecteurs $u, v \in \mathbb{R}^n$ que l'on souhaite stocker dans la variable r : une unité parallèle peut être définie comme :

1. : choisir $i \in [\![1, n]\!]$ non déjà choisi et le marquer comme choisi.
2. : calculer le produit $u_i v_i$ et le stocker dans r_{temp}.

3. : calculer la somme $r + r_{\text{temp}}$ et placer le résultat dans r.

Les instructions 1 et 3 font intervenir des variables globales, ainsi, lorsqu'un fil d'exécution va exécuter cette séquence d'instruction, il devra mettre à jour les informations globales, et donc des communications seront nécessairement engendrées, pour verrouiller et déverrouiller les espaces mémoire associés aux variables en jeu. Ces opérations de verrouillage et déverrouillage des espaces mémoires sont relativement coûteuse comparé à une simple addition. Ainsi, on ne choisit pas le nombre de fils d'exécution comme égal au nombre d'unités parallèles présentes dans la tache que l'on souhaite réaliser, mais plutôt égal à deux fois le nombre de processeurs. Cela permet de limiter les communications et les changements de contextes associés à l'exécution d'un fil, et aussi de pouvoir continuer à travailler lorsqu'un des fils est en attente (ex : communication réseau) sans que le nombre de fils d'exécution en attente devienne trop grand. Donc, si l'on souhaite utiliser p processeurs lors de l'optimisation, fixer `NB_THREAD_OPENMP`$= 2p$ est un bon moyen empirique d'occuper au mieux les processeurs utilisés. Cependant, si les évaluations de la boite noire sont suffisamment peu coûteuse, comme nous allons le voir dans la section 4, il pleut être plus efficace d'utiliser un seul fil d'exécution, même si le nombre d'évaluations dépasse le nombre de processeurs physiques.

`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 donc le nombre maximum de points envoyés d'un seul tenant à la boite noire. Il est essentiel de comprendre que ces deux paramètres sont en fait équivalents. La différence est que, dans le cas de `NB_THREAD_OPENMP`, la création et la gestion des fils d'execution est effectuée par `NOMAD` tandis que dans le cas de `BB_MAX_BLOCK_SIZE`, c'est directement le programme qui implémente la boite noire qui gère la répartition parallèle du travail. Ainsi, dans le second cas, on 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 permet donc pas les communications entre les différents noeuds d'une infrastructure de calcul.

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

[20] décrit 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 : [37]). [38] décrit un ensemble de librairies en fortran qui implémentent, entre autres, l'algorithme décrit dans [20].

Le parallélisme synchrone et asynchrone de `GPS` : [26, 30, 31, 29] 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 entraine l'inactivité de certains processeurs à chacune des itérations, car comme expliqué dans [26], les temps nécessaire à deux évaluations distinctes sont, en terme général, distincts. L'asnchronisme retire cette barrière et offre à l'algorithme un fonctionnement opportuniste, dans le sens où des décisions sont prises alors que certains processeurs n'ont pas fini certaines évaluations de f. Une execution 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.

[27] décrit 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. [16] détaille une méthode de Powell 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. [25] présente 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.[23] propose une classe d'algorithmes de recherche directe pour l'optimisation sans contrainte, et expose 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.

Avant de continuer, il est essentiel de rappeler les travaux qui ont été effectués sur `MADS` portant sur l'utilisation de ressources parallèles. `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. `COOP-MADS` [3] exécute plusieurs instances de `MADS` en parallèle, chacune des instances étant pourvu d'une graine aléatoire différente. `PSD-MADS` [5] est applicable aux problèmes de grande dimension devant le nombre de processeurs. Cette version de `MADS` effectue une décomposition de l'espace et utilise plusieurs processeurs pour optimiser chacun des sous problèmes.

La figure 7 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 suffisament 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 7 est plus susceptible de survenir. Ainsi, dans NOMAD 4, plusieurs stratégies de recherche globale ont déjà été implémentées pour résoude ce problème de sous utilisation des ressources. Lors du fonctionnement de l'algorithme 1, il est pertinent d'utiliser des ressources parallèles uniquement pour déterminer si les assertions « $\exists t \in \mathbb{S}^k,\ f_\Omega(t) < f_\Omega(x^k)$» et «$\exists t \in \mathbb{P}^k,\ f_\Omega(t) < f_\Omega(x^k)$» sont vraies ou fausses. Ainsi, lorsqu'on est rendu à l'une de ces deux étapes, on effectue des évaluations comme représenté sur la figure 7. Dans un fonctionnement non opportuniste, le nombre d'évaluations à effectuer est alors $|\mathbb{S}^k|$ ou $|\mathbb{P}^k|$ suivant l'étape à laquelle on se trouve.

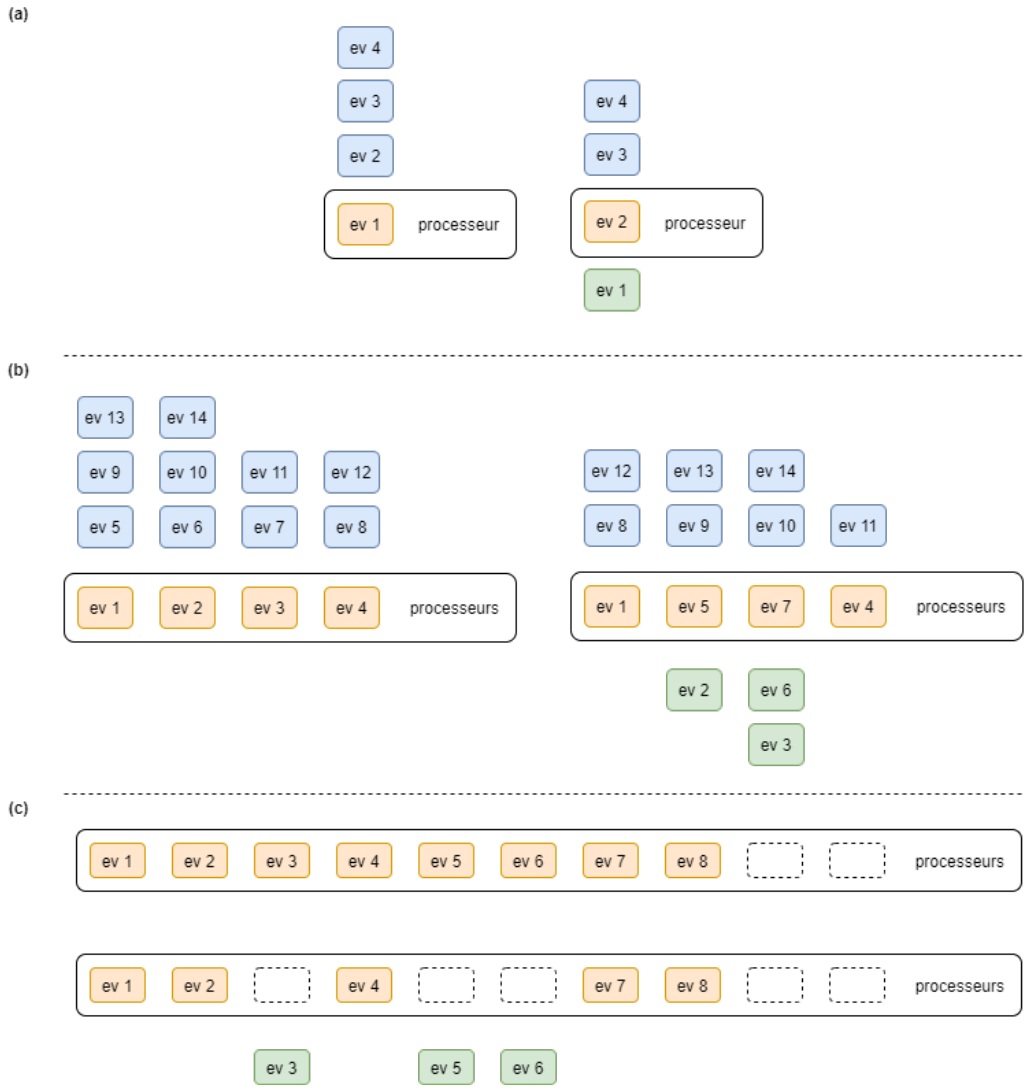


FIGURE 7 – De haut en bas : une suite d'évaluations en séquentielle, une suite d'évaluations parallèle avec moins de processeurs que d'évaluations à effectuer, une suite d'évaluations avec plus de processeurs que d'évaluation à effectuer.

2.5 Occupation d'un grand nombre de processeur lors de l'étape de recherche globale

Remarquons que les travaux exposés jusque là supposent implicitement que nous nous trouvons dans le cas (b) de la figure 7. Nous expliquons maintenant les travaux de B. Talgorn, M. Kokkolaras [28] et S. Alarie [4] concernant la génération de points lors de l'étape de recherche globale à l'aide d'une optimisation sur un modèle $\mathcal{S}$ (15) du problème $\mathcal{P}$. Ces travaux concernent le cas (c) de la figure 7 et vise à obtenir une configuration ressemblant plus au cas (b) de 7, donc à construire $\mathbb{S}^k$ de manière pertinente de sorte que $|\mathbb{S}^k| > p$. Pour simplifier les notations, on parlera d'une évaluation de $\mathcal{P}$ ou d'une évaluation de la boite noire, pour signifier que l'on évalue f et les c_j. Nous l'avons vu dans la section 2.1, il est possible de construire un modèle $\mathcal{S}$ du problème $\mathcal{P}$, bien moins coûteux à évaluer et qui partage néanmoins des similarités avec le problème initial :

$$\mathcal{S} : \begin{cases} \min\limits_{x \in \mathcal{X}} & \tilde{f}(x) \\ \text{s.c} & \tilde{c_j}(x) \leq 0, \quad j \in [\![1, m]\!]. \end{cases} \tag{15}$$

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 [24],[17],[36].

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

$$d(x, y) = \|x - y\|. \tag{16}$$

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

$$d(x, Y) = \min_{y \in Y} d(x, y) = \min_{y \in Y} \|x - y\|. \tag{17}$$

On définit également l'agrégation des contraintes [9] par :

$$\tilde{h}(x) = \sum_{j=1}^{m} \max(\tilde{c_j}(x), 0)^2. \tag{18}$$

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

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

$$x \prec y \iff \tilde{h}(x) < \tilde{h}(y) \text{ ou } (\tilde{h}(x) = \tilde{h}(y) \text{ et } \tilde{f}(x) < \tilde{f}(y)). \tag{19}$$

La diversification se fait à l'aide de l'optimisation d'un modèle $\mathcal{S}$ du problème $\mathcal{P}$. À chaque itération k de MADS, nous exécutons l'algorithme 2 lors de la recherche globale :

Algorithm 2 Sélection de points lors de la recherche globale

[0]. Initialisation :
- k : valeur de l'itération de MADS courante.
- $T \leftarrow \varnothing$.
- $i \leftarrow 1$: numéro de la sélection utilisée.
- Sélectionner $t_{max} \in \mathbb{N}^*$: nombre de points à sélectionner au maximum.
- Sélectionner X un ensemble de points déjà évalués sur $\mathcal{P}$.

[1]. Génération de S :
- Construire $\mathcal{S}$ à l'aide de X.
- Effectuer une optimisation de $\mathcal{S}$ par la méthode de notre choix pour construire S.
- Aller à 2.

[2]. Sélection de points dans S
- **Tant que** la sélection i retourne un point s **faire** :
  - **Si** $|T| < t_{max}$ **faire** :
    - $T \leftarrow T \cup \{s\}$.
  - **Sinon** :
    - aller à [3].
- **Sinon** :
  - $i \leftarrow i + 1$.
  - Aller à [2].

[3]. Evaluation sur $\mathcal{P}$
- **Pour** $s \in T$ **faire** :
  - Projeter s sur M^k pour obtenir $\hat{s}$.
  - Evaluer $f(\hat{s}),\ c_1(\hat{s}), ...,\ c_m(\hat{s})$.

L'étape 3 de l'algorithme 2 est effectuée en parallèle. Les six stratégies de sélection de points applicables lors de l'étape de recherche globale écrites par B. Talgorn et M. Kokkolaras dans [28] 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 $S \setminus X \cup T$ sont triés suivant la relation d'ordre $\prec$, et le meilleur point est séléctionné pour être ajouté à T.

Point le plus distant de X et T Itéré plusieurs fois, la stratégie précédente peut facilement retrourner des points dans la même région. Dans cette deuxième stratégie, on désire donc aller explorer des régions que l'on a pas encore échantilloné. C'est donc le point de S le plus loin de $X \cup T$ (au sens de d) qui est ajouté à T.

Meilleur point à distance minimale de X et T Un autre moyen d'appliquer la stratégie 2.5 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

On sélectionne donc le meilleur point de S (au sens de $\prec$) à une distance $d_{\min}$ (au sens de d) de $X \cup T$. Notons qu'avec cette stratégie, si $d_{\min}$ est trop grand, la stratégie peut ne rien renvoyer. En pratique, on commence avec $d_{\min} = 0$, puis $d_{\min} = \Delta_k$

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

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

Avec :

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

au début, comme première valeur, puis :

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

On force la sélection à renvoyer 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 2.5 qui consiste à chercher un point loin des régions que l'on a déjà échantillonné. Cette fois ci, on construit un indicateur d'isolement,d_{isolated}, que l'on peut appliquer à chacun de points. Cette mesure, appliqué à un point $s \in S$ 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 on cherche le plus "haut" sommet s^*(dans notre cas, on minimise, donc on devrait parler de plus profonde vallée). Ce plus haut sommet, $s*$, est caractérisé comme ayant le plus grand nombre de points dans un rayon de $d_{\text{isolated}}(s^*)$.

Pour $s \in S$:

1. $d_{\text{isolated}}(s)$ représente la distance (au sens de d) à l'ensemble des points $u \in S$ 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 :

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

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

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

Point dans la zone la plus peuplée Ici, nous cherchons le point s^* dans la zone la plus peuplée. Le but de cette stratégie est de sélectionner les points de S ayant beaucoup de voisins dans S, mais peu de voisins dans X. Ainsi, si l'on trouve un tel point, cela signifie qu'il est dans une zone que l'optimisation sur $\mathcal{S}$ a déterminée comme prometteuse mais qui n'a pas beaucoup été explorée jusqu'à maintenant lors de l'optimisation sur $\mathcal{P}$.

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

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

s^* doit alors vérifier :

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

3 Occupation d'un grand nombre de processeur lors de l'étape de sonde locale

Ces stratégies ont pour but d'établir des règles de sélection de des points candidats à être évalués lors de l'étape de sonde locale définie dans la section 2.2. En effet, comme la recherche globale, l'étape de sonde locale reste très générale concernant les conditions que doivent satisfaire les points que l'on évalue. Nous présentons donc trois stratégies de sélection de points. Les deux premières sont construite sur une idée semblable aux propos exposés dans [20] : La sonde locale enrichie est bâti en anticipant que l'on trouve des succès dans tout ou partie des voisins de sonde locale. Inversement, la sonde locale 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 donc de passer d'une configuration semblable à 7(c) vers une configuration de type 7(b) en augmentant le nombre de points candidats, et donc le nombre d'évaluations à effectuer. La sonde locale enrichie a pour but d'explorer également l'intérieur du cadre de sonde.

3.1 Cadre de travail pour l'étape de sonde locale

Comme nous l'avons vu dans la section 2.2 l'étape de sonde locale n'est pas beaucoup plus contraignante que l'étape de recherche globale pour la génération de points. Un choix qui a éte fait dans [2] est d'utiliser des directions orthogonales ajoutées au centre de sonde locale courant x^k et de projeter le résultat sur le maillage M^k. Ceci peut être résumé dans l'algorithme 3 :

Algorithm 3 sonde locale classique - Génération de points

[0]. Initialisation :

- δ^k : Pas de maillage.
- Δ^k : Pas de cadre.
- x^k : Point donnant lieu au succès courant.
- $\mathbb{D} \leftarrow \varnothing$: Ensemble de direction de sonde locale.
- $\mathbb{P} \leftarrow \varnothing$: Ensemble de points de sonde locale.

[1]. Génération de direction de sonde locale :

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]. Construction de points de sonde locale :

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

RETOURNER $\mathbb{P}$.

Où h_j est la j-ième colonne de H. $\mathbb{D}$ forme une base positive de l'espace et chacune des coordonnées de ses éléments sont un multiple du pas de maillage δ^k. De cette manière, on peut

considérer la génération de points de sonde locale classique comme une fonction :

$$\text{CLASSICALPOLL} : \mathbb{R}_+^* \times \mathbb{R}_+^* \times \mathbb{R}^n \to F^k \tag{20}$$

$$(\delta^k, \Delta^k, x^k) \mapsto \mathbb{P} \tag{21}$$

3.2 Sélection de points lors de l'étape de sonde locale

Ces stratégies permettent de générer des points de sonde locale en plus grande quantité que ce qui est proposé dans le fonctionnement de NOMAD . Avec plusieurs structures de construction, on espère obtenir des différences de performances dans la résolution de $\mathcal{P}$ suivant la stratégie que l'on utilise.

3.2.1 Multi-sonde locale

Ici, nous voulons fournir de la charge de travail aux processeurs disponibles en générant les voisins des voisins de sonde locale. Sur le centre de sonde locale principal, x_k, nous générons q directions de sonde locale primaires $(d^1, .., d^i, .., d^q)$ qui donnent lieu à q centres de sonde locale secondaires $(y^1, .., y^i, .., y^q)$. Pour chaque centre de sonde locale secondaire $y^i = x^k + d^i$, nous générons à nouveau q_i directions de sonde locale secondaires, dans un cadre de taille $\Delta^{k,i}$, centré en y^i, qui permettent de construire les points $y^{i,j},\ j \in [\![1, q_i]\!]$. À une étape de sonde locale, on produit alors $q + \sum_{i=1}^{q} q^i$ candidats à évaluer.

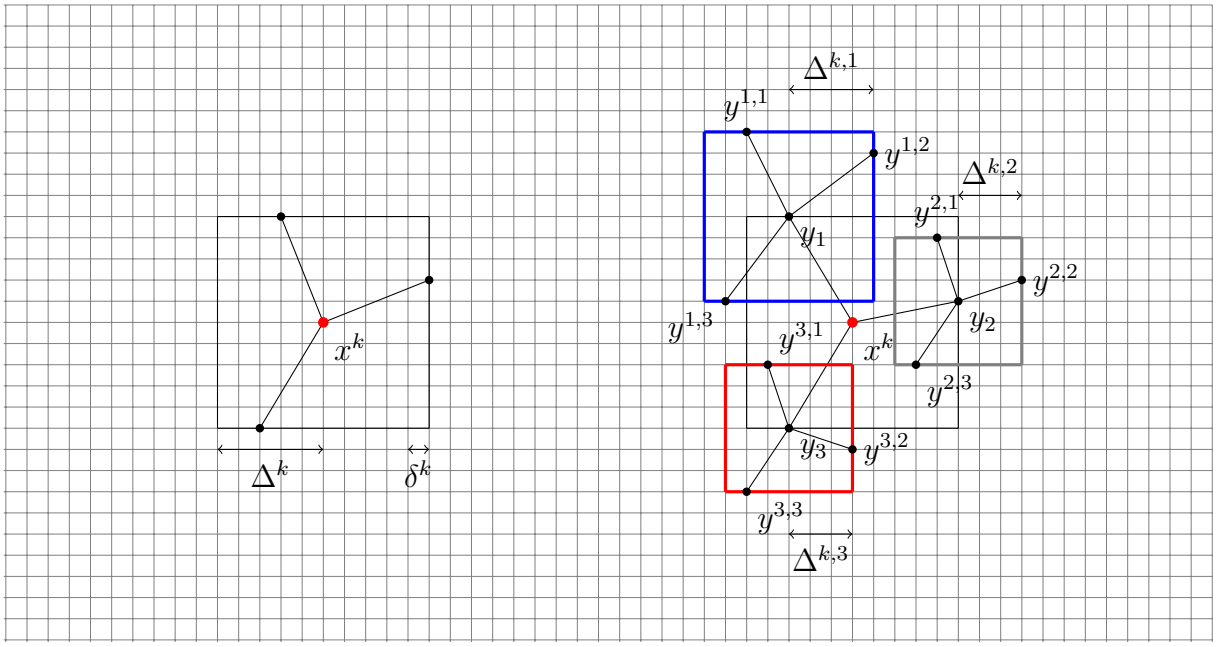


FIGURE 8 – Comparaison sonde locale classique et sonde locale multiple avec le même nombre de directions dans chacune des sonde locale secondaire : $q = q_1 = q_2 = q_3$, et $\Delta^{k,1} = 4\delta^k, \Delta^{k,2} = \Delta^{k,3} = 3\delta^k$

On remarque que conserver des directions de sonde et des tailles de cadre identiques entre la sonde principale et la sonde secondaire pourrait engendrer de nombreux doublons.

L'algorithme 4 fournit la procédure à suivre pour pouvoir générer des points de multi-sonde locale.

Algorithm 4 Multi-sonde locale - Génération de points

[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 $\mathbb{P}$.

La version implémentée dans NOMAD 4, est fixée avec $\Delta^{k,i} = \Delta^k$ et $q = q_i = 2n$. On décide de fixer tous les paramètres de sonde secondaire égaux à Δ^k, ainsi que le nombre de points de sonde éccantillonée 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.

3.2.2 Sonde locale en oignon

Cette stratégie vise à imiter un comportement de MADS lorsque x^k est un minimum local, c'est-à-dire quand les échecs de sonde locale se succèdent. Il est supposé que le succès obtenu à l'étape $k-1$ conduit à une suite d'échecs des étapes $k, k+1, ..., k+c$ d'une étape de sonde locale classique. Par conséquent, on sélectionne des points dans les cadres de sonde locale $F^k, ..., F^{k+c}$ tous centrés en x^{k-1} sur le maillage M^k, dont les paramètres respectifs sont $\Delta^k, \Delta^{k+1}, ..., \Delta^{k+c}$). Ainsi, en supposant que les ressources nécessaires soient disponibles, si chaque étape de sonde locale génère n_p points, cette méthode permet de générer $(c+1)\times n_p$ points. Donc, à supposer que toutes les évaluations nécessitent le même temps (hypothèse non vérifiée en pratique mais c'est pour fixer les idées), 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 la durée des étapes de sonde locale classiques équivalentes effectuées séquentiellement et la durée de la sonde locale en oignon est de c. La figure 9 donne un aperçu d'une sonde locale en oignon pour $c = 2$.

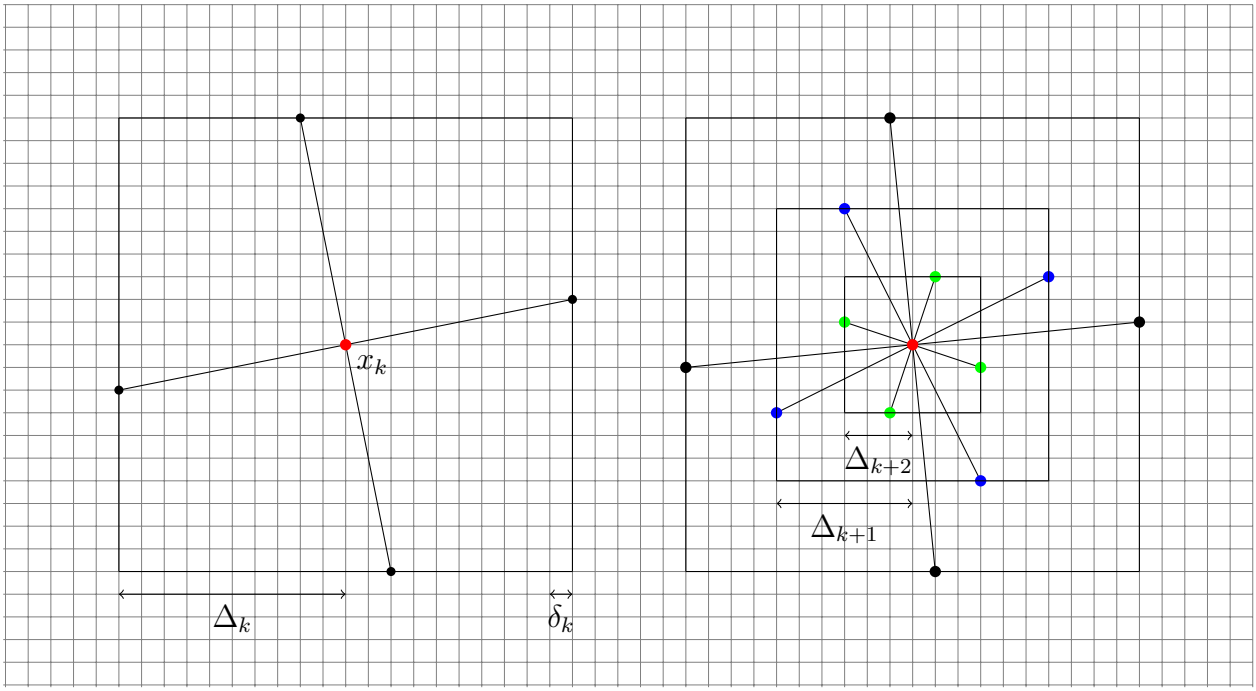


FIGURE 9 – Des directions sont construites sur les cadres de sondes correspondant à deux réductions successives de Δ^k.

L'algorithme 5 donne une description détaillée de la stratégie de sonde locale en oignon

Algorithm 5 sonde locale en oignon - Génération de points

[0]. Initialisation :

δ^k	: Paramètre de maillage.
Δ^k	: Paramètre de cadre.
x^k	: Meilleur candidat.
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 :

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

[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 sonde locale est relativement flexible, les suivants sont implémentés dans NOMAD 3 :

— $n+1$-MADS : Sonde l'espace suivant $n+1$ direction dans le cadre de sonde locale,

— $2n$-MADS : Sonde l'espace suivant $2n$ direction dans le cadre de sonde locale,

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

$$\mathbb{D}^k = \left\{ \pm \delta^k \text{round}\left(\frac{\Delta^k}{\delta^k} \frac{h_i}{\|h_i\|_\infty} \right) \ : \ i \in [\![1, n]\!] \right\} \tag{22}$$

En effet, dans cette définition, $\| \frac{h_i}{\|h_i\|_\infty} \|_\infty = 1$ 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 uniformément distribués dans F^k serait alors 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 donc 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, on peut décider de construire des points de sonde comme sur le deuxième dessin de la figure 10, en conservant les colonnes de H de norme unitaire, ou alors, en multipliant chacune de ces directions par un coefficient aléatoire, comme décrit plus haut, et représenté sur le troisième dessin de la figure 10.

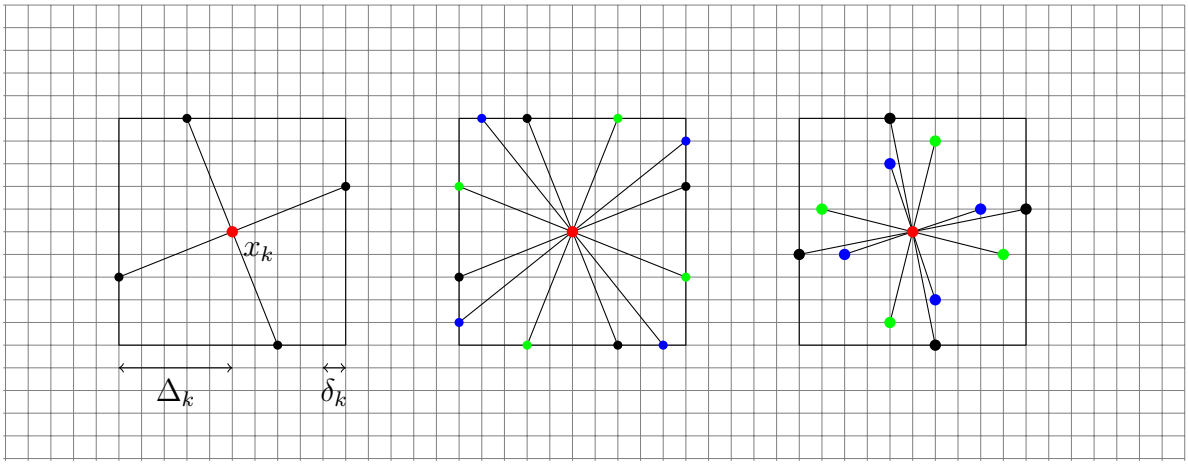


FIGURE 10 – De gauche à droite : Ensemble de sonde locale classique, Ensemble de sonde locale enrichie sur le cadre, Ensemble de sonde locale enrichie sur et à l'intérieur du cadre.

Sur la figure 10, chaque couleur représente un ensemble de directions générées à l'aide d'une direction v^k et de l'opérateur de Householder.

Algorithm 6 sonde locale enrichie - Génération de points

[0]. Initialisation :

Δ^k : Pas de cadre de sonde locale.
δ^k : Pas de maillage.
q : Nombre de bases positives à générer.
x^k : Point donnant lieu au succès courant.
$\mathbb{D} \leftarrow \varnothing$: Ensemble de direction de sonde locale.
$\mathbb{P} \leftarrow \varnothing$: Ensemble de points de sonde locale.

[1]. Génération de direction de sonde locale :

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 $\xi = (\xi_1, ..., \xi_{2n}) \in [0, 1]^{2n}$ 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 sonde locale :

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

Remarquons que la distribution de probabilité qui permet de générer ξ influe donc sur la répartition des points. En effet, si les ξ_i sont échantillonné 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 échantillonné suivant une loi non uniforme, certains zones de F^k seront privilégiées. Enfin, changer les bornes d'échantillonage des ξ_i nous donne accès à des points qui ne sont pas dans le cadre de sonde locale F^k définit dans la section 2.2 si l'on modifie la borne supérieure, ou des points qui sont à une distance minimale de x^k si l'on modifie la borne inférieure. Fixer la borne supérieure d'échantillonnage et la borne inférieur d'échantillonage égales conduit à obtenir plus de points, tous sur la frontière du cadre.

4 Résultats numérique

Les méthodes que nous avons proposées plus haut, nous donnent des règles pour sélectionner des points à l'étape de sonde locale pour occuper les processeurs mis à disposition. Nous cherchons ici à 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'execution de NOMAD .

4.1 Ce que l'on cherche à mesurer

Les méthodes d'analyse comparative proposées dans [33] ne sauraient pas démarquer deux solveurs implémentant le même algorithme (exécutes avec les mêmes paramètres), l'un de manière séquentielle, l'autre de manière parallèle. En effet, ces méthodes d'analyse comparative tendent à évaluer l'efficacité avec laquelle un algorithme fait usage de l'information qu'il tire de chaque évaluation de la fonction objectif.

Pour une exécution séquentielle, un test de convergence permet de connaître le nombre d'évaluations de la boite noire nécessaire à l'algorithme d'intérêt pour considérer qu'il a convergé. Ce test de convergence peut concerner la valeur de Δ_k ou encore la valeur de $f(x_k)$ par rapport à la meilleure valeur connue de la fonction, tous solveurs confondus. Dans ce cas il faut donc avoir pris le temps de mener l'optimisation avec différents solveurs.

Les performances d'une exécution parallèle sont plus subtiles à mesurer. Il faut 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 (language, paradigme de programmation utilisé, etc.). En pratique, lorsqu'on a affaire à un problème industriel, la part de ressources informatique consommées par NOMAD lors de l'optimisation est négligeable devant la part de ressources consommées par les évaluations.

Prenons un exemple : Si l'on mène deux optimisation avec MADS, identiques, sans stratégie opportuniste, à l'exception que sur l'une, les évaluations sont effectuées en série, sur l'autre, les évaluations sont effectuées en parallèle, le seul gain que nous observons ne se compte pas en nombre d'évaluations mais en temps nécessaire à passer au travers des étapes de recherche globale et de sonde locale. Si l'on suppose que l'on effectue toutes les évaluations correspondant à une étape de MADS (recherche globale et sonde locale) avant de passer à l'étape suivante, la parallélisation de MADS ne devrait donc pas influer les performances de l'amélioration de la fonction objectif pendant l'optimisation mais la manière dont sont utilisées les ressources et donc le temps nécessaire à minimiser convenablement la fonction. En effet, à chaque étape, dans les deux cas (parallèle ou séquentiel) on attend que toutes les évaluations soient terminées pour prendre une décision. Cette décision ne dépend que des valeurs retournées par la boite noire, et non pas de leur ordre d'obtention.

De plus, il ne semble pas judicieux de vouloir mesurer l'occupation des processeurs physiques, 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) introduira nécessairement des attentes lors de l'exécution. En pratique, NOMAD 4 pourrait être exécuté sur un seul processeur, car les opérations propre à son fonctionnement sont peu coûteuses en comparaison avec le fonctionnement d'une simulation numérique utilisée comme boite noire.

Pour une exécution, on note f^* la meilleure valeur de f obtenue et x^* tel que $f^* = f(x^*)$ les figures suivantes représentent f^* retourné par NOMAD 4 utilisé avec chacune des stratégies. Notre banc d'essais est construit comme suit :

- 24 problèmes tests analytiques de minimisation sans contraintes tirés de [22]
- Chaque problème est borné inférieurement par 0 et atteint cette valeur (on a arrangé l'expression de la fonction de Weierstrass pour permettre cela en remplaçant le cube par un carré) dans l'ensemble $[-5, 5]^n$. Ainsi, on se fixe comme contrainte de borne $\forall i \in [\![1, n]\!],\ -5 < x_i < 5$.
- Chaque problème peut être généré dans la dimension que l'on souhaite, nous choisissons $n \in \{2, 4, 8, 16, 32, 64\}$.
- Il est possible de générer autant de variantes que l'on souhaite de chaque problème, en fournissant une graine aléatoire différente lors de la construction. Nous en fournissons 5.
- On dispose de quatre stratégies de sonde locale à tester : la sonde locale classique, la multi sonde locale, la sonde locale en oignon et la sonde locale enrichie.
- Tous les pas de cadre de sonde locale secondaires de la multi sonde locale sont égaux au pas de cadre de sonde locale principale ($\forall i \in [\![1, n]\!],\ \Delta^{k,i} = \Delta^k$), et chaque sonde locale principal et secondaires comporte $2n$ directions orthogonales.
- Nous faisons varier le nombre de couches de la sonde locale en oignon de 2 à 8 par pas de 1 pour les problèmes avec $n \leq 8$ et nous étudions éegalement les cas à 16, 32, 64, 128 couches pour les problèmes à $n > 8$. Chaque couche comportant alors $2n$ points générés à l'aide de $2n$ directions orthogonales.
- Nous faisons varier le nombre de blocs de $2n$ points de la sonde locale enrichie de 2 à 8 par pas de 1 pour les problèmes avec $n \leq 8$ et nous étudions éegalement les cas à 16, 32, 64, 128 couches pour les problèmes à $n > 8$. Chaque bloc étant généré à l'aide de directions orthogonales.
- Lors de notre exécution de MADS, seule l'étape de sonde locale est effectuée.
- Lors de chaque étape de sonde locale, tous les points sont évalués (la stratégie opportuniste n'est pas utilisée)
- On effectue 500 itérations au plus (donc 500 étapes de sonde locale). On a également comme critère d'arrêt, la taille minimale du maillage, mesurée par δ^k.

L'environnement numérique des tests est le suivant :

- Une évaluation d'un problème test prend entre 0.001 et 0.00001 s en dimension 1000, et nous faisons des tests jusqu'en dimension 64.
- NOMAD est utilisé sur la grappe de calcul CASIR de l'IREQ, qui possède 152 noeuds, chacun muni de 36 processeurs physiques. Nous éxecutons une optimisation par porcesseur, car au sens de la définition 1, un processeur physique est en fait l'équivalent de plusieurs processeurs pour les problèmes tests utilisés, et il se trouve que l'optimisation s'effectue plus rapidement en séquentiel qu'en parallèle sur les problèmes tests, pour des raisons semblables à celles exposées en section 2.3.
- Le nombre de fils d'exécution OpenMP alloués à NOMAD est réglé à 1, pour les mêmes

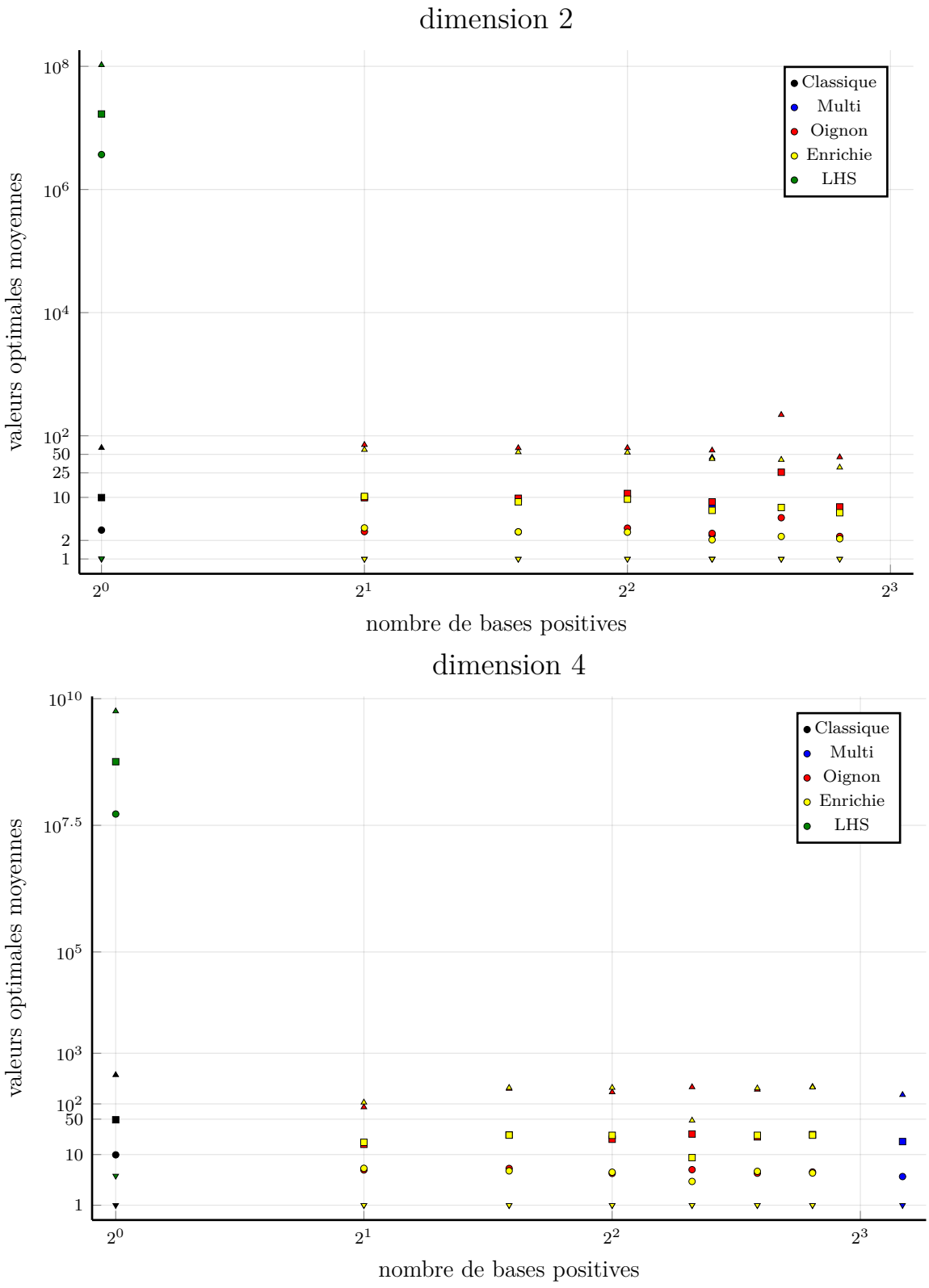
raisons que ci dessus.

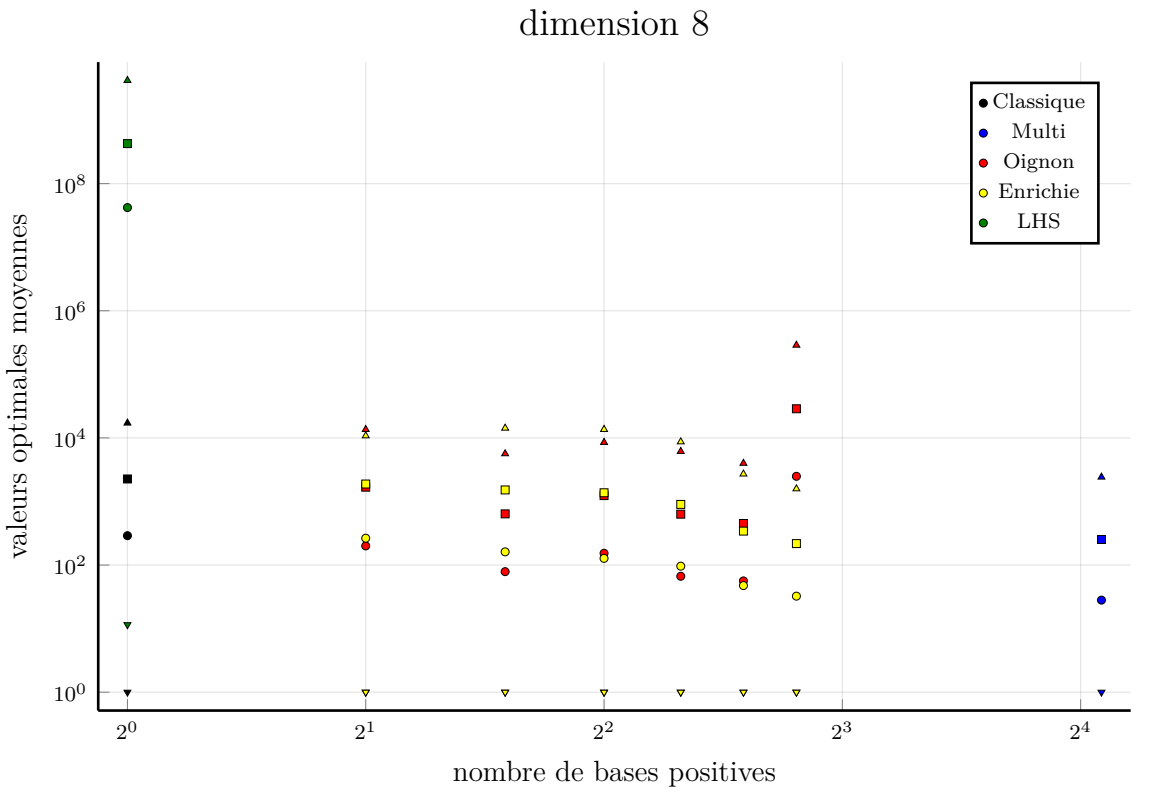
Note sur la lecture des figures : Les figures vont par paire et sont produit à dimension fixée. La figure du haut représente f^* en fonction du nombre de multiples de $2n$ points évalués à chaque étape de sonde locale. La figure du bas réunit moyenne (cercle), moyenne + écart type (carré), maximum (triangle pointant vers le haut), minimum (triangle pointant vers le bas). L'échelle logarithmique est utilisée pour rendre les données plus lisibles, on ajoute alors 1 à toutes les valeurs de f^* obtenues pour les rendre représentables dans le domaine logarithmique, et on est forcé de ne représenter que l'écart type au dessus de la moyenne, l'écart type en dessous de la moyenne donnant lieu à des valeurs négatives. Ainsi, l'écart type est représenté par l'espace entre le point circulaire et le point carré.

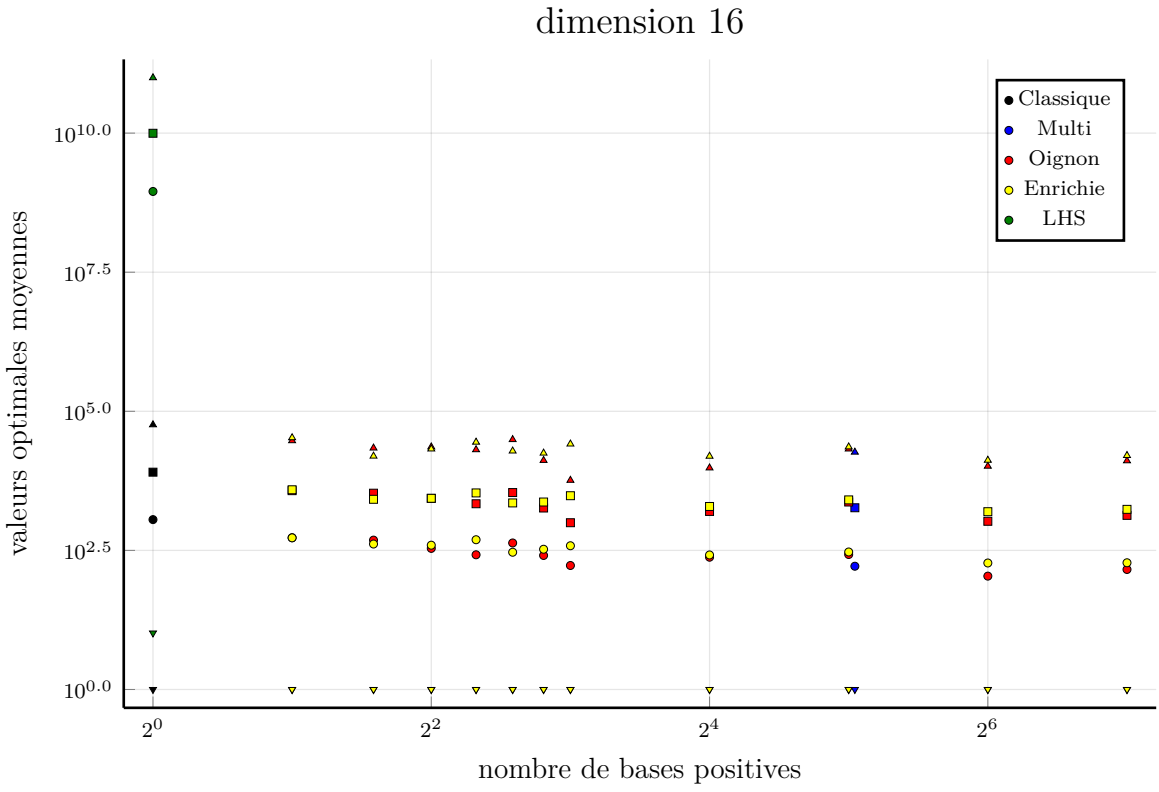
DONE : Faire la même chose mais avec des problèmes réels avec des contraintes comme styrène (j'ai fait l'interface pour Styrène j'ai plus qu'à le faire tourner)+ le cas d'application à l'IREQ (début mars)

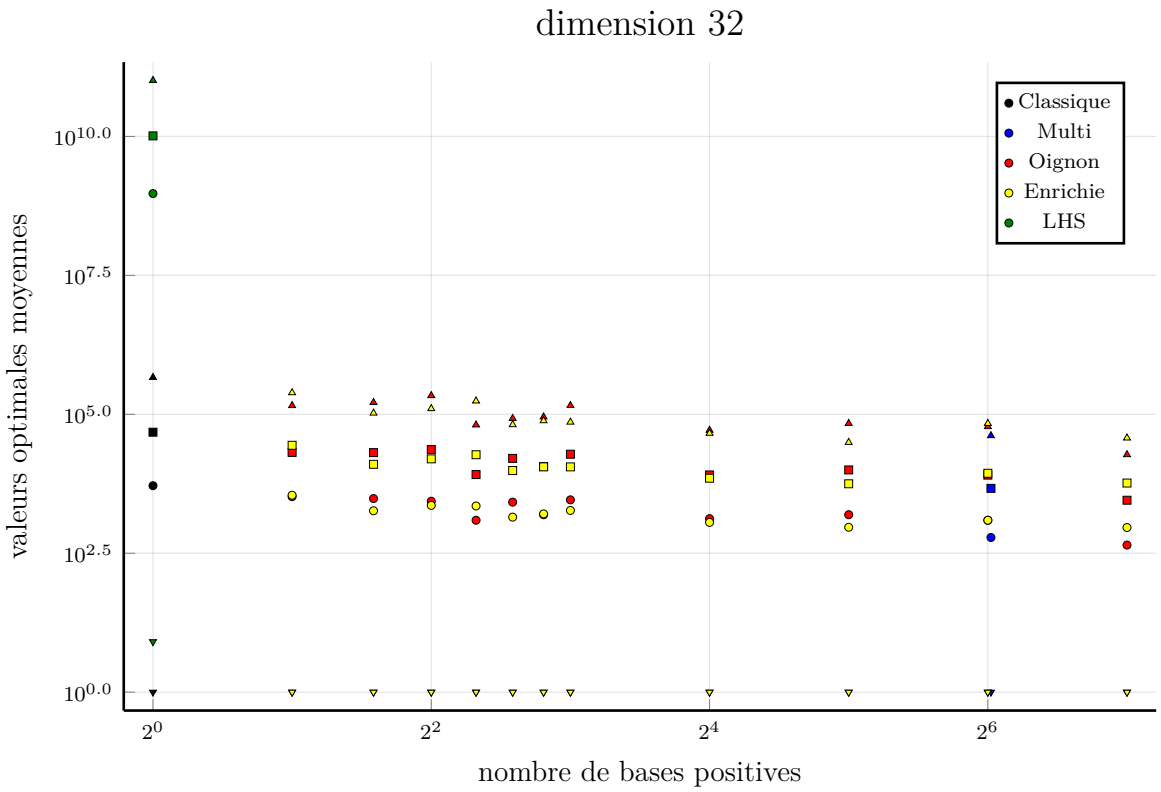
DONE : faire des profiles de performance en temps : les données sont prêtes j'ai plus qu'à coder - à retravailler voir ce qu'on peut en tirer

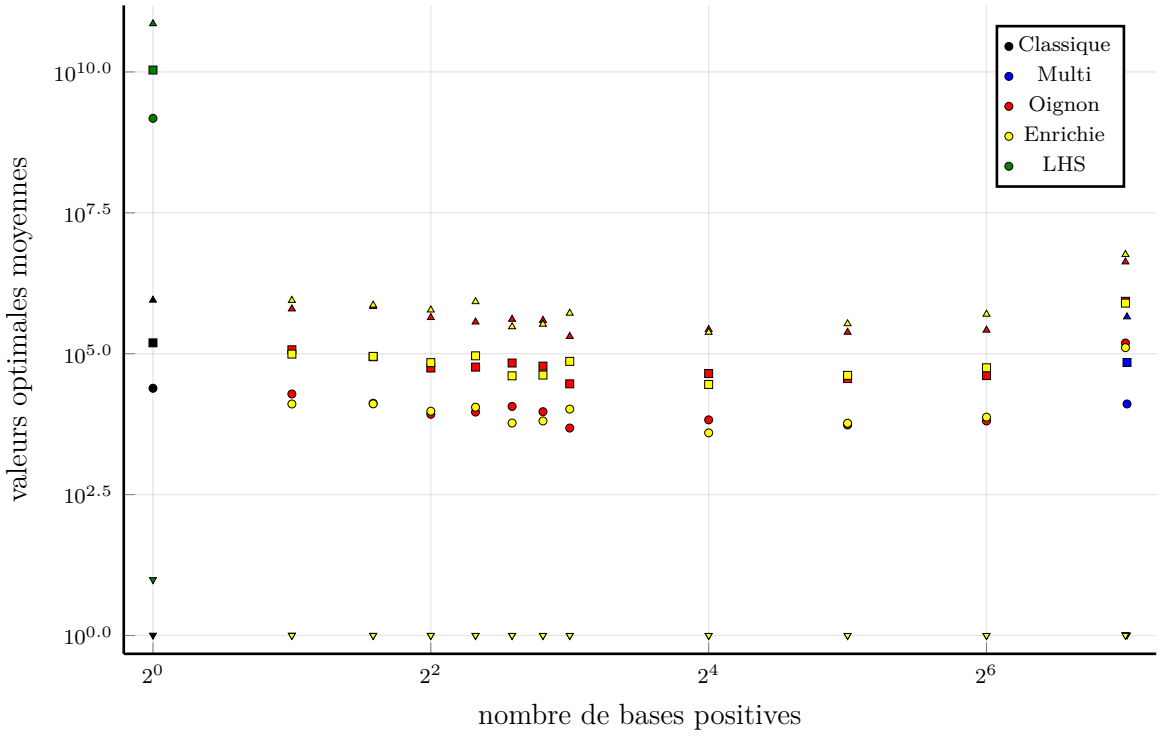
DONE : représenter l'évolution de la fonction objectif par itération pour avoir une idée de la progression de $f(x^k)$: pour chaque itération donner la meilleure valeur de f. - ça donne un paquet de graphes !


dimension 2
valeurs optimales moyennes
nombre de bases positives
Classique
Multi
Oignon
Enrichie
LHS
dimension 4
valeurs optimales moyennes
nombre de bases positives
Classique
Multi
Oignon
Enrichie
LHS


dimension 8
Classique
Multi
Oignon
Enrichie
LHS
valeurs optimales moyennes
10^0
10^2
10^4
10^6
10^8
2^0
2^1
2^2
2^3
2^4
nombre de bases positives


dimension 16
Classique
Multi
Oignon
Enrichie
LHS
valeurs optimales moyennes
$10^{0.0}$
$10^{2.5}$
$10^{5.0}$
$10^{7.5}$
$10^{10.0}$
2^0
2^2
2^4
2^6
nombre de bases positives


dimension 32
Classique
Multi
Oignon
Enrichie
LHS
$10^{10.0}$
$10^{7.5}$
$10^{5.0}$
$10^{2.5}$
$10^{0.0}$
valeurs optimales moyennes
2^0
2^2
2^4
2^6
nombre de bases positives


dimension 64
Classique
Multi
Oignon
Enrichie
LHS
$10^{10.0}$
$10^{7.5}$
$10^{5.0}$
$10^{2.5}$
$10^{0.0}$
valeurs optimales moyennes
2^0
2^2
2^4
2^6
nombre de bases positives

Ces figures ne tiennent compte uniquement de la valeur finale, et ne donnent aucune information sur la manière dont la valeur de la fonction objectif a diminué au cours de chacune des exécutions A creuser :

— comment représenter la tendance d'évolution de la fonction objectif en fonction de la stratégie utilisée, le nombre de points de chaque étape de sonde locale et de la dimension ? pour une dimension donnée et un nombre de bloc de 2n points fixé, faire la moyenne des courbes d'évolution de la fonction objectif par itération : moyenne du pourcentage d'amélioration par itération.

4.2 Ordonnancement optimal de la pile d'évaluation

Soit $\mathcal{P}$ un problème comme défini en 1 et $x \in \mathbb{R}^n$. On note respectivement $\sigma(x)$ et $t(x)$ le travail et le temps nécessaire à un processeur pour effectuer une évaluation de $\mathcal{P}$, c'est à dire, pour construire $e = (x, f(x), c_1(x), ..., c_m(x))$.

Le travail dépend de la nature de la boite noire et de l'intérêt de l'utilisateur, en effet, cela peut être le nombre d'opérations effectuées, l'énergie utilisée, etc.), tandis que le temps est le temps usager, celui que pourrait mesurer l'utilisateur avec sa montre, entre le début et la fin de l'exécution du travail par le processeur.

Comme spécifié dans la définition 1, dans notre contexte, un processeur peut être un téléphone portable, un banc d'essai en laboratoire, un prototype de voiture pour un crash test, le simulateur de réseau électrique d'Hydro-Québec, etc. Et nous possédons un nombre fixé $p \in \mathbb{N}^*$ de ces processeurs pour évaluer les points de $\mathbb{S}^k$ et de $\mathbb{P}^k$ sur $\mathcal{P}$. Sur notre banc d'essais, nous avons, pour tout k :

- $\mathbb{S}^k = \varnothing$, donc $|\mathbb{S}^k| = 0$.
- $|\mathbb{P}^k| = 2n$ dans le cas de la sonde locale classique.
- $|\mathbb{P}^k| = 2n \times (2n + 1)$ dans le cas de la multi sonde locale.
- $|\mathbb{P}^k| = 2n \times b$ dans le cas des sonde locale en oignon ou sonde locale enrichie, avec b désignant le nombre de couche ou de blocs de $2n$ points dans le cas des types de sonde locale correspondant.

Dans tous les cas on pose :

$$n_s = |\mathbb{S}^k| \tag{23}$$

$$n_p = |\mathbb{P}^k| \tag{24}$$

Ce qui suit peut également être écrit pour n_s et $\mathbb{S}^k$. Comme le montre la figure 7, si nous avons $n_p < p$ alors, l'ensemble de processeurs disponibles est nécessairement utilisé en dessous de ses pleines capacités. Les stratégies proposées en section 3.2 on donc pour but de faire en sorte que $n_p > p$. Ainsi, le travail ω^k et la durée τ^k que génère l'étape de sonde locale de l'itération k est :

$$\omega^k = \sum_{x \in \mathbb{P}^k} \sigma(x) \tag{25}$$

$$\tau^k = \sum_{x \in \mathbb{P}^k} t(x) \tag{26}$$

Notons alors T^k la durée de l'étape de sonde locale de l'itération k mesurée par l'utilisateur. Dans un un contexte séquentiel, on a l'égalité :

$$T^k = \tau^k \tag{27}$$

Dans un contexte parallèle avec $n_p \leq p$, on a alors :

$$T^k = \max_{x \in \mathbb{P}^k} t(x) \tag{28}$$

Ainsi, de manière générale, on a :

$$\tau^k \geq T^k \geq \max_{x \in \mathbb{P}^k} t(x) \tag{29}$$

Si nous possédons une fonction oracle permettant d'obtenir les valeurs de $t(x)$ et de $\sigma(x)$ sans évaluer $\mathcal{P}$, il est possible d'ordonner la queue d'évaluations et ainsi d'accélérer l'optimisation tout en économisant du travail. Avec les notations qui précèdent, N itérations de MADS nécessitent $\Theta = \sum_{k=0}^{N} \omega^k$ quantité de travail et $\Gamma = \sum_{k=0}^{N} T^k$ unités de temps. Tout comme le temps, le travail peut être vu comme une quantité consommable. Supposons alors que l'on ait à disposition Θ_0 unités de travail et Γ_0 unités de temps à disposition, et qu'avec le problème $\mathcal{P}$ on nous fournit les fonctions oracles σ et t. Comment mener l'optimisation ? Faut-il juste s'arrêter une fois que l'un des deux budgets est épuisé ? Ou est-il possible d'élaborer une stratégie qui utilise l'information retournée par σ et t ?

Formalisons un peu la figure 7(.)

On appelle Π_s^k et Π_p^k les piles d'évaluations obtenues à l'itération k de MADS pour la recherche globale et la sonde locale. Les objets Π_s^k , Π_p^k et $\mathbb{S}^k$, $\mathbb{P}^k$ sont semblables, on définit alors les fonctions σ et t sur ces quatre ensembles.On cherche alors à partitionner ces piles d'évaluations et à assigner à chaque processeur une partition sur laquelle travailler.

Pour une pile d'évaluation Π, une p-partition de cette pile est une famille $\pi = (\pi_i)_{1 \leq i \leq p}$ de sous ensembles de Π, de sorte que :

$$\forall i \in [\![1, p]\!], \pi_i \subset \Pi \tag{30}$$

$$\forall i, j \in [\![1, p]\!], i \neq j \iff \pi_i \cap \pi_j = \varnothing \tag{31}$$

$$\bigcup_{i=1}^{p} \pi_i = \Pi \tag{32}$$

Avec de l'information supplémentaire sur les évaluations, comme par exemple avec les fonctions oracles σ et t on peut espérer construire les π_i de sorte que le temps utilisateur passé à effectuer les évaluations de Π soit le plus petit possible.

On pose $\phi : P(\Pi) \to \mathbb{R}^+$, vérifiant $\phi(\pi_i) = 0 \iff \pi_i = \varnothing$, pour fixer les idées : $\phi(\pi_i) = \sum_{x \in \pi_i} t(x)$ ou $\phi(\pi_i) = \sum_{x \in \pi_i} \sigma(x)$ ou encore $\phi(\pi_i) = |\pi_i|$. En notant $\coprod$ l'ensemble des p-partitions de Π, une p-partition $\pi^* = (\pi_i^*)_{1 \leq i \leq p}$ ϕ-optimale de Π est une p-partition vérifiant :

$$\max_{i \in [\![1,p]\!]} \phi(\pi_i^*) = \min_{\pi \in \coprod} \left\{ \max_{i \in [\![1,p]\!]} \phi(\pi_i) \right\} \tag{33}$$

Ainsi, si l'on dispose de p processeurs, partitionner Π selon π^* et, pour tout $i \in [\![1, p]\!]$,effectuer les évaluations de π_i sur le processeur i, permet d'écouler la pile d'évaluation en un temps/travail minimal.

Comment formuler les choses lorsque :

- σ et t ne fournissent pas une information exacte, mais avec une erreur aléatoire e_σ et e_t ?
- On se place dans un contexte asynchrone où à chaque pas de temps on ajoute de nouvelles évaluations à la pile Π ==> est-il nécessaire de résoudre le problème posé ci dessus à chaque pas de temps ? Ne peut-on pas en résoudre seulement une partie ?
- les fonctions σ et t dépendent du processeur utilisé ?

Se référer au problème de bin-packing, avec un nombre de boite fixé à p et comme objectif de remplir le moins possible la plus pleine des boites.

4.3 Intensification dynamique lors de l'étape de poll

Aux vues des résultats obtenus plus haut (TODO : inclure les autres graphes), la structure donnée aux points lors de l'étape de sonde locale n'a pas beaucoup d'influence sur la valeur finale, et seule la quantité de points semble avoir un effet notable. Concernant les stratégies dont le nombre de points est modifiable (Oignon et Enrichie), considérons un historique des échecs successifs définit comme suit :

$$e^0 = \begin{cases} 1 & \text{si l'étape de sonde locale à l'itération 0 est un échec} \\ 0 & \text{sinon} \end{cases} \tag{34}$$

$$e^k = \begin{cases} e^{k-1} + 1 & \text{si l'étape de sonde locale à l'itération } k \text{ est un échec} \\ 0 & \text{sinon} \end{cases} \tag{35}$$

Si on considère le cas général :

$$\textsc{ScalablePoll}(\delta^k, \Delta^k, x^k, \min(e^{k-1}, n_p^{max})) = \mathbb{P}^k \tag{36}$$

On peut attribuer aux paramètres c et q des sonde locale en oignon et enrichies respectivement, la valeur de $\min(e^{k-1}, n_p^{max})$, avec n_p^{max} un maximum à ne pas dépasser pour le nombre de bases positives générées. Ainis, lorsque l'étape de sonde locale échoue à améliorer la fonction objectif, «on regarde plus près et plus en détail»

TODO : implémenter ça dans `NOMAD` 4 et le tester

5 travail à venir

— développer l'ordonnancement sur l'information retourné par des fonctions oracle donnant le temps ou la consommation en ressources d'un évaluation.
— fixer le nombre de points de sonde locale en début d'optimisation ne semble pas pertinent. voir pour sonder l'espace avec une intensité variable : plus il y a d'échecs successifs lors de la sonde locale plus il y a de points de sonde locale pour l'itération suivante, et repartir de 2n points lors d'un succès

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

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

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

[11] 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.

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

[13] 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.

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

[15] 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.

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

[17] 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.

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

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

[26] 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.

[27] 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.

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

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

[30] 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.

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

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

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

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

[35] 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.

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

[37] 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.

[38] 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.

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

[40] 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.